Predicting Time to Failure for Large Scale Distributed Systems

Nawanol Theera-Ampornpunt¹, Bowen Zhou¹, Saurabh Bagchi²

¹Department of Computer Science, and ²School of Electrical and Computer Engineering
Purdue University

I. INTRODUCTION

Reliability is of increasing concern in distributed computing systems, which typically depend on multi-million lines of code running on thousands of components. Many such large-scale systems are assembled from commodity hardware and third-party software, which makes the errors more probable due to a lack of understanding of the third-party software affecting the integration or due to defects in the third-party software itself (into which the system owner often has little visibility). As a result, our starting point is that failures are inevitable. We ask the question if it is possible to predict the failure and also importantly, the time to the failure, so that recovery actions, manual or automated, can be triggered.

Driven by the first of our above goals, researchers have put much effort into failure prediction for large-scale distributed systems. Salfner et al. [4] tried to predict if a failure is going to happen within a fixed period of five minutes by using hidden semi-Markov model which allows the inter-state transition probability to be a continuous function of time. Williams et al. [5] measured mean value and standard deviation of critical performance metrics in healthy systems and applied a threshold to detect performance anomalies. If the time between two consecutive anomalies, called dispersion frame, decreases in a speed faster than a preset threshold, an imminent failure would be declared. Cohen et al. [3] applied tree-augmented naïve Bayesian networks to detect failures. Failure forecasting is briefly touched on in the paper as a potential application of their technique. They were trying to answer the same yes or no question here, that is to predict whether there would be a failure. Some of the works explicitly specify the prediction period in which the prediction is valid, but none specify an estimated time to failure. While this is enough for the prediction to be useful, we argue that it is more helpful to know roughly when the failure is going to occur.

When a failure is forthcoming in a large-scale distributed system, there are several mechanisms that can be utilized proactively against the failure, such as redirecting future requests to other servers, performing rejuvenation in some components, taking checkpoints and letting a backup take over, or replacing the soon-to-fail component by a backup component. Each of these actions requires different amount of time to complete. Even within one technique, it may take differing amount of time depending on the dynamic state of the application. For example, rejuvenation with stateful applications will take time dependent on how much state has been built up in the application. If the failure occurs before the mitigation action completes, the user may experience the failure, and important states could be lost. On the other hand, if a lower than actual time to failure is estimated, a more long-running but effective action such as migrating the virtual machines may be unnecessarily avoided. Thus, knowing the time to failure lets the operator choose the appropriate proactive action.

The goal of our project is to enable the above goal with a fine-grained failure prediction technique that provides an estimation of time to failure based on the historical and current records of system performance metrics from multiple sources in the system, such as hardware, operating system, middleware, and the application itself. Both (labeled) normal and failure records are used as input to our method. During the training phase, we divide the traces into classes and train a Hidden Markov Model (HMM) for each class. In the prediction phase, the current window of observations is mapped to the best fit among those HMMs learned from the training data and therefore the corresponding lead time to failure. We evaluated the proposed technique by synthetically injecting a fault in RUBiS [1], an auction site prototype that supports the core functionality of a large-scale auction site such as eBay. Our technique achieved 81.20% overall accuracy in predicting time to failure of RUBiS.

II. METHODOLOGY

Fig. 1. Overview of our method. T denotes the time to the next failure in the system. $T = \infty$ indicates that there is no imminent failure.

We perceive the traces of a given system as a time series of multi-dimensional random variables, thus it is natural to model the system behavior in a Hidden Markov Model (HMM). In a HMM, the relationship between metrics is modeled without any assumption of the underlying distribution. Figure 1 shows the overview of our method, which consists of offline training phase and online prediction phase.

In the offline phase, we first convert normal behavior and failure traces into instances, each of which is a fixed-length
sequence of measurements of performance metrics specific to the target process, such as CPU usage, L2 cache misses, number of threads, disk I/O, etc. The instances are divided into different classes based on their lead time to failure. Table I shows an example of how the instances are divided. A Hidden Markov Model is then trained for each class, using Baum-Welch [2] algorithm.

In the online phase, where the target system is running and we want to predict the failure time, we assume the performance metrics are captured and delivered to our tool by monitoring software such as HP OpenView and IBM Tivoli. We then compute the probability of the current observed sequence of measurements based on the learned HMMs of each class, using the forward-backward algorithm. The class corresponding to the HMM that matches the sequence the best (gives highest probability) is given as the output. Note that the output of our method is the estimated range of the time to failure rather than an exact time. This is enough for system operators or automatic fault recovery mechanisms to decide which action to take regarding the imminent failure.

For our method to be useful, the measurements leading to a failure in the online phase need to be similar to the failure traces used in training in terms of absolute values and patterns. For example, suppose the traces of one specific memory leak are used to train the models and in the online phase, two threads activate this same memory leak at roughly the same time. The rate of memory usage increase would be twice the rate seen during training. In this case, our method will not be able to accurately predict the time to failure, or even predict that there would be a failure at all. On the other hand, if the system encounters an unseen bug that affects metrics in a way we have seen before in training (e.g., another memory leak with similar rate), our method would be able to accurately predict the time to failure.

### III. Evaluation and Preliminary Results

To evaluate our approach, we inject memory leak into Apache Tomcat running RUBiS, which is an auction site prototype modeled after eBay.com [1]. The memory leak rate is 1.06 MB per second, which causes the Java virtual machine to run out of memory in approximately 21 minutes. We use emulated clients to generate realistic workload to the application. The experiment is repeated 30 times, where each run is one hour long and contains one failure. We define failure to be the first point where a wrong result is sent to the user or the reply time exceeds mean + 3*SD duration. For this memory leak injection, the failure point is essentially the same point as the first OutOfMemoryError. The collected traces were divided into classes as described in Table I. The normal behavior instances came from the prefix of traces before the injection of fault. The whole dataset is used as both the training data and testing data.

The overall accuracy of predictions made by our technique is 81.20%. Table II shows the detailed classification results for each class. In particular, the classification between normal and failure is perfect, while the classification among the five failure classes achieves 73.87% accuracy. Furthermore, when our algorithm makes a misclassification, the predicted class tends to be a ‘neighbor’ of the actual class. This means that the predicted time to failure is close to the actual time to failure. With that said, we expect the accuracy of prediction to depend on the granularity of these failure classes. A finer separation of failure classes will lead to more precise predictions of time to failure while a coarser one can give us high accuracy in these predictions. We use an ad-hoc definition of failure classes as shown by Table I for now and leave more sophisticated schemes for future work.

### IV. Conclusion

We proposed a method of determining whether a distributed system is likely to experience a known imminent failure and more importantly, when in the future the failure will happen based on system performance metrics. The initial results are promising, as classification between normal and failure has 100% accuracy, while the estimation of the time to failure achieves 73.87% accuracy. For future research, we would evaluate the method more thoroughly using other failure traces, as well as previously unseen failures that have similar manifestation pattern to a known failure. We also plan to compare the performance of Hidden Markov Model used in this paper with other classification techniques, such as Support Vector Machine.

### REFERENCES


