Predicting Failures of Computer Systems: A Case Study for a Telecommunication System

Felix Salfner, Michael Schieschke and Miroslaw Malek
Institut für Informatik, Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
{salfner|schiesch|malek}@informatik.hu-berlin.de

Abstract

The goal of online failure prediction is to forecast imminent failures while the system is running. This paper compares Similar Events Prediction (SEP) with two other well-known techniques for online failure prediction: a straightforward method that is based on a reliability model and Dispersion Frame Technique (DFT). SEP is based on recognition of failure-prone patterns utilizing a semi-Markov chain in combination with clustering. We applied the approaches to real data of a commercial telecommunication system. Results are presented in terms of precision, recall, F-measure and accumulated runtime-cost. The results suggest a significantly improved forecasting performance.

1. Introduction

Software is becoming the main reason for system failures. As a study by Candea suggests, the cause for downtime shifted significantly from hardware to software over the past 20 years [3]. The main reason for this development is that software complexity is increasing faster than new technologies emerge that try to avoid software failures during design and programming. Therefore, additional mechanisms have to be applied. One promising direction potentially being less expensive than traditional fault tolerance methodologies is to predict the occurrence of failures in order to prevent them or to prepare repair mechanisms for an upcoming failure. We call this approach *proactive fault handling*. In [9] we provide a list of different mechanisms together with a formula to compute steady-state availability for such systems.

This paper is focused on the prediction part of proactive fault handling: the online prediction of failures. It compares our prediction technique called *Similar Events Prediction (SEP)* to two other well-known failure prediction mechanisms: a straightforward prediction based on an exponential reliability model and the dispersion frame technique (DFT) developed by Lin and Siewiorek [8]. SEP is an error pattern recognition algorithm that is based on a semi-Markov chain and clustering. In contrast to the techniques SEP is compared with, it builds on the time of error occurrence plus the information that is contained in the error messages. Comparison of the three techniques is accomplished by application to real data of a complex commercial telecommunication system.

2. Defining Online Failure Prediction

The goal of online failure prediction is to forecast while the system is running whether a failure occurs at some time in the future. This is quite different from reliability prediction in software engineering where it is intended to predict reliability metrics such as failure rates from characteristics of the software or the development process. The main difference between both is that software engineering predictions deal with long-term predictions while online failure prediction deals with comparatively short time intervals and is based on the current system state.

Online (short-term) prediction of eventually upcoming failures is shown in Figure 1. If we perform a prediction at time t we would like to know whether at time $t + \Delta t_l$ a failure will occur or not. We call Δt_l the *lead time*. Δt_l has a lower bound Δt_w, which is called *warning time*: If, for example, a preventive restart of a component should be triggered by failure prediction, Δt_w is the time needed to restart the component. Therefore, the lead time must be at least as long as the warning time. On the other hand, if Δt_l is too large, predictions will be inaccurate.

The prediction period Δt_p describes the length of

1-4244-0054-6/06/$20.00

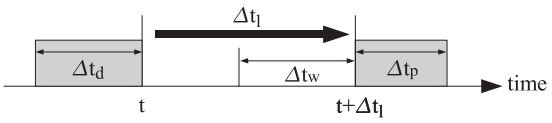

Figure 1. Online Failure Prediction. Definition of lead time (Δt_l), warning time (Δt_w) data window size (Δt_d), and prediction period (Δt_p)

the time interval for which the prediction holds. If Δt_p becomes large, the probability that a failure occurs within Δt_p increases.[1] On the other hand, a large Δt_p limits the use of predictions.

As online failure prediction takes into account the current state of the system, some prediction techniques rely on data that was evaluated earlier. The length of the data window from which data is used is denoted by Δt_d. In the case of DFT, occurrence times of the last five errors are used, therefore, Δt_d is the time interval between the last five errors. Within SEP, Δt_d is constant and determines the maximum length of error patterns.

3. Prediction Techniques

Two principal approaches to failure prediction exist: One class of techniques analyzes periodic measurements of system parameters like workload, memory consumption, etc. Examples are the Multivariate State Estimation Technique (MSET) [10] and trend analysis techniques like [5]. A second class of prediction techniques builds on time series of errors or failures. One well-known example is the Dispersion Frame Technique (DFT) developed by Lin and Siewiorek [8]. A more recent publication by Levy et al. [7] proposes further methods but lacks the ability to predict the time of failure occurrence and details are not yet available to the public. Both approaches use time series of *error* occurrence. Estimation of failure probability from time series of *failure* occurrence is the objective of classical reliability models (see [4] for an overview). SEP extends the second class of prediction algorithms by augmenting time of error occurrence with information that is stored in the error messages.

In this paper we compare predictive accuracy of Similar Events Prediction (SEP) with two approaches of the same class: DFT and a straightforward prediction method that is based on a classical reliability model.

[1] If $\Delta t_p \to \infty$, predicting a failure would always be true!

This section provides a detailed description of SEP and a short recapitulation of the two other prediction techniques.

3.1. Similar Events Prediction (SEP)

The basic assumption behind SEP is that, due to dependencies within a software, special patterns of errors indicate upcoming failures. To find out what the special patterns are, recorded data is used that contains additional information when failures have occurred. In machine learing this is called offline supervised training [1]. The term *training* addresses the process of building a model and tuning the model's parameters. In the case of SEP the model is a semi-Markov chain with enriched state characterization.

After training, the model is used for online failure prediction: During runtime, it is fed with errors that have occurred within the data window of length Δt_d before present time. The time series of errors is analyzed in order to compute the probability that it belongs to a failure-prone pattern. The probability of matching a pattern is combined with the probability that this pattern leads to a failure. Please note that the analyzed series can be part of multiple patterns. All matching patterns are combined to form the output of the model: a failure probability distribution over time. Looking at error logfiles of contemporary business-scale software, it is striking that error messages contain a variety of information, e.g., the software component that has logged the error, the type of error or the depth of the stacktrace. Therefore, error events can be characterized by their timestamp and a set of additional information – the so called *properties* of the event. To deal with the fact that error patterns are almost never fully identical, SEP builds on a notion of similarity between error events that is based on a distance metric. Therefore, similar error events are grouped by *clustering* forming the states of the model.

Description of the model. Each state in the semi-Markov chain encodes restrictions on properties of error events and on their "position" in the event pattern. The semi-Markov chain is determined by three probabilities: for each state i, s_i denotes the initial probability. p_{ij} is the probability that the process transits from state i to state j and the probability distribution $D_{ij}(t)$ describes the duration of the transition. In SEP, $D_{ij}(t)$ are uniform distributions bounded by minimum and maximum delay.

Figure 2 presents a small exemplary model. In order to be able to draw the model in 2D space, error events are only characterized by their timestamp (x-axis) and

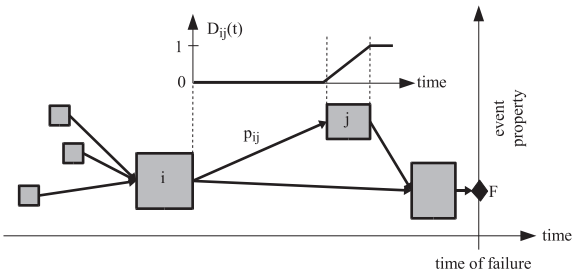

Figure 2. A simple SEP model. p_{ij} denotes the probability that the process transits from state i to state j and $D_{ij}(t)$ denotes the distribution of the transit's duration.

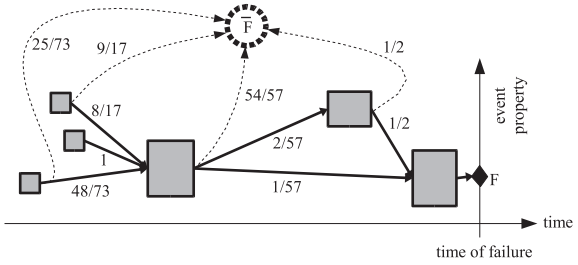

Figure 4. Resulting model after analyzing the complete data set. Error sequences that do not precede a failure leave the pattern at some point and transit to the non-failure state $\overline{F}$.

a single property (y-axis). Since each state in the model puts restrictions on events, and transition durations put bounds on the delay between successive events, the states of the small model can be depicted as boxes.

Training the model. Construction of the model is accomplished in two steps: The first step constructs the states of the semi-Markov chain and defines transition durations D_{ij} while the second step estimates transition probabilities p_{ij}.

In order to determine the states of the semi-Markov chain in the first step, all patterns of errors that lead to a failure have to be identified in the training data set. Each pattern of maximum length Δt_w forms a path of points determined by the properties contained in the error message and the delay between them. Each path ends with a failure. All paths are right-aligned to the time of failure so that they represent the "history" of the training data set before occurrence of a failure. This is shown in Figure 3.

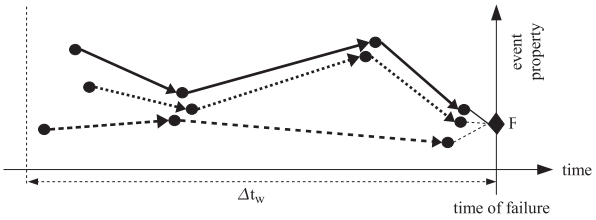

Figure 3. Three error event paths preceding failures – each is right-aligned to the time of failure occurrence

After alignment, a hierarchical clustering algorithm runs "backwards"[2] over the data and groups error events that are similar in their event properties as well as in time-before-failure. Each group forms a state in the semi-Markov chain. The clustering algorithm preserves connections between corresponding states.

The second step of model training aims at estimating the transition probabilities p_{ij}. To achieve this, the whole training data set including error event sequences that do not precede any failure is analyzed. Transition probabilities are determined by the number of error patterns that follow a transition divided by the number of times, the source state of the transition has been reached. Note that sequences that do not result in a failure "leave the path" at some point and transit to a "non-failure state" $\overline{F}$. Figure 4 shows an example.

Predicting failures. The output of SEP online prediction is a curve specifying failure probability over lead-time $P_F(t)$. A failure is predicted if $P_F(t)$ exceeds a threshold $\theta(t)$.

Computation of $P_F(t)$ comprises two parts: estimation of state probabilities π_i and computation of $F_{iF}(t)$, which is the first passage time distribution to the failure state given that the process is in state i at present time:

$$P_F(t) = \sum_i F_{iF}(t)\ \pi_i \quad (t_l \le t \le t_l + \Delta t_p) \qquad (1)$$

$F_{iF}(t)$ is a common quantity in Markov theory that can be computed by

$$F_{iF}(t) = G_{iF}(t) + \sum_{k \neq F} \int_0^t d\, G_{ik}(x) F_{kF}(t-x) \qquad (2)$$

[2]From time of failure occurrence backward in time.

where $G_{ij}(x)$ is the kernel of the semi-Markov process (see [6] for details). Please note that $F_{iF}(t)$ can be pre-computed at training time.

In order to estimate π_i, the time series of error events that have occurred during runtime have to be compared to the patterns of the semi-Markov chain. They have to fit both in event properties as well as in delays between the events. π_i can be computed using a dynamic programming approach by keeping track and updating of the set of possible states and their probabilities. Note that π_i increases towards the failure while maintaining the condition that $\sum_i \pi_i = 1$.

Computational complexity. The most time-consuming parts of SEP are the clustering of error events and computation of $F_{iF}(t)$. Both clustering and computation of $F_{iF}(t)$ can be precomputed during model training and do not affect complexity of online failure prediction, which is more time-critical. Clustering proceeds backwards through the graph and keeps a list of predecessors that have to be checked for grouping. Therefore, most of the nodes are considered several times until they are merged into a state of the semi-Markov chain. Computation of $F_{iF}(t)$ can be time-consuming as it requires order of $\mathcal{O}(n^2)$ operations where n is the number of (clustered) states. But due to the left-to-right structure of the model the transition matrix is sparse and complexity can be reduced. The convolution in Equation 2 can be solved explicitly using Laplace transformations since the kernel G is made up of bounded uniform distributions D_{ij}.

During prediction, π_i'es have to be estimated and failure probabilities $F_{if}(t)$'es have to be computed. Applying a dynamic programming approach the first has maximum complexity $\mathcal{O}(n)$ in the unlikely case that the system might be in all states of the semi-Markov process. Then Equation 1 has to be computed. The complexity is exactly $n-1$. But as the sum consists of scaled probability distributions, complexity is also dependent on the prediction period Δt_p and the granularity of evaluation. Hence, if T denotes the number of time evaluation points, the overall complexity of online prediction is $\mathcal{O}(nT)$.

3.2. Reliability-based prediction

A straightforward method to estimate the probability of failures is to use a standard reliability model since by definition the probability of a failure before time t is

$$F(t) = P[T \leq t] = 1 - R(t) \tag{3}$$

Many models exist that try to estimate $R(t)$ (see, e.g., [4] for an overview). Nevertheless, Brocklehurst and Littlewood [2] have shown, that none of the existing models really get close to reality. This is especially true for online failure prediction because the models are targeted at long-term behavior (in the order of system lifetime) rather than short-term predictions (in the order of minutes or few hours). They take into account properties of the software development and fault removal process, etc. but do not analyze the current state of the system such as, e.g., the number of users that are currently logged in.

In order to compare SEP to a reliability-based prediction technique, we use a standard reliability model that assumes a Poisson failure process and hence approximates reliability by an exponential distribution:

$$F(t) = 1 - e^{-\lambda t} \tag{4}$$

The distribution is fit to the short-term behavior of the system by setting the failure rate to:

$$\lambda = \frac{1}{MTTF} \tag{5}$$

where MTTF is mean-time-to-failure of the data set that is used to assess the predictive accuracy of the technique. This yields an optimistic assessment of predictive accuracy since in a real environment, MTTF of the training data set will be different from MTTF of the production system.

Using this model, a failure is predicted according to the median of the distribution:

$$T_p = \frac{1}{\lambda} ln(2) \tag{6}$$

After each failure that occurs in the test data set, the timer is reset and the next failure is predicted at time T_p in the future.

3.3. Dispersion Frame Technique (DFT)

DFT is a well-known heuristic approach to analyze a trend in error occurrence frequency developed by Lin and Siewiorek [8]. In their paper, the authors have shown that DFT is superior to classic statistical approaches like fitting of Weibull distribution shape parameters. We use DFT as a reference prediction technique that analyzes *time of error occurrence*.

A Dispersion Frame (DF) is the interval time between successive error events. Therefore, Δt_d is the sum of the last four DFs. The Error Dispersion Index (EDI) is defined as the number of error occurrences in the later half of a DF. A failure is predicted if one of five heuristic rules fires that account for several system behaviors. For example, one rule puts a threshold on

error-occurrence frequencies, another one on window-averaged occurrence frequency. Yet another rule fires if the EDI is decreasing for four successive DFs and at least one is half the size of its previous frame.

3.4. Comparison of the techniques

The three prediction techniques can be ordered according to the extent how much information from the running system is used. Reliability-based prediction relies only on time of *failure* occurrence. DFT analyzes the time of *error* occurrence, which means that much more data is analyzed. In contrast to only analyzing the time when errors have occurred, SEP additionally exploits the information that is contained in the error message itself. For example, the type of an error or the name of the process that caused the error can also be taken into account.

It is not surprising that the computational complexity of the prediction techniques increases along with the amount of information that is analyzed. To perform online prediction, the reliability-based algorithm consumes almost no computational power while SEP is the most costly prediction algorithm of the three being linear in the number of states of the semi-Markov chain: For every error message the set of possible states has to be updated and the failure-passage distributions have to be looked up. But as Section 6 will show, this additional complexity pays off by significantly improved prediction quality.

The same ordering holds for the complexity of setting up the prediction techniques. While reliability-based prediction requires only the estimation of λ, some more parameters have to be determined for DFT. SEP involves heavy computations during the learning phase: Error patterns have to be clustered, transition probabilities have to be estimated from a training data set and failure-passage distributions have to be computed, which has quadratic complexity in terms of the number of clustered states.

4. Metrics

In order to investigate the quality of online failure prediction a test dataset is used for which the occurrence of failures is known. A perfect failure prediction would achieve a one-to-one matching between predicted and actual failures. Several metrics exist to measure the goodness of fit for prediction algorithms. We focused on metrics that have an intuitive interpretation: precision, recall, F-measure and accumulated runtime-cost.

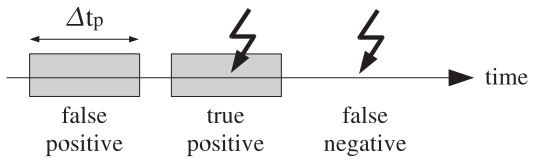

Figure 5. Classification of failure prediction situations. Gray boxes indicate failure predictions and arrows actual failures.

Figure 5 defines three cases: A failure prediction is a *true positive* if a true failure in the test data occurs within the prediction period Δt_p of a failure prediction. If no failure occurs, the prediction is a *false positive*. If the failure predictor missed to predict a true failure, it is a *false negative*.

4.1. Precision, Recall and F-Measure

Precision and recall, originally defined to evaluate information retrieval strategies, are frequently used to express classification quality. They have been used in similar studies [13] and can also be utilized to compute steady state system availability [9].

Precision is the ratio of the number of correctly identified failures to the number of all positive predictions:

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \quad (7)$$

Following [12], we use discounted false positives.

Recall is defined as the ratio of the number of correctly predicted failures to the total number of failures that actually occurred. Recall is sometimes also called true positive rate.

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \quad (8)$$

Consider the following example for clarification: If a prediction algorithm achieves precision of 0.8, the probability is 80% that any generated failure warning is correct (refers to a true failure) and 20% are false positives. A recall of 0.9 expresses that 90% of all actual failures are predicted (and 10% are missed).

There is often an inverse proportionality between high recall and high precision. Improving recall mostly lowers precision and vice versa. A widely used metric integrating the trade-off between precision and recall is the F-measure [11]:

$$F - measure = \frac{2 * precision * recall}{precision + recall} \in [0, 1] \quad (9)$$

4.2. Accumulated Runtime-Cost

In many environments not all false predictions have the same consequences. For example, in some application it might be tolerable to have false alarms while a system failure is very costly. Accumulated runtime-cost graphs account for that.

To produce an accumulated cost graph, costs for true positives, false positives and false negatives have to be assigned. Then a system run is analyzed and total runtime costs are plotted as they accumulate over time. One advantage of the accumulated cost graph is that there is no averaging in place as it is the case with precision and recall. Therefore, the accumulated cost graph may, e.g., reveal whether false positives occurred uniformly distributed over the entire run or if the prediction algorithm performed really well most of the time except for a short period where hundreds of false positives occurred. This information cannot be revealed from precision and recall.

A drawback of accumulated runtime-cost graphs is that the cost of false positives, true positives and false negatives can be chosen arbitrarily, which can alter the graph significantly. We tried to use a realistic setting where a true positive costs 10 units, false positive costs 20 units and the price for false negatives was 100 units.

5. Experiment Description

All three prediction techniques presented in this paper have been applied to data of a commercial telecommunication system. The main characteristics of the software platform are its distributed component-based software architecture running on top of a full-featured container runtime environment. Specification of failures is usually application specific. In our case, a failure is defined as the situation if within five minutes more than 0.01% of calls exhibit a response time over 250ms, which falls into the class of performance failures. In order to find out, when failures occurred during the tests (to label the data sets) we had access to logs of an external stress generator keeping track of both the call load put onto the platform and the time when failures occurred. See Figure 6.

We used error-logfiles of two one-day runs recorded from tests that are applied immediately before customer delivery. Data of one day were used for training of the models and evaluations were performed with data of the second day. The amount of data is admittedly limited and not sufficient for a full-fledged analysis of SEP but gives a first impression of its prediction potential. The large variety of logged information includes 55 different, partially non-numeric variables of several architectural layers. The amount of log data per time unit varies from 2 to 30.000 log records per hour. Figure 7 shows number of errors per 100 seconds and the occurrence of failures in the test data.

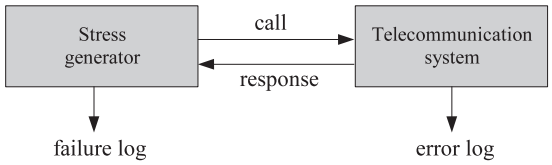

Figure 6. Experimental setup

Training data contained 354 errors and 22 failures whereas test data had 616 errors and 22 failures.

6. Results

Average predictive accuracy[3] is reported in terms of precision and recall while a more detailed behavior is shown by an accumulated runtime-cost graph. All results are computed for a warning-time Δt_w of 30 seconds resulting in an elimination of predictions with a lead-time of less than 30 seconds. The output of SEP is a failure probability distribution over time, hence SEP can predict failures with a lead-time Δt_l from zero up to the maximum pattern length Δt_d. In our studies, a lead-time of 1 minute was chosen. The two other prediction techniques show varying lead-times. The prediction period Δt_p was set to 60 seconds.

6.1. Precision, Recall and F-Measure

Values for precision, recall and F-measure are reported in Figure 8. SEP clearly outperforms the two comparative approaches. We think that this is due to the exploitation of more fine-grained information contained in the error logs. Of course, this implies higher computational overhead. However, our implementation of the SEP algorithm, which is not performance optimized, was able to compute predictions of the whole-day test run within 20 minutes on a contemporary workstation. It should also be mentioned that values for the reliability-based prediction are optimistic estimates since the parameter λ was estimated from the same data the tests were conducted with. The results for DFT refer to a set of parameters where all parameters have been optimized separately. If two choices were almost equal in precision and recall we took the one with less false positives.

[3]Not meant in the strict sense of the definition: true predictions vs. all predictions

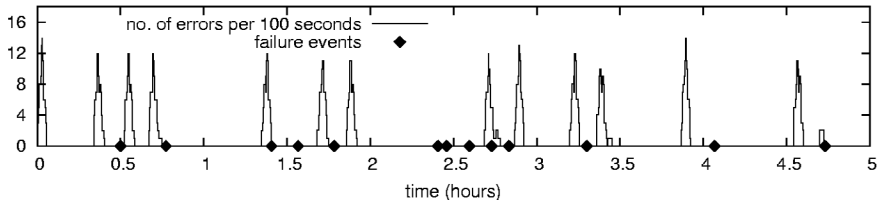

Figure 7. Number of errors per 100 seconds in test data. Diamonds indicate occurrence of failures.

Prediction technique	Precision	Recall	F-Measure
Similar Event Prediction	0.800	0.923	0.8571
DFT	0.389	0.538	0.4515
reliability-based	0.214	0.154	0.1791

Figure 8. Precision, recall and F-measure of the three prediction algorithms applied to the test data.

6.2. Accumulated Runtime-Cost

Figure 9 shows accumulated runtime-cost for the telecommunication system. In order to prevent that details of the cost plot are obscured by too many data points, we used only a part of the test dataset for the accumulated runtime-cost graph. The plot shows a run of five hours containing 13 failures. For comparison, we added two curves: "perfect prediction" and "no prediction". The first refers to the case where a "perfect failure predictor" would predict all failures without any mispredictions. With such a predictor in place, each upcoming failure would be prevented causing minimum cost. The curve "no prediction" refers to a system with neither a predictor nor any reaction schemes in place: Each time a failure occurs, costs increase by 100, which is the price for false negatives.

SEP outperforms all prediction techniques and is close to the cost curve of perfect prediction – only missing to predict one single failure. As could have been expected from precision and recall, DFT performs worse than SEP but better than reliability-based prediction. It is also noteworthy that reliability-based prediction is quite close to the reference curve for a system without prediction suggesting that it cannot be used to enhance system availability.

7. Conclusion and Future Work

Similar Events Prediction (SEP) is an approach to online failure prediction that is based on the recognition of failure-prone patterns of error events. In this paper we compared SEP to two failure prediction techniques of the same class that evaluate event logs such as error or failure logs. SEP was compared to a straightforward prediction method based on a well-known reliability model and to the Dispersion Frame Technique (DFT) by Lin and Siewiorek. The first is based on reliability approximation by a Poisson failure process predicting failures at the median of the resulting exponential distribution. DFT is an approach exploiting the times of error occurrence applying heuristic rules for prediction.

All three models have been applied to data of a complex commercial telecommunication system. Predictive power of the approaches is compared in terms of precision, recall, F-measure and accumulated runtime-costs. SEP outperformed the other failure prediction techniques in all measures. It achieved a precision of 80% and recall of 92% which is strongly supporting the thesis that a proactive handling of faults has the potential to improve system availability up to an order of magnitude.

The main reason for achieving better predictive power is most likely the fact that SEP takes into account more information about the current system state. In addition to analyzing solely the time of occurrence, SEP investigates properties of the error event itself such as the type of error message, the software component that reported the event or the depth of the stack trace. However, taking into account more information has to be paid for in terms of computational complexity.

Comparing the results of DFT with the original work by Lin and Siewiorek, performance was worse. The main reason for that seems to be the difference of investigated systems: While the original paper investigated failures in the Andrews distributed File System based on the occurrence of host errors, our study applied the

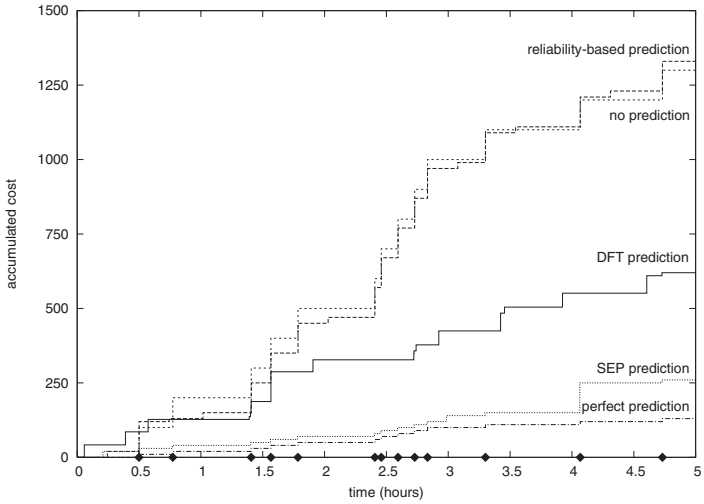

Figure 9. Accumulated cost for each technique for a five-hour runtime. Cost of 10 has been assigned to true positives, 20 to false positives and 100 to false negative predictions. Diamonds indicate failures which have occurred in the data set.

technique to errors that had been reported by software components in order to predict upcoming performance failures.

Our study used error data of a one-day test run. However, computational complexity of training grows heavily if the amount of training data increases. Limiting the growth will be the major challenge for future work. Additionally, sensitivity of SEP with respect to lead-time and changing system configuration as well as root cause analysis will be investigated.

References

[1] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[2] S. Brocklehurst and B. Littlewood. New ways to get accurate reliability measures. *IEEE Software*, 9(4):34–42, 1992.

[3] G. Candea, M. Delgado, M. Chen, and A. Fox. Automatic failure-path inference: A generic introspection technique for internet applications. In *Proceedings of the 3rd IEEE Workshop on Internet Applications (WIAPP)*, San Jose, CA, Jun. 2003.

[4] W. Farr. Software reliability modeling survey. In M. R. Lyu, editor, *Handbook of software reliability engineering*, chapter 3, pages 71–117. McGraw-Hill, 1996.

[5] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi. A methodology for detection and estimation of software aging. In *Proceedings of the Int'l. Symp. on Software Reliability Engineering, ISSRE 1998*, Nov. 1998.

[6] V. G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman and Hall, London, UK, first edition, 1995.

[7] D. Levy and R. Chillarege. Early warning of failures through alarm analysis - a case study in telecom voice mail systems. In *ISSRE '03: Proceedings of the 14th International Symposium on Software Reliability Engineering*, Washington, DC, USA, 2003. IEEE Computer Society.

[8] T.-T. Y. Lin and D. P. Siewiorek. Error log analysis: statistical modeling and heuristic trend analysis. *IEEE Transactions on Reliability*, 39(4):419–432, Oct. 1990.

[9] F. Salfner and M. Malek. Proactive fault handling for system availability enhancement. In *IEEE Proceedings of the DPDNS Workshop in conjunction with IPDPS 2005*, Denver, CO, 2005.

[10] R. M. Singer, K. C. Gross, J. P. Herzog, R. W. King, and S. Wegerich. Model-based nuclear power plant monitoring and fault detection: Theoretical foundations. In *Proceedings of Intelligent System Application to Power Systems (ISAP 97)*, pages 60–65, Seoul, Korea, Jul. 1997.

[11] C. Van Rijsbergen. *Information Retrieval*. Butterworth, London, second edition, 1979.

[12] G. Weiss. Timeweaver: A genetic algorithm for identifying predictive patterns in sequences of events. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 718–725, San Francisco, CA, 1999. Morgan Kaufmann.

[13] G. Weiss. Predicting telecommunication equipment failures from sequences of network alarms. In *Handbook of Knowledge Discovery and Data Mining*, pages 891–896. Oxford University Press, 2002.