A Survey of Online Failure Prediction Methods

FELIX SALFNER, MAREN LENK, and MIROSLAW MALEK

Humboldt-Universität zu Berlin

With the ever-growing complexity and dynamicity of computer systems, proactive fault management is an effective approach to enhancing availability. Online failure prediction is the key to such techniques. In contrast to classical reliability methods, online failure prediction is based on runtime monitoring and a variety of models and methods that use the current state of a system and, frequently, the past experience as well. This survey describes these methods. To capture the wide spectrum of approaches concerning this area, a taxonomy has been developed, whose different approaches are explained and major concepts are described in detail.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: *Reliability, availability, and serviceability; Fault tolerance*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Error handling and recovery*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Error, failure prediction, fault, prediction metrics, runtime monitoring

ACM Reference Format:

Salfner, F., Lenk, M., and Malek, M. 2010. A survey of online failure prediction methods. ACM Comput. Surv. 42, 3, Article 10 (March 2010), 42 pages.
DOI = 10.1145/1670679.1670680 http://doi.acm.org/10.1145/1670679.1670680

1. INTRODUCTION

Predicting the future has fascinated people from the beginning of time. Several millions of people work on prediction daily: astrologers, meteorologists, politicians, pollsters, stock analysts, and doctors, as well as computer scientists and engineers. As computer scientists, we focus on the prediction of computer system failures, a topic that has attracted interest for more than 30 years. However, what is understood by the term *failure prediction* varies among research communities and has also changed over the decades.

As computer systems are growing more and more complex, they are also changing dynamically due to the mobility of devices, changing execution environments, frequent updates and upgrades, online repairs, the addition and removal of system components, and the system/network complexity itself. Classical reliability theory and conventional

This research was supported in part by Intel Corporation and by German Research Foundation.

Authors' addresses: Department of Computer Science, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany; email: {salfner,lenk,malek}@informatik.hu-berlin.de.

DOI 10.1145/1670679.1670680 http://doi.acm.org/10.1145/1670679.1670680

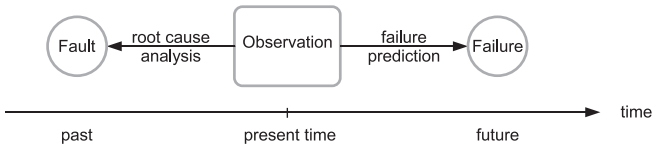

Fig. 1. Distinction between root cause analysis and failure prediction.

methods rarely consider the actual state of a system and are therefore not capable of reflecting the dynamics of runtime systems and failure processes. Such methods are typically useful in design for long-term or average behavior predictions and comparative analysis.

The motto of research on online failure prediction techniques can be well expressed by the words of the Greek poet C. P. Cavafy, who said [Cavafy 1992, page 99]:

> Ordinary mortals know what's happening now, the gods know what the future holds because they alone are totally enlightened. Wise men are aware of future things just about to happen.

For ordinary mortals, predicting the near-term future is more clever and frequently more successful than attempting long-term predictions. Short-term predictions are especially helpful to prevent potential disasters or to limit the damage caused by computer system failures. Allowing for the dynamic properties of modern computer systems, online failure prediction incorporates measurements of actual system parameters during runtime in order to assess the probability of failure occurrence in the near future in terms of seconds or minutes.

1.1. Focus of This Survey

In computer science, prediction methods are used in various areas. For example, branch prediction in microprocessors tries to prefetch instructions that are most likely to be executed, or memory or cache prediction tries to forecast what data might be required next. Limiting the scope to failures, there are several areas where the term *prediction* is used. For example, in reliability theory, the goal of reliability prediction is to assess the future reliability of a system from its design or specification. Lyu [1996], and especially the chapters Farr [1996] and Brocklehurst and Littlewood [1996], provide a good overview, while Musa et al. [1987] and Blischke and Murthy [2000] cover the topic comprehensively. Denson [1998] gives an overview of reliability prediction techniques for electronic devices. However,

> the topic of this survey is to identify *during runtime* whether a failure will occur in the *near future* based on an assessment of the *monitored current system state*. Such type of failure prediction is called *online failure prediction*.

Although architectural properties such as interdependencies play a crucial role in some prediction methods, online failure prediction is concerned with a *short-term* assessment of whether or not there will be a failure, for example, in 5 minutes. Prediction of systems reliability, however, is concerned with long-term predictions based on, for example, failure rates, architectural properties, or the number of bugs that have been fixed.

Online failure prediction is frequently confused with *root cause analysis*. Having observed some misbehavior in a running system, *root cause analysis* tries to identify the fault that caused it, while *failure prediction* tries to assess the risk that the misbehavior will result in future failure (see Figure 1). For example, if it is observed that a database is not available, root cause analysis tries to identify what the reason for the unavailability

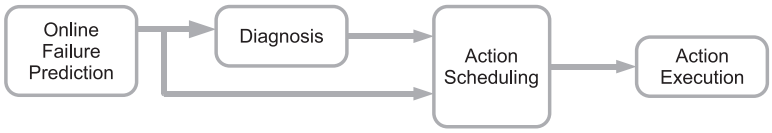

Fig. 2. The steps involved in proactive fault management. After prediction of an upcoming failure, diagnosis might be required in order to find the fault that causes the upcoming failure. Failure prediction and/or diagnosis results are used to decide which proactive method to apply and to schedule their execution.

is: a broken network connection, or a changed configuration, etc. Failure prediction, on the other hand, tries to assess whether this situation bears the risk that the system cannot deliver its expected service, which may depend on system characteristics, the failure prediction model, and the current situation: is there a backup database or some other fault tolerance mechanism available? What is the current load of the system? This survey focuses on failure prediction only.

1.2. The Big Picture: Proactive Fault Management

When both industry and academia realized that traditional fault tolerance mechanisms could not keep pace with the growing complexity, dynamics, and flexibility of new computing architectures and paradigms, they set off in search for new concepts, as can be seen from initiatives and research efforts on autonomic computing [Horn 2001], trustworthy computing [Mundie et al. 2002], adaptive enterprise [Coleman and Thompson 2005], recovery-oriented computing [Brown and Patterson 2001], and various conferences on self-*properties where the asterisk can be replaced by any of the terms *configuration*, *healing*, *optimization*, or *protection* (see, e.g., Babaoglu et al. [2005]). Most of these terms span a variety of research areas ranging from adaptive storage to advanced security concepts. One of these areas is concerned with the task of determining how computer systems can proactively handle failures: if the system knows about a critical situation in advance, it can try to apply countermeasures in order to prevent the occurrence of a failure, or it can prepare repair mechanisms for the upcoming failure in order to reduce time-to-repair. In analogy to the term *fault tolerance*, we use *proactive fault management* as an umbrella term for these techniques.

Proactive fault management consists basically of four steps (see Figure 2):

(1) In order to identify failure-prone situations, that is, situations that will probably evolve into a failure, *online failure prediction* has to be performed. The output of online failure prediction can either be a binary decision or some continuous measure judging the current situation as more or less failure-prone.

(2) Even though techniques such as checkpointing can be triggered directly by a binary failure prediction algorithm, further *diagnosis* is required in many other cases. The objective is, depending on the countermeasures that are available in the system, to find out where the error is located (e.g., at which component) or what the underlying fault is. Note that in contrast to traditional diagnosis, in proactive fault management diagnosis is invoked by failure prediction, that is, when the failure is imminent but has not yet occurred.

(3) Based on both the outcome of online failure prediction and/or diagnosis, a decision needs to be made which of the actions, that is, the countermeasures, should be applied and when it should be executed in order to remedy the problem. This step is termed *action scheduling*. These decisions are based on an objective function taking into account the cost of the actions, the confidence in the prediction, and the effectiveness and complexity of the actions in order to determine the optimal

tradeoff. For example, in order to trigger a rather costly technique, the scheduler should be almost sure about an upcoming failure, whereas for a less expensive action less confidence in the correctness of failure prediction is required. Candea et al. [2004] have examined this relationship quantitatively. They showed that short restart times (microreboots) allow for a higher false positive rate in comparison to slower restarts (process restarts).[1] Many emerging concepts such as the policies used in IBM's autonomic manager relate to action scheduling as well.

(4) The last step in proactive fault management is the actual *execution of actions*. Challenges for action execution include online reconfiguration of globally distributed systems, data synchronization of distributed data centers, and many more.

In summary, accurate online failure prediction is only the prerequisite in the chain and each of the remaining three steps constitutes a whole field of research of its own. Not devaluing the efforts that have been made in the other fields, this survey provides an overview of online failure prediction.

In order to build a proactive fault management solution that is able to boost system dependability by up to an order of magnitude, the best techniques from all four fields for the given surrounding conditions have to be combined. However, this requires *comparability* of approaches, which can only be achieved if two conditions are met:

—a set of standard quality evaluation metrics is available, and

—publicly available reference data sets can be accessed.

Regarding reference data sets, a first initiative was started in 2006 by Carnegie Mellon University, called the Computer Failure Data Repository,[2] that publicly provides detailed failure data from a variety of large production systems such as high-performance clusters at the Lawrence Livermore National Laboratory.

Regarding standard metrics, this survey provides the first step by presenting and discussing major metrics for the evaluation of online failure prediction approaches.

1.3. Outline

This article is a survey on failure prediction methods that have been used to predict failures of computer systems online, that is, those based on the current system state. Starting from a definition of the basic terms such as *errors*, *failures*, and *lead time* (Section 2), established metrics to investigate the quality of failure prediction algorithms are reviewed in Section 3. In order to structure the wide spectrum of methods, a taxonomy is introduced in Section 4 and almost 50 online failure prediction approaches are surveyed in Section 5. A comprehensive list of all failure prediction methods together with demonstrated and potential applications is provided in the summary and conclusions (Section 6). In order to give further insight into online failure prediction approaches, selected representative methods are described in greater detail in the electronic appendix available in the ACM Digital Library. In Appendix A a table of the selected methods is provided, and in Appendices B–K the techniques are discussed.

[1]Although in Candea et al. [2004] false positives relate to falsely suspecting a component to be at fault, similar relationships should hold for failure predictions too.

[2]`http://cfdr.usenix.org`.

2. DEFINITIONS

The aim of online failure prediction is to predict the occurrence of failures during runtime based on the current system state. The following sections provide more precise definitions of the terms used throughout this article.

2.1. Faults, Errors, Symptoms, and Failures

Several attempts have been made to get to a precise definition of faults, errors, and failures, among which are Melliar-Smith and Randell [1977]; Avižienis and Laprie [1986]; Laprie and Kanoun [1996]; and IEC: International Technical Comission [2002]; Siewiorek and Swarz [1998], page 22; and most recently Avižienis et al. [2004]. Since the latter seems to have broad acceptance, its definitions are used in this article, with some additional extensions and interpretations.

—A *failure* is defined as "an event that occurs when the delivered service deviates from correct service." The main point here is that a failure refers to misbehavior that can be observed by the user, which can either be a human or another computer system. Things may go wrong inside the system, but as long as the problem does not result in incorrect output (including the case where there is no output at all) there is no failure.

—The situation when "things go wrong" in the system can be formalized as the situation when the system's state deviates from the correct state, which is called an *error*. Hence, "an error is the part of the total state of the system that may lead to its subsequent service failure."

—Finally, *faults* are the adjudged or hypothesized cause of an error—the root cause of an error. In most cases, faults remain dormant for some time and, once they become active, they cause an incorrect system state, which is an error. That is why errors are also called *manifestations* of faults. Several classifications of faults have been proposed in the literature, among which the distinction between transient, intermittent, and permanent faults [Siewiorek and Swarz 1998, page 22] is best known.

—The definition of an error implies that the activation of a fault leads to an incorrect state; however, this does not necessarily mean that the system knows about it. In addition to the definitions given by Avižienis et al. [2004], we distinguish between *undetected errors* and *detected errors*: an error remains undetected until an error detector identifies the incorrect state.

—Besides causing a failure, undetected or detected errors may cause out-of-norm behavior of system parameters as a side effect. We call this out-of-norm behavior a *symptom*.[3] In the context of *software aging*, symptoms are similar to *aging-related errors*, as implicitly introduced in Grottke and Trivedi [2007] and explicitly named in Grottke et al. [2008].

Figure 3 visualizes how a fault can evolve into a failure. Note that there can be an m-to-n mapping between faults, errors, symptoms, and failures: For example, several faults may result in one single error or one fault may result in several errors. The same holds for errors and failures: some errors result in a failure, some errors do not, and more complicated, some errors only result in a failure under special conditions. As is also indicated in the figure, an undetected error may cause a failure directly or might even be nondistinguishable from it. Furthermore, errors do not necessarily show symptoms.

[3]This should not be confused with Iyer et al. [1986], who used the term *symptom* for the most significant errors within an error group.

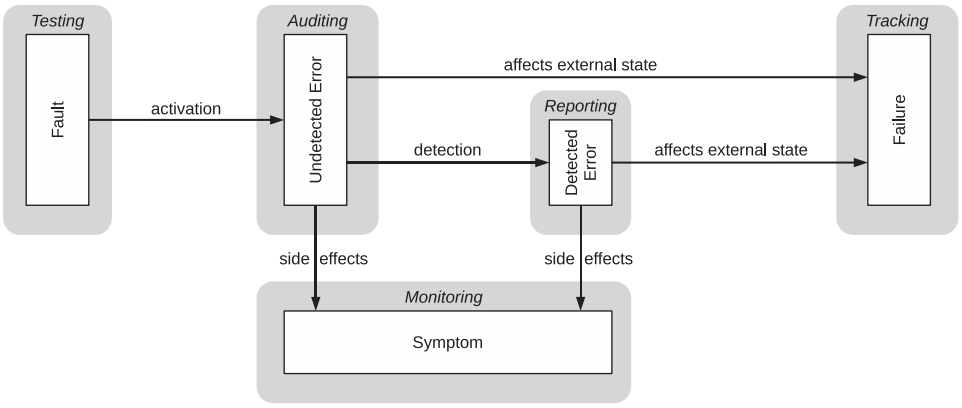

Fig. 3. Interrelations of faults, errors, symptoms, and failures. Encapsulating boxes show the technique by which the corresponding flaw can be made visible.

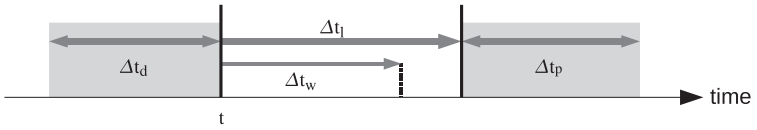

Fig. 4. Time relations in online failure prediction. Present time is denoted by t. Failures are predicted with lead-time Δt_l, which must be greater than minimal warning-time Δt_w. A prediction is assumed to be valid for some time period, named prediction-period, Δt_p. In order to perform the prediction, some data up to a time horizon of Δt_d is used. Δt_d is called *data window size*.

To further clarify the terms *fault*, *error*, *symptom*, and *failure*, consider a fault-tolerant system with a memory leak in its software. The fault is, for example, a missing `free` statement in the source code. However, as long as this part of the software is never executed, the fault remains dormant. Once the piece of code that should free memory is executed, the software enters an incorrect state, that is, it turns into an error (memory is consumed and not freed although it is not needed anymore). If the amount of unnecessarily allocated memory is sufficiently small, this incorrect state will neither be detected nor will it prevent the system from delivering its intended service (no failure is observable from the outside). Nevertheless, if the piece of code with the memory leak is executed many times, the amount of free memory will slowly decrease in the long run. This out-of-norm behavior of the system parameter "free memory" is a symptom of the error. At some point in time, there might not be enough memory for some memory allocation and the error is detected. However, if it is a fault-tolerant system, the failed memory allocation still does not necessarily lead to a service failure. For example, the operation might be completed by some spare unit. Only if the entire system, as observed from the outside, cannot deliver its service correctly, does a failure occur.

2.2. Online Prediction

The task of online prediction is visualized in Figure 4: At present time t, the potential occurrence of a failure is to be predicted some time ahead (lead-time Δt_l) based on the current system state, which is assessed by system monitoring within a data window of length Δt_d. The prediction is valid for some time interval Δt_p, which is called the *prediction period*. Increasing Δt_p increases the probability that a failure is predicted

Table I. Contingency Table
(Any failure prediction belongs to one out of four cases: if the prediction algorithm decides in favor of an upcoming failure, which is called a *positive*, it results in raising a failure warning. This decision can be right or wrong. If in reality a failure is imminent, the prediction is a true positive. If not, a false positive. Analogously, in case the prediction indicates that the system is running well (a negative prediction), this prediction may be right (true negative) or wrong (false negative)).

	True failure	True nonfailure	Sum
Prediction: failure (failure warning)	True positive (*TP*) (correct warning)	False positive (*FP*) (false warning)	Positives (*POS*)
Prediction: no failure (no failure warning)	False negative (*FN*) (missing warning)	True negative (*TN*) (correctly no warning)	Negatives (*NEG*)
Sum	Failures (*F*)	Nonfailures (*NF*)	Total (*N*)

correctly.[4] On the other hand, if Δt_p is too large, the prediction is of little use since it is not clear when exactly the failure will occur. Since failure prediction does not make sense if the lead-time is larger than the time the system needs to react in order to avoid a failure or to prepare for it, Figure 4 introduces the minimal warning time Δt_w. If the lead-time were shorter than the warning time, there would not be enough time to perform any preparatory or preventive actions.

3. EVALUATION METRICS

In order to investigate the quality of failure prediction algorithms and to compare their potential, it is necessary to specify metrics (figures of merit). It is the goal of failure prediction to predict failures accurately: covering as many failures as possible while at the same time generating as few false alarms as possible. A perfect failure prediction would achieve a one-to-one matching between predicted and true failures. This section will introduce several established metrics for the goodness of fit of prediction. Some other metrics have been proposed, for example, the kappa statistic Altman [1991, page 404], but they are rarely used by the community. A more detailed discussion and analysis of evaluation metrics for online failure prediction can be found in Salfner [2008], Chapter 8.2.

Table I defines four cases: a failure prediction is a true positive if a failure occurs within the prediction period and a failure warning is raised. If no failure occurs and a warning is given, the prediction is a false positive. If the algorithm fails to predict a true failure, it is a false negative. If no true failure occurs and no failure warning is raised, the prediction is a true negative.

3.1. Contingency Table Metrics

The metrics presented here are based on the contingency table (see Table I) and therefore called *contingency table metrics*. They are often used in pairs such as precision/recall, true positive rate/false positive rate, sensitivity/specificity, and positive predictive value/negative predictive value. Table II provides an overview.

In various research areas, different names have been established for the same metrics. Hence the leftmost column indicates which terms are used in this article, and the rightmost column lists additional names.

Precision is defined as the ratio of correctly identified failures to the number of all predicted failures:

$$precision = \frac{TP}{TP + FP}. \tag{1}$$

[4]For $\Delta t_p \to \infty$, simply predicting that a failure will occur would always be 100% correct!

Table II. Metrics Obtained from Contingency Table (c.f. Table I)
(Different names for the same metrics have been used in various research areas, as listed in the rightmost column).

Name of the metric	Formula	Other names
Precision	$\frac{TP}{TP+FP} = \frac{TP}{POS}$	Confidence, positive predictive value
Recall, true-positive rate	$\frac{TP}{TP+FN} = \frac{TP}{F}$	Support, sensitivity, statistical power
False-positive rate	$\frac{FP}{FP+TN} = \frac{FP}{NF}$	Fallout
Specificity	$\frac{TN}{TN+FP} = \frac{TN}{NF}$	True-negative rate
False-negative rate	$\frac{FN}{TP+FN} = \frac{FN}{F}$	1-recall
Negative predictive value	$\frac{TN}{TN+FN} = \frac{TN}{NEG}$	
False-positive error rate	$\frac{FP}{FP+TP} = \frac{FP}{POS}$	1-precision
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN} = \frac{TP+TN}{N}$	
Odds ratio	$\frac{TP \cdot TN}{FP \cdot FN}$	

Recall is the ratio of correctly predicted failures to the number of true failures:

$$recall = \frac{TP}{TP + FN}. \tag{2}$$

Consider the following example for clarification: a prediction algorithm that achieves precision of 0.8, generates correct failure warnings (referring to true failures) with a probability of 0.8 and false positives with a probability of 0.2. A recall of 0.9 expresses that 90% of all true failures are predicted and 10% are missed.

In Weiss [1999], variants of precision and recall have been introduced that account for multiple predictions of the same failure and of bursts of false positive predictions.

Improving precision, that is, reducing the number of false positives, often results in worse recall, that is, increasing the number of false negatives, at the same time. To integrate the tradeoff between precision and recall, the F-measure was introduced by van Rijsbergen [1979], Chapter 7, as the harmonic mean of precision and recall. Assuming equal weighting of precision and recall, the resulting formula is

$$F\text{-}measure = \frac{2 \cdot precision \cdot recall}{precision + recall} \in [0, 1]. \tag{3}$$

The higher the quality of the predictor, the higher the F-measure. If precision and recall both approach zero, the limit of the F-measure is also zero.

One problem with precision and recall is that they do not account for true negative predictions. Hence the following metrics should be used in combination with precision and recall. The false positive rate is defined as the ratio of incorrectly predicted failures to the number of all nonfailures. The smaller the false positive rate, the better, provided that the other metrics are not changed for the worse.

$$false\ positive\ rate = \frac{FP}{FP + TN}. \tag{4}$$

Specificity is defined as the ratio of all correctly not raised failure warnings to the number of all nonfailures:

$$specificity = \frac{TN}{FP + TN} = 1 - false\ positive\ rate. \tag{5}$$

The negative predictive value (NPV) is the ratio of all correctly not raised failure warnings to the number of all not raised warnings:

$$negative\ predictive\ value = \frac{TN}{TN + FN}. \tag{6}$$

Accuracy is defined as the ratio of all correct predictions to the number of all predictions that have been performed:

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN}. \tag{7}$$

Due to the fact that failures usually are rare events, accuracy does not appear to be an appropriate metric for failure prediction: a strategy that always classifies the system to be nonfaulty can achieve excellent accuracy since it is right in most of the cases, although it does not catch any failure (recall is zero).

From this discussion it might be concluded that true negatives are not of interest for the assessment of failure prediction techniques. This is not necessarily true since the number of true negatives can help to assess the impact of a failure prediction approach on the system. Consider the following example: for a given time period including a given number of failures, two prediction methods do equally well in terms of TP, FP, and FN; hence both achieve the same precision and recall. However, one prediction algorithm performs 10 times as many predictions as the second since, for example, one operates on measurements taken every second and the other on measurements that are taken only every 10s. The difference between the two methods is reflected only in the number of TNs and will hence only become visible in metrics that include TNs. The number of true negatives can be determined by counting all predictions that were performed when no true failure was imminent and no failure warning was issued as a result of the prediction.

It should also be pointed out that quality of predictions depends not only on algorithms but also on the data window size Δt_d, lead-time Δt_l, and prediction-period Δt_p. For example, since it is very unlikely to predict that a failure will occur at one exact point in time but only within a certain time interval (prediction period), the number of true positives depends on Δt_p: the longer the prediction period, the more failures are captured and hence the number of true positives goes up, which affects, for example, recall. That is why the contingency table should only be determined for one specific combination of Δt_d, Δt_p, and Δt_l.

3.2. Precision/Recall Curve

Many failure predictors involve an adjustable decision threshold, upon which a failure warning is raised or not. If the threshold is low, a failure warning is raised very easily, which increases the chance to catch a true failure (resulting in high recall). However, a low threshold also results in many false alarms, which leads to low precision. If the threshold is very high, the situation is the other way around: precision is good while recall is low. Precision/recall curves are used to visualize this tradeoff by plotting

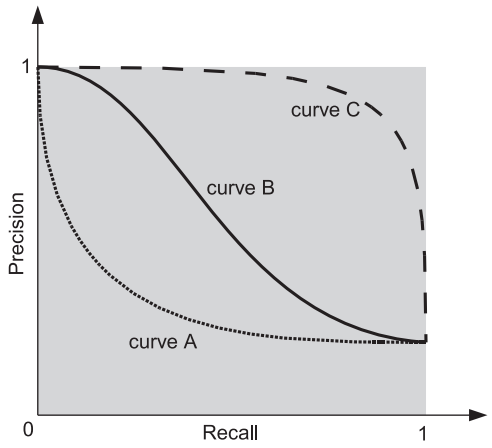

Fig. 5. Sample precision/recall curves visualizing the tradeoff between precision and recall. Curve A shows a predictor that is performing quite poorly: there is no point, where precision and recall simultaneously have high values. The failure prediction method pictured by curve B performs slightly better. Curve C reflects an algorithm whose predictions are mostly correct.

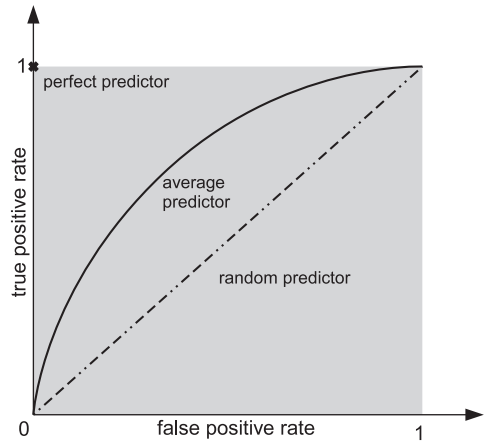

Fig. 6. Sample ROC plots. A perfect failure predictor shows a true positive rate of 1 and a false positive rate of zero. Many existing predictors facilitate to adjust the tradeoff between true positive rate and false positive rate, as indicated by the solid line. The diagonal shows a random predictor: at each point the chance of a false or true positive prediction is equal.

precision over recall for various threshold levels. The plots are sometimes also called *positive predictive value / sensitivity plots*. An example is shown in Figure 5.

3.3. Receiver Operating Characteristic (ROC) and Area Under the Curve (AUC)

Similar to precision/recall curves, the receiver operating characteristic (ROC) curve (see Figure 6) plots the true positive rate versus false positive rate (sensitivity/recall versus "1-specificity," respectively) and therefore enables one to assess the ability of a model to discriminate between failures and nonfailures. The closer the curve gets to the upper left corner of the ROC space, the more accurate is the model.

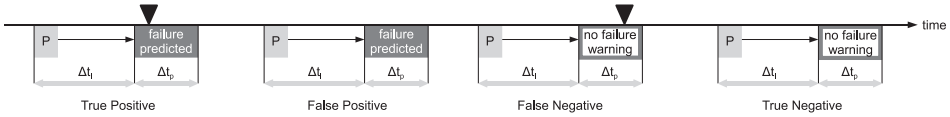

Fig. 7. A time line showing true failures of a test data set (▾) and all four types of predictions: TP, FP, FN, TN. A failure is counted as predicted if it occurs within a prediction period of length Δt_p, which starts lead-time Δt_l after the beginning of prediction P.

As ROC curves accomplish for all thresholds, accuracy of prediction techniques can easily be evaluated by comparing their ROC curves: Area Under the Curve (AUC) is defined as the area between a ROC curve and the x-axis. It is calculated as

$$AUC = \int_0^1 tpr(fpr)\,\mathrm{d}\,fpr \quad \in [0, 1], \tag{8}$$

where *tpr* and *fpr* denote true positive rate and false positive rate, respectively. AUC is basically the probability that a data point of a failure-prone situation receives a higher score than a data point of a non-failure-prone situation. As AUC turns the ROC curve into a single number by measuring the area under the ROC curve, it summarizes the inherent capacity of a prediction algorithm to discriminate between failures and nonfailures. A random predictor receives an AUC of 0.5 (the inversion is not always true, see, e.g., Flach [2004]), while a perfect predictor results in an AUC of one.

3.4. Estimating the Metrics

In order to estimate the metrics discussed in this section, a reference data set is needed, for which it is known when failures have occurred. In machine learning, this is called a *labeled data set*. Since the evaluation metrics are determined using statistical estimators, the data set should be as large as possible. However, failures are in general rare events, usually putting a natural limit to the number of failures in the data set.

If the online prediction method involves estimation of parameters from the data, the data set has to be divided into up to three parts:

(1) *Training* data set. The data on which parameter optimization is performed.
(2) *Validation* data set. In case the parameter optimization algorithm might result in local rather than global optima, or in order to control the so-called bias-variance tradeoff, validation data is used to select the best parameter setting.
(3) *Test* data set. Evaluation of failure prediction performance is carried out on data that has not been used to determine the parameters of the prediction method. Such evaluation is also called *out-of-sample* evaluation.

In order to determine the number of TP, FP, FN, and TN predictions required to fill out the contingency table and to subsequently compute metrics such as precision and recall, the prediction algorithm is applied to test data and prediction outcomes are compared to the true occurrence of failures. The four cases that can occur are depicted in Figure 7. As can be seen from the figure, prediction period Δt_p (c.f. Section 2.2) is used to determine whether a failure is counted as predicted or not. Hence, the choice of Δt_p impacts the contingency table and should be chosen in congruence with requirements for subsequent steps in proactive fault management.

In order to determine curves such as precision/recall curves or ROC plots, the predictors rating rather than the threshold-based binary decision should be stored, which

enables one to generate the curve for all possible threshold values using an algorithm such as that described in Fawcett [2004].

Estimating evaluation metrics from a finite set of test data only yields an approximate assessment of the prediction performance and should hence be accompanied by confidence intervals. Confidence intervals are usually estimated by running the estimation procedure several times. Since this requires an enormous amount of data, techniques such as cross validation, jackknife, or bootstrapping are applied. A more detailed discussion of such techniques can be found in Salfner [2008], Chapter 8.4.

4. A TAXONOMY OF ONLINE FAILURE PREDICTION METHODS

A significant body of work has been published in the area of online failure prediction research. This section introduces a taxonomy of online failure prediction approaches in order to structure the manifold of approaches. In order to predict upcoming failures from measurements, the causative factors, which are faults, have to be made visible. As explained in Section 2, a fault can evolve into a failure through four stages: *fault, undetected error, detected error*, and *failure*, and it might cause side effects which are called *symptoms*. Therefore, measurement-based failure prediction has to rely on capturing faults (see Figure 3):

(1) In order to identify a fault, *testing* must be performed. The goal of testing is to identify flaws in a system regardless whether the entity under test is actually used by the system or not. For example, in memory testing, the entire memory is examined even though some areas might never be used.
(2) Undetected errors can be identified by *auditing*. Auditing describes techniques that check whether the entity under audit is in an incorrect state. For example, memory auditing would inspect used data structures by checksumming.
(3) Symptoms, which are side effects of errors, can be identified by *monitoring* system parameters such as memory usage, workload, sequence of function calls, etc. An undetected error can be made visible by identifying out-of-norm behavior of the monitored system variable(s).
(4) Once an error detector identifies an incorrect state, the detected error may become visible by *reporting*. Reports are written to some logging mechanism such as logfiles or Simple Network Management Protocol (SNMP) messages.
(5) Finally, the occurrence of failures can be made visible by *tracking* mechanisms. Tracking includes, for example, watching service response times or sending testing requests to the system for the purpose of monitoring.

The taxonomy introduced here is structured along the five stages of fault capturing. However, the focus of this article is *online* failure prediction, which means that short-term predictions are made on the basis of *runtime monitoring*. Hence, methods based on *testing* are not included since testing is not performed during runtime. *Auditing* of undetected errors can be applied offline as well as during runtime, which qualifies it for being included in the taxonomy. However, we have not found any publication investigating audit-based *online* failure prediction, and hence the branch has no further subdivisions. The full taxonomy is shown in Figure 8.

In Figure 8, the tree is split vertically into four major branches of the *type of input data* used, namely, data from failure tracking, symptom monitoring, detected error reporting, and undetected error auditing. Each major branch is further divided vertically into *principal approaches*. Each principal approach is then horizontally divided into categories grouping the methods that we have surveyed. In this section we briefly

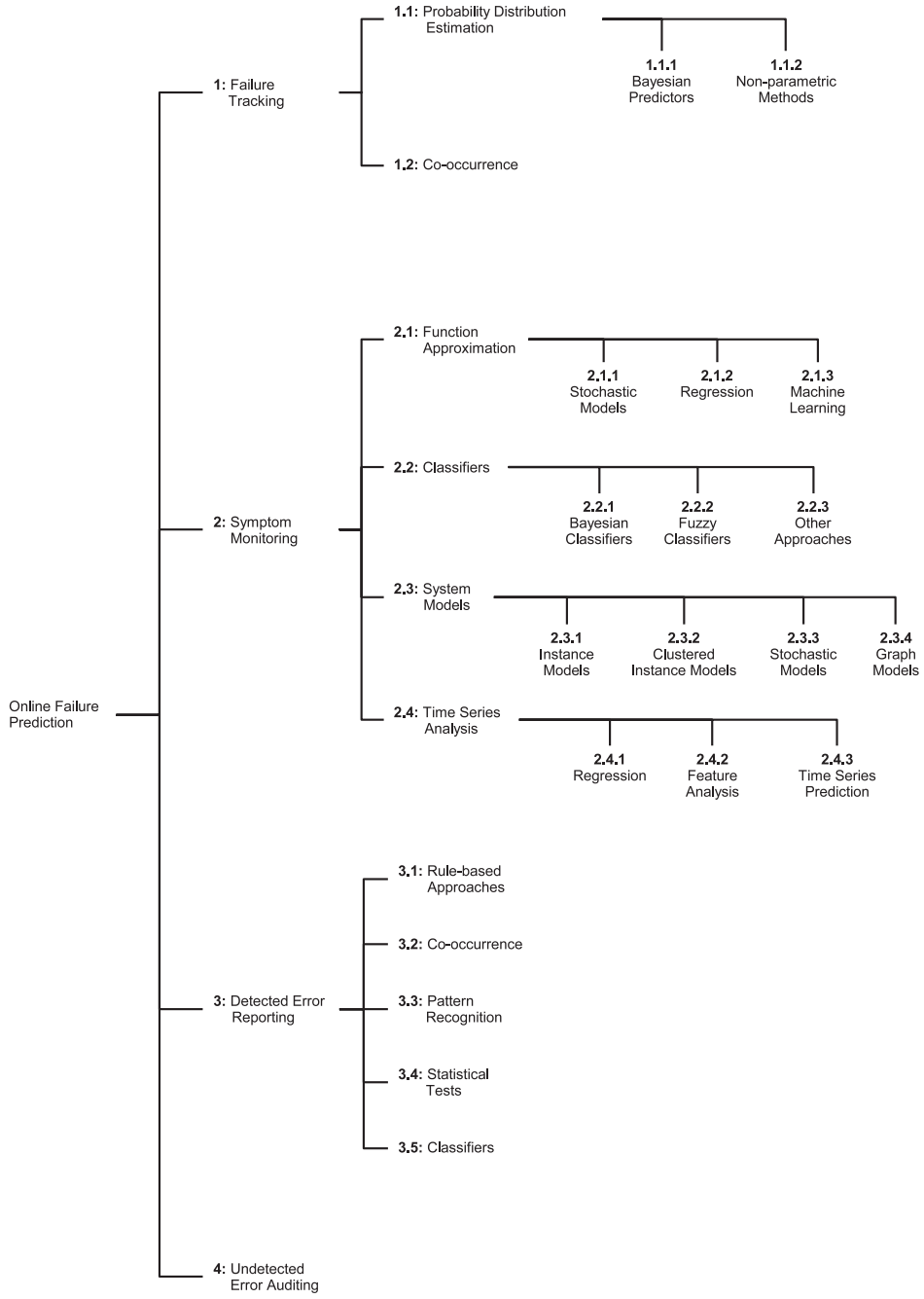

Fig. 8. A taxonomy for online failure prediction approaches.

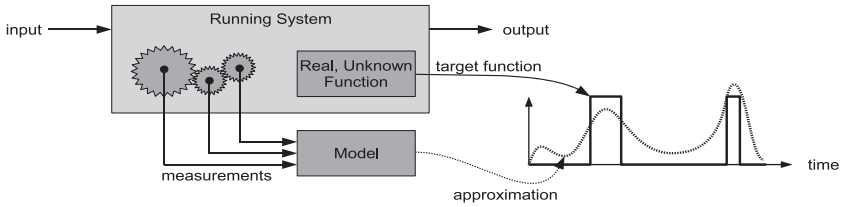

Fig. 9. Function approximation tries to mimic an unknown target function by the use of measurements taken from a system at runtime.

describe the major categories (vertical splits), whereas details on the methods that are actually used (horizontal splits) are provided in Section 5.

4.1. Failure Tracking (1)

The basic idea of failure prediction based on failure tracking is to draw conclusions about upcoming failures from the occurrence of previous failures. This may include the time of occurrence as well as the types of failures that have occurred.

4.1.1. Probability Distribution Estimation (1.1). Prediction methods belonging to this category try to estimate the probability distribution of the time to the next failure from the previous occurrence of failures. Such approaches are in most cases rather formal since they have their roots in (offline) reliability prediction, even though they are applied during runtime.

4.1.2. Cooccurrence (1.2). The fact that system failures can occur close together either in time or in space (e.g., at proximate nodes in a cluster environment) can be exploited to make an inference about failures that might come up in the near future.

4.2. Symptom Monitoring (2)

The motivation for analyzing periodically measured system variables such as the amount of free memory in order to identify an imminent failure is the fact that some types of errors affect the system even before they are detected (this is sometimes referred to as *service degradation*). A prominent example for this are memory leaks: due to the leak the amount of free memory is slowly decreasing over time, but, as long as there is still memory available, the error is neither detected nor is a failure observed. When memory is getting scarce, the computer may first slow down (e.g., due to memory swapping) and only if there is no memory left is an error detected and a failure might result. The key notion of failure prediction based on monitoring data is that errors like memory leaks can be grasped by their side effects on the system such as exceptional memory usage, CPU load, disk I/O, or unusual function calls in the system. These side effects are called *symptoms*. Symptom-based online failure prediction methods frequently address nonfailstop failures, which are usually more difficult to grasp. Four principle approaches have been identified: failure prediction based on function approximation, classifiers, a system model, and time series analysis.

4.2.1. Function Approximation (2.1). Function approximation techniques try to mimic a target value, which is supposed to be the output of an unknown function of measured system variables as input data (see Figure 9). For failure prediction the target function is usually one of the following:

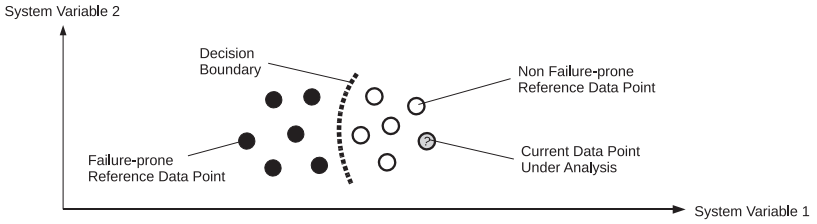

Fig. 10. Online failure prediction by classification of system variable observations. A decision boundary is determined from labeled reference data points (training data). During runtime, the current situation is judged to be failure-prone or not depending on which side of the decision boundary the current data point under analysis is.

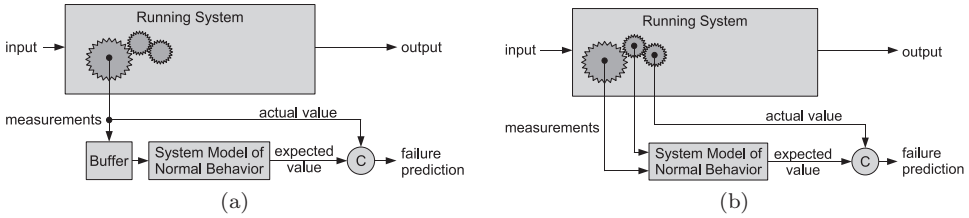

Fig. 11. Online failure prediction using a system model. Failure prediction is performed by comparison (C) of an expected value to an actual, measured value. The expected value is computed from a system model of normal behavior. The expected value is either computed from previous (buffered) monitoring values (a) or from other monitoring variables measured at the same time (b).

(1) The probability of failure occurrence. In this case, the target value is a Boolean variable only available in the training data set but not during runtime. This case is depicted in Figure 9.
(2) Some computing resource such as the amount of free memory. Although the current value is measurable during runtime, function approximation is used in order to extrapolate resource usage into the future and to predict the time of resource exhaustion.

4.2.2. Classifiers (2.2). Instead of approximating a target function, some failure prediction algorithms evaluate the current values of system variables directly. Failure prediction is achieved by classifying whether the current situation is failure-prone or not. The classifier's decision boundary is usually derived from a reference data set for which it is known for each data point whether it indicates a failure-prone or non-failure-prone situation. Online failure prediction during runtime is then accomplished by checking on which side of the decision boundary the current monitoring values are (see Figure 10). The dimensions of data points can be discrete or continuous values. For example, in hard-disk failure prediction based on self-monitoring and reporting technology (SMART) values, input data may consist of the number of reallocated sectors (discrete value) and the drive's temperature (theoretically continuous variable).

4.2.3. System Models (2.3). In contrast to the classifier approach, which requires training data for both the failure-prone and non-failure-prone case, system model-based failure prediction approaches rely on modeling of failure-free behavior only, that is, normal system behavior. The model is used to compute expected values, to which the current measured values are compared. If they differ significantly, the system is suspected not to behave as normal and an upcoming failure is predicted (see Figure 11).

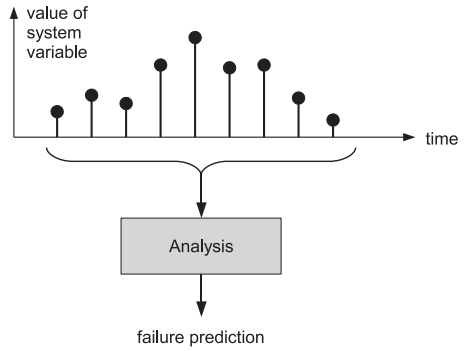

Fig. 12. Online failure prediction by time series analysis. Several successive measurements of a system variable are analyzed in order to predict upcoming failures.

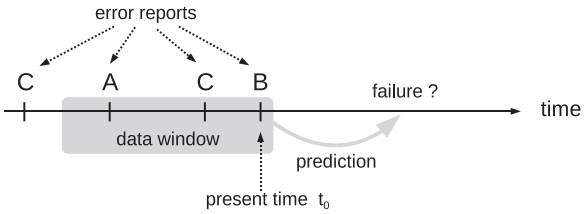

Fig. 13. Failure prediction based on the occurrence of error reports (A, B, C). The goal is to assess the risk of failure at some point in the future. In order to perform the prediction, some data that have occurred shortly before present time t_0 are taken into account (data window).

4.2.4. Time Series Analysis (2.4). As the name suggests, failure prediction approaches in this category treat a sequence of monitored system variables as a time series. This means that the prediction is based on an analysis of several successive samples of a system variable, as shown in Figure 12. The analysis of the time series either involves computation of a residual value on which the current situation is judged to be failure-prone or not, or the future progression of the time series is predicted in order to estimate, for example, time until resource exhaustion.

4.3. Detected Error Reporting (3)

When an error is detected, the detection event is usually reported using some logging facility. Hence, failure prediction approaches that use error reports as input data have to deal with event-driven input data. This is one of the major differences to symptom monitoring-based approaches, which in most cases operate on periodic system observations. Furthermore, symptoms are in most cases real-valued while error events mostly are discrete, categorical data such as event IDs, component IDs, etc. The task of online failure prediction based on error reports is shown in Figure 13: at present time t_0, error reports that have occurred during some data window before t_0 are analyzed in order to decide whether there will be a failure at some point in time in the future.

4.3.1. Rule-Based Systems (3.1). The essence of rule-based failure prediction is that the occurrence of a failure is predicted once at least one of a set of conditions is met.

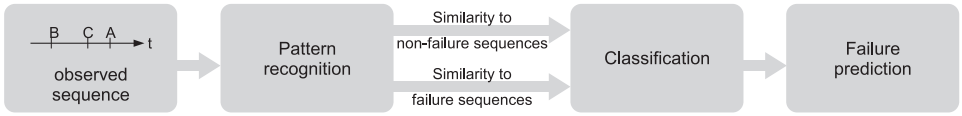

Fig. 14. Failure prediction by recognition of patterns in sequences of error reports.

Hence rule-based failure prediction has the form

$$\texttt{IF } \langle condition_1 \rangle \texttt{ THEN } \langle failure\ warning \rangle$$
$$\texttt{IF } \langle condition_2 \rangle \texttt{ THEN } \langle failure\ warning \rangle$$
$$\ldots$$

Since in most computer systems the set of conditions cannot be set up manually, the goal of failure prediction algorithms in this category is to identify the conditions algorithmically from a set of training data. The art is to find a set of rules that is general enough to capture as many failures as possible but that is also specific enough not to generate too many false failure warnings.

4.3.2. Cooccurrence (3.2). Methods that belong to this category analyze error detections that occur close together either in time or in space. The difference to Category 1.2 is that the analysis is based on detected errors rather than previous failures.

4.3.3. Pattern Recognition (3.3). Sequences of error reports form error patterns. The goal of pattern recognition-oriented failure prediction approaches is to identify patterns that indicate an upcoming failure. In order to achieve this, usually a ranking value is assigned to an observed sequence of error reports expressing similarity to patterns that are known to lead to system failures and to patterns that are known not to lead to a system failure. The final prediction is then accomplished by classification on basis of similarity rankings (see Figure 14).

4.3.4. Statistical Tests (3.4). The occurrence of error reports can be analyzed using statistical tests. For example, the histogram of number of error reports per component can be analyzed and compared to the "historically normal" distribution using a statistical test.

4.3.5. Classifiers (3.5). The goal of classification is to assign a class label to a given input data vector, which in this case is a vector of error detection reports. Since one single detected error is generally not sufficient to infer whether a failure is imminent or not, the input data vector is usually constructed from several errors reported within a time window.

4.4. Undetected Error Auditing (4)

In order to predict failures as early as possible, one can actively search for incorrect states (undetected errors) within a system. For example, the inode structure of a UNIX file system could be checked for consistency. A failure might then be predicted depending on the files that are affected by a file system inconsistency. The difference to detected error reporting (Category 3) is that auditing actively searches for incorrect states regardless whether the data is used at the moment or not, while error detection performs checks on data that is actually used or produced. However, as stated above, we have

not found any failure prediction approaches that apply online auditing and hence the taxonomy contains no further subbranches.

5. SURVEY OF PREDICTION METHODS

In this survey, failure prediction methods are briefly described with appropriate reference to the source, and summarized in Table III in Section 6. Representative selected methods are explained in greater detail in online Appendices A–K available in the ACM Digital Library.

5.1. Failure Tracking (1)

Two principal approaches to online failure prediction based on the previous occurrence of failures can be determined: estimation of the probability distribution of a random variable for time to the next failure, and approaches that build on the co-occurrence of failure events.

5.1.1. Probability Distribution Estimation (1.1). In order to estimate the probability distribution of the time to the next failure, Bayesian predictors as well as non-parametric methods have been applied.

5.1.1.1. Bayesian Predictors (1.1.1). The key notion of Bayesian failure prediction is to estimate the probability distribution of the next time to failure by benefiting from the knowledge obtained from previous failure occurrences in a Bayesian framework.

In Csenki [1990], such a Bayesian predictive approach [Aitchison and Dunsmore 1975] was applied to the Jelinski-Moranda software reliability model [Jelinski and Moranda 1972] in order to yield an improved estimate of the next time to failure probability distribution. Although developed for (offline) software reliability prediction, the approach could be applied in an online manner as well.

5.1.1.2. Nonparametric Methods (1.1.2). It has been observed that the failure process can be nonstationary and hence the probability distribution of time-between-failures (TBF) varies. The reasons for nonstationarity are manifold, since the fixing of bugs, changes in configuration, or even varying utilization patterns can affect the failure process. In these cases, techniques such as histograms result in poor estimations since stationarity (at least within a time window) is inherently assumed. For these reasons, the nonparametric method of Pfefferman and Cernuschi-Frias [2002] assumes the failure process to be a Bernoulli experiment where a failure of type k occurs at time n with probability $p_k(n)$. From this assumption, it follows that the probability distribution of TBF for failure type k is geometric since only the nth outcome is a failure of type k, and hence the probability is

$$Pr\{TBF_k(n) = m \mid \text{failure of type } k \text{ at } n\} = p_k(n)\big(1 - p_k(n)\big)^{m-1}. \tag{9}$$

The authors proposed a method to estimate $p_k(n)$ using an autoregressive averaging filter with a "window size" depending on the probability of the failure type k.

5.1.2. Cooccurrence (1.2). Due to sharing of resources, system failures can occur close together either in time or in space (at a closely coupled set of components or computers) (see, e.g., Tang and Iyer [1993]). However, in most cases, cooccurrence has been analyzed for root cause analysis rather than failure prediction.

It has been observed several times that failures occur in clusters in a temporal as well as in a spatial sense. Liang et al. [2006] chose such an approach to predict failures of

IBM's BlueGene/L from event logs containing reliability, availability, and serviceability data. The key to their approach is data preprocessing employing first a categorization and then temporal and spatial compression: temporal compression combines all events at a single location occurring with interevent times lower than some threshold, and spatial compression combines all messages that refer to the same location within some time window. Prediction methods are rather straightforward: using data from temporal compression, if a failure of type application I/O or network appears, it is very likely that a next failure will follow shortly. If spatial compression suggests that some components have reported more events than others, it is very likely that additional failures will occur at that location. Please refer to online Appendix B for further details.

Fu and Xu [2007] further elaborated on temporal and spatial compression and introduced a measure of temporal and spatial correlation of failure events in distributed systems.

5.2. Symptom Monitoring (2)

Symptoms are side effects of errors. In this section, online failure prediction methods are surveyed that analyze monitoring data in order to detect symptoms that indicate an upcoming failure.

5.2.1. Function Approximation (2.1). *Function approximation* is a term used in a large variety of scientific areas. Applied to the task of online failure prediction, there is an assumed unknown functional relationship between monitored system variables (input to the function) and a target value (output of the function). The objective is to reveal this relationship from measurement data.

5.2.1.1. Stochastic Models (2.1.1). Vaidyanathan and Trivedi [1999] tryed to approximate the amount of swap space used and the amount of real free memory (target functions) from workload-related input data such as the number of system calls. They constructed a semi-Markov reward model in order to obtain a workload-based estimation of resource consumption rate, which was then used to predict the time to resource exhaustion. In order to determine the states of the semi-Markov reward model, the input data was clustered. The authors assumed that these clusters represented 11 different workload states. State transition probabilities were estimated from the measurement dataset and sojourn-time distributions were obtained by fitting two-stage-hyperexponential or two-stage-hypoexponential distributions to the training data. Then, a resource consumption "reward" rate for each workload state was estimated from the data: Depending on the workload state the system was in, the state reward defined at what rate the modeled resource was changing. The rate was estimated by fitting a linear function to the data using the method of Sen [1968]. Experiments have been performed on data recorded from a SunOS 4.1.3 workstation. Please refer to online Appendix C for more details on the approach.

Li et al. [2002] collected various parameters such as used swap space from an Apache Web server and built autoregressive model with auxiliary input (ARX) to predict the further progression of system resources utilization. Failures were predicted by estimating resource exhaustion times. They compared their method to that of Castelli et al. [2001] (see Category 2.4.1) and showed that, on their data set, ARX modeling resulted in much more accurate predictions.

5.2.1.2. Regression (2.1.2). In curve fitting, which is another name for regression, parameters of a function are adapted such that the curve best fits the measurement data, for example, by minimizing mean square error. The simplest form of regression is curve fitting of a linear function.

Andrzejak and Silva [2007] applied deterministic function approximation techniques such as splines to characterize the functional relationships between the target function (the authors used the term *aging indicator*) and "work metrics" as input data. Work metrics are, for example, the work that has been accomplished since the last restart of the system. Deterministic modeling offers a simple and concise description of system behavior with few parameters. Additionally, using work-based input variables rather than time-based variables offers the advantage that the function does not depend on absolute time anymore: for example, if there is only little load on a server, aging factors accumulate slowly and so does accomplished work whereas, in the case of high load, both accumulate more quickly. The authors presented experiments where performance of an Apache Axis SOAP (Simple Object Access Protocol) server has been modeled as a function of various input data such as requests per second or the percentage of CPU idle time.

5.2.1.3. Machine Learning (2.1.3). Function approximation is one of the predominant applications of machine learning. It seems natural that various techniques have a long tradition in failure prediction, as can also be seen from various patents in that area. Troudet et al. [1990] proposed using neural networks for failure prediction of mechanical parts and Wong et al. [1996] used neural networks to approximate the impedance of passive components of power systems. The authors used an RLC-Π model, which is a standard electronic circuit consisting of a two resistors (R), an inductor (L), and two capacities (C), where faults have been simulated to generate the training data. Neville [1998] described how standard neural networks can be used for failure prediction in large-scale engineering plants.

Turning to publications regarding failure prediction in large-scale computer systems, various techniques have been applied there, too.

Hoffmann [2006] developed a failure prediction approach based on universal basis functions (UBF), which are an extension to radial basis functions (RBF) that use a weighted convex combination of two kernel functions instead of a single kernel. UBF approximation has been applied to predict failures of a telecommunication system. Hoffmann et al. [2007] conducted a comparative study of several modeling techniques with the goal to predict resource consumption of the Apache Web server. The study showed that UBF turned out to yield the best results for free physical memory prediction, while server response times could be predicted best by support vector machines (SVM). Online Appendix D provides further details on UBF-based failure prediction.

One of the major findings in Hoffmann et al. [2007] was that the issue of choosing a good subset of input variables has a much greater influence on prediction accuracy than the choice of modeling technology. This means that the result might be better if, for example, only workload and free physical memory are taken into account and other measurements such as used swap space are ignored. *Variable selection* (some authors also use the term *feature selection*) is concerned with finding the optimal subset of measurements. Typical examples of variable selection algorithms are principle component analysis (PCA; see Hotelling [1933]) as used in Ning et al. [2006], or Forward Stepwise Selection (see, e.g., Hastie et al. [2001], Chapter 3.4.1), which was used in Turnbull and Alldrin [2003]. In addition to UBF, Hoffmann [2006] has also developed a new algorithm called the *Probabilistic Wrapper Approach* (PWA), which combines probabilistic techniques with forward selection or backward elimination.

Instance-based learning methods store the entire training dataset including input and target values and predict by finding similar matches in the stored database of training data (and eventually combining them). Kapadia et al. [1999] have applied three learning algorithms (k-nearest-neighbors, weighted average, and weighted polynomial regression) to predict the CPU time of the semiconductor manufacturing simulation

software T-Suprem3. The prediction was based on input parameters to the software such as minimum implant energy or number of etch steps in the simulated semiconductor manufacturing process.

Fu and Xu [2007] built a neural network to approximate the number of failures in a given time interval. The set of input variables consisted of a temporal and spatial failure correlation factor together with variables, such as CPU utilization or the number of packets transmitted by a computing node. The authors used (not further specified) neural networks. Data of 1 year of operation of the Wayne State University Grid was analyzed as a case study. Due to the fact that a correlation value of previous failures was used as input data as well, this prediction approach also partly fits into Category 1.2.

In Abraham and Grosan [2005], the target function is the so-called stressor-susceptibility-interaction (SSI), which basically denotes failure probability as a function of external stressors such as environment temperature or power supply voltage. The overall failure probability can be computed by the integration of single SSIs. The paper presents an approach where genetic programming has been used to generate code representing the overall SSI function from training data of an electronic device's power circuit.

5.2.2. Classifiers (2.2). Failure prediction methods in this category build on classifiers that are trained from failure-prone as well as non-failure-prone data samples.

5.2.2.1. Bayesian Classifiers (2.2.1). In Hamerly and Elkan [2001], two Bayesian failure prediction approaches were described. The first Bayesian classifier proposed by the authors is abbreviated by NBEM, expressing that a specific naïve Bayes model is trained with the Expectation Maximization algorithm based on a real data set of SMART values of Quantum Inc. disk drives. Specifically, a mixture model was proposed where each naïve Bayes submodel m is weighted by a model prior $P(m)$ and an expectation maximization algorithm is used to iteratively adjust model priors as well as submodel probabilities. Second, a standard naïve Bayes classifier is trained from the same input data set. More precisely, SMART variables x_i such as read soft error rate or number of calibration retries are divided into bins. The term *naïve* derives from the fact that all attributes x_i in the current observation vector $\vec{x}$ are assumed to be independent and hence the joint probability $P(\vec{x} \mid c)$ can simply be computed as the product of single attribute probabilities $P(x_i \mid c)$. The authors reported that both models outperformed the rank sum hypothesis test failure prediction algorithm of Hughes et al. [2002][5] (see Category 2.3.1). Please refer to online Appendix E for more details on these methods. In a later study, Murray et al. [2003], the same research group applied two additional failure prediction methods: support vector machines (SVM) and an unsupervised clustering algorithm. The SVM approach is assigned to Category 2.2.2 and the clustering approach belongs to Category 2.3.2.

Pizza et al. [1998] proposed a method to distinguish (i.e., classify) between transient and permanent faults: whenever erroneous component behavior is observed (e.g., by component testing), the objective is to find out whether this erroneous behavior was caused by a transient or permanent fault. Although not mentioned in the paper, this method could be used for failure prediction. For example, a performance failure of a grid computing application might be predicted if the number of permanent grid node failures exceeded a threshold (under the assumption that transient outages do not affect overall grid performance severely). This method enables one to decide whether a tested grid node has a permanent failure or not.

[5]Although Hughes et al. [2002] appeared after Hamerly and Elkan [2001], it was announced and submitted already in 2000.

5.2.2.2. Fuzzy Classifier (2.2.2). Bayes classification requires that input variables take on discrete values. Therefore, monitoring values are frequently assigned to finite number of bins (as, e.g., in Hamerly and Elkan [2001]). However, this can lead to bad assignments if monitoring values are close to a bin's border. Fuzzy classification addresses this problem by using probabilistic class membership.

Turnbull and Alldrin [2003] used radial basis functions networks (RBFNs) to classify monitoring values of hardware sensors such as temperatures and voltages on motherboards. More specifically, all N monitoring values occurring within a data window were represented as a feature vector which was then classified as belonging to a failure-prone or non-failure-prone sequence using RBFNs. Experiments were conducted on a server with 18 hot-swappable system boards with four processors, each. The authors achieved good results, but failures and nonfailures were equally likely in the data set.

Berenji et al. [2003] used an RBF rule base to classify whether a component is faulty or not: using Gaussian rules, a so-called diagnostic model computes a diagnostic signal based on input and output values of components ranging from zero (fault-free) to 1 (faulty). The rule base is algorithmically derived by means of clustering of training data, which consists of input/output value pairs both for the faulty and the fault-free case. The training data is generated from so-called component simulation models that try to mimic the input/output behavior of system components (fault-free and faulty). The same approach is then applied on the next hierarchical level to obtain a system-wide diagnostic models. The approach has been applied to model a hybrid combustion facility developed at the NASA Ames Research Center. The diagnostic signal can be used to predict slowly evolving failures.

Murray et al. [2003] applied SVMs in order to predict failures of hard-disk drives. SVMs were developed by Vapnik [1995] and are powerful and efficient neural network classifiers. In the case of hard-disk failure prediction, five successive samples of each selected SMART attribute set up the input data vector. The training procedure of SVMs adapts the classification boundary such that the margin between the failure-prone and non-failure-prone data points becomes maximal. Although the naïve Bayes approach developed by the same group (see Hughes et al. [2002], Category 2.3.1) is mentioned in the paper, no comparison has been carried out.

In Bodík et al. [2005] hit frequencies of Web pages were analyzed in order to quickly identify nonfailstop failures in the operation of a big commercial Web site. The authors used a naïve Bayes classifier. Following the same pattern as described in Category 2.2.1, the probability $P(k \mid \vec{x})$, where k denotes the class label (normal or abnormal behavior) and $\vec{x}$ denotes the vector of hit frequencies, was computed from likelihoods $P(x_i \mid k)$ which are approximated by Gaussian distributions. Since the training data set was not labeled (it was not known when failures had occurred), likelihoods for the failure case were assumed to be uniformly distributed and unsupervised learning techniques had to be applied. The output of the naïve Bayes classifier was an anomaly score. In the article, a second prediction technique based on a χ^2 test was proposed which is described in Category 2.3.1.

Another valuable contribution of this work was a successful combination of anomaly detection and detailed analysis support in the form of a visual tool.

5.2.2.3. Other Approaches (2.2.3). In a joint effort, the University of California Berkeley and Stanford University have developed a computing approach called *recovery-oriented computing*. As main references, see Brown and Patterson [2001] and Patterson et al. [2002] for an introduction and Candea et al. [2003, 2006] for a description of JAGR (JBoss with Application Generic Recovery), which combines several of the techniques to build a dependable system. Although primarily targeted toward a quick detection and

analysis of failures after their occurrence, several techniques could be used for failure prediction as well. Hence, in this survey runtime path-based methods are included, which are Pinpoint (Category 2.3.1), path modeling using probabilistic context-free grammars (Category 2.3.3), component peer models (Category 2.3.4), and decision trees, which belong to this category.

Kiciman and Fox [2005] proposed constructing a decision tree from runtime paths in order to identify faulty components. The term *runtime path* denotes the sequence of components and other features such as IP addresses of server replicas in the cluster, etc., that are involved in handling one request in a component-based software such as a J2EE application server. Runtime paths are obtained using Pinpoint (see Category 2.3.1). Having recorded a sufficiently large number of runtime paths including failed and successful requests, a *decision tree* for classifying requests as failed or successful is constructed using algorithms such as ID3 or C4.5. Although primarily designed for diagnosis, the authors pointed out that the approach could be used for failure prediction of single requests as well.

Daidone et al. [2006] proposed using a hidden Markov model approach to infer whether the true state of a monitored component is healthy or not. Since the outcome of a component test does not always represent its true state, hidden Markov models were used where observation symbols related to outcomes of component probing, and hidden states related to the (also hidden) component's true state. The true state (in a probabilistic sense) was inferred from a sequence of probing results by the so-called forward algorithm of hidden Markov models. Although not intended by the authors, the approach could be used for failure prediction in the following way: assuming that there are some erroneous states in a system that lead to future failures and others that do not, the proposed hidden Markov model approach can be used to determine (classify) whether a failure is imminent or not on the basis of probing.

5.2.3. System Models (2.3). Online failure prediction approaches belonging to this category utilize a model of failure-free normal system behavior. The current measured system behavior is compared to the expected normal behavior and a failure is predicted in case of a significant deviation. We have categorized system model-based prediction approaches according to how normal behavior is stored: as instances, by clustered instances, using stochastic descriptions, or using formalisms known from control theory.

5.2.3.1. Instance Models (2.3.1). The most straightforward data-driven way to memorize how a system behaves normally is to store monitoring values recorded during failure-free operation. If the current monitoring values are not similar to any of the stored values, a failure is predicted.

Elbaum et al. [2003] carried out experiments where function calls, changes in the configuration, module loading, etc., of the email client "pine" had been recorded. The authors proposed three types of failure prediction among which sequence-based checking was most successful: a failure was predicted if two successive events occurring in "pine" during runtime did not belong to any of the event transitions in the stored training data.

Hughes et al. [2002] applied a simple albeit robust statistical test for hard-disk failure prediction. The authors employed a rank sum hypothesis test to identify failure prone hard disks. The basic idea was to collect SMART values from fault-free drives and store them as reference data set. Then, during runtime, SMART values of the monitored drive were tested the following way: the combined data set consisting of the reference data and the values observed at runtime was sorted and the ranks of

the observed measurements were computed.[6] The ranks were summed up and compared to a threshold. If the drive was not fault-free, the distribution of observed values were skewed and the sum of ranks tended to be greater or smaller than for fault-free drives.

Pinpoint [Chen et al. 2002] tracked requests to a J2EE application server on their way through the system in order to identify the software components that are correlated with a failure. Tracking of the requests yields a set of components used to process the request. In case of a failure, the sets of components for several requests are clustered using a Jaccard score-based metric measuring similarity of the sets. A similar approach could be applied for online failure prediction. If several sets of failure-free request paths were stored, the same distance metric could be used to determine whether the current set of components were similar to any of the known sets, and if not a failure would be supposed to be imminent.

In Bodík et al. [2005], which was described in Category 2.2.2, a second detection/prediction technique was applied to the same data: The current hit frequencies of the 40 most frequently used pages were compared to previous, "historically normal" hit frequencies of the same pages using a χ^2 test. If the two distributions were different with a significance level of 99%, the current observation period was declared anomalous. In addition to this binary decision, an anomaly score was assigned to each page in order to support quick diagnosis. The results achieved using the χ^2 test were comparable to those of the naïve Bayes approach described in Category 2.2.2.

5.2.3.2. Clustered Instance Models (2.3.2). If the amount of storage needed for instance system models exceeds a reasonable level or if a more general representation of training instances is required, training instances can be clustered. In this case, only cluster centroids or the boundaries between clusters need to be stored.

In a followup comparative study to Hughes et al. [2002] (see Category 2.3.1), Murray et al. [2003] introduced a clustering-based failure predictor for hard-disk failure prediction. The basic idea was to identify areas of SMART values where a failure is very unlikely using unsupervised clustering. In other words, all areas of SMART values where the disk operates normally were algorithmically identified from failure-free training data. In order to predict an upcoming failure, the current SMART values were assigned to the most likely cluster. If they did not fit any known cluster (more specifically, maximum class membership probability was below a given threshold), the disk was assumed not to behave normally and a failure was assumed to be imminent.

5.2.3.3. Stochastic Models (2.3.3). Especially in the case of complex and dynamic systems, it seems appropriate to store system behavior using stochastic tools such as probability distributions.

Ward et al. [1998] estimated the mean and variance of the number of TCP connections from two Web proxy servers in order to identify Internet service performance failures. Using a statistical test developed by Pettitt [1977], the values were compared to the previous number of connections at the same time of the day (holidays were treated separately). If the current value deviated significantly, a failure was predicted.

In Chen et al. [2004][7] a runtime path analysis technique based on probabilistic context free grammars (PCFG) was proposed. Probabilistic context-free grammars have been developed in the area of statistical natural language processing (see, e.g., Manning and Schütze [1999, page 381]). The notion of the approach is to build a grammar of all possible nonfaulty sequences of components. By assigning a probability to each

[6]Which in fact only involved simple counting.

[7]The same approach was described in more detail in Kiciman and Fox [2005].

grammar rule, the overall probability of a given sequence of components can be computed. From this, an anomaly score is computed and, if a sufficiently large amount of paths (i.e., requests) have a high anomaly score, a failure is assumed. The approach has been applied to call paths collected from a Java 2 Enterprise Edition (J2EE) demo application, an industrial enterprise voice application network, and from eBay servers. Although not intended by the authors, the same approach could be used to predict, for example, service level failures: if a significant amount of requests do not seem to behave normally, the system might not be able to deliver the required level of service shortly in the future. It might also be applicable for request failure prediction: if the probability of the beginning of a path is very low, there is an increased probability that a failure will occur in the further course of the request.

5.2.3.4. Graph Models (2.3.4). In Kiciman and Fox [2005], a second approach was proposed based on component interaction graphs (a so-called peer model). The peer model reflects how often one component interacts with another component: each component is a node and how often one component interacts with another is expressed by weighted links. One specific instance of a component is checked to see whether it is errorfree by using a χ^2 goodness-of-fit test: if the proportion of runtime paths between a component instance and other component classes deviates significantly from the reference model representing fault-free behavior, the instance is suspected to be faulty. By conducting a trend analysis on the anomaly score, a future failure of the component instance might be predicted.

5.2.3.5. Control Theory Models (2.3.5). It is common in control theory to have an abstraction of the controlled system estimating the internal state of the system and its progression over time by some mathematical equations, such as linear equation systems, differential equation systems, Kalman filters, etc. (see, e.g., Lunze [2003], Chapter 4). These methods are widely used for fault diagnosis (see, e.g., Korbicz et al. [2004], Chapters 2 and 3) but have only rarely been used for failure prediction. However, many of the methods inherently include the possibility of predicting future behavior of the system and hence have the ability to predict failures. For example, in his doctoral dissertation Neville [1998] described the prediction of failures in large-scale engineering plants. Another example was Discenzo et al. [1999], who mention that such methods have been used to predict failures of an intelligent motor using the standard IEEE motor model.

With regard to online failure prediction, Singer et al. [1997] proposed the Multivariate State Estimation Technique (MSET) to detect system disturbances by a comparison of the estimated and measured system state. More precisely, a matrix D of selected measurement data of normal operation is collected. In the operational phase, a state estimation is computed as a combination of D and the current (runtime) observations. The difference between the observed and estimated states constitutes a residual that is checked for significant deviation by a sequential probability ratio test (SPRT). Gross et al. [2002] applied the method to detect software aging [Parnas 1994] in an experiment where a memory-leak fault injector consumed system memory at an adjustable rate. MSET and SPRT have been used to detect whether the fault injector was active and, if so, at what rate it was operating. By this, time to memory consumption can be estimated. MSET has also been applied to online transaction processing servers in order to detect software aging [Cassidy et al. 2002].

5.2.3.6. Other Potential Approaches. A more theoretic approach that could in principle be applied to online failure prediction is to abstract system behavior by a queuing model that incorporates additional knowledge about the current state of the system. Failure prediction can be performed by computing the input value dependent expected response

time of the system. Ward and Whitt [2000] showed how to compute estimated response times of an M/G/I processor-sharing queue based on measurable input data such as number of jobs in the system at time of arrival using a numerical approximation of the inverse Laplace transform.

5.2.4. Time Series Analysis (2.4). We have identified four groups of methods that perform time series analysis: regression tries to fit a function to the data, feature analysis computes a residual of the measurement series, time series prediction tries to predict the future progression of the target function from the series' values itself (without using other measurements as input data), and, finally, signal processing techniques can also be used for time series analysis.

5.2.4.1. Regression (2.4.1). Similar to Category 2.1.2, regression techniques adjust parameters of a function such that it best fits some set of training data. However, in this section the function is fit directly into the sequence of monitored values, whereas in Category 2.1.2 some target function needs to be fit.

Garg et al. [1998] presented a three-step approach to compute time to resource exhaustion. First, the time series of monitored values is smoothed using robust locally weighted regression [Cleveland et al. 1979]. In order to detect whether a trend is present or not, a seasonal Kendall test is applied since this statistic test can detect trends even in the presence of cycles. If a trend is detected, the rate of resource consumption is estimated using a nonparametric procedure developed by Sen [1968]. Experiments have been performed on measurements of system variable "real memory free," size of file table, process table size, and used swap space of UNIX machines.

Castelli et al. [2001] mentioned that IBM has implemented a curve-fitting algorithm for the xSeries Software Rejuvenation Agent. Several types of curves are fit to the measurement data and a model-selection criterion is applied in order to choose the best curve. Prediction is again accomplished by extrapolating the curve.

Cheng et al. [2005] presented a two-step approach for failure prediction in a high-availability cluster system. Failure prediction is accomplished in two stages: first, a health index $\in [0, 1]$ is computed using fuzzy logic, and, the case of a detected "sick" state of a node, a linear function is mapped to the monitored values of the resource in order to estimate mean time to resource exhaustion. The authors also referenced a technique called *prediction interval* to compute a lower and upper bound for time to resource exhaustion. The fuzzy logic assignment of the health index is based on "processor time," "privileged time," "pool nonpaged bytes," and "available Mbytes" (presumably of free memory).

5.2.4.2. Feature Analysis (2.4.2). The goal of feature analysis is to compute a residual value from the time series upon which the decision of whether a failure is imminent or not can be based.

Crowell et al. [2002] discovered that memory-related system parameters such as kernel memory or system cache resident bytes show multifractal characteristics in the case of software aging. The authors used the Hölder exponent to identify fractality, which is a residual expressing the amount of fractality in the time series. In a later paper [Shereshevsky et al. 2003], the same authors extended this concept and built a failure prediction system by applying the Shewhart change detection algorithm [Basseville and Nikiforov 1993, page 31] to the residual time series of Hölder exponents. A failure warning was issued after detection of the second change point. Experiments have been performed on two machines running the System Stress for Windows NT 4.0 and Windows 2000 provided by Microsoft. The algorithm was applied to several memory-related variables in order to predict system crashes.

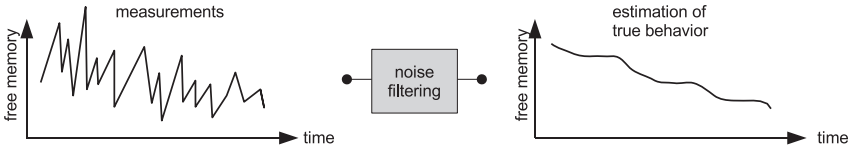

Fig. 15. Failure prediction using signal processing techniques on measurement data.

5.2.4.3. Time Series Prediction (2.4.3). Hellerstein et al. [1999], described an approach to predict if a time series will violate a threshold. In order to achieve this, several time series models are employed to model stationary as well as nonstationary effects. For example, the model accounts for the influence of the day-of-the-week, or time-of-the-day, etc. Experiments have been carried out on prediction of HTTP operations per second of a production Web server. A similar approach has also been described in Vilalta et al. [2002]. In online Appendix F, a more detailed description of this method can be found.

Sahoo et al. [2003] applied various time series models to data of a 350-node cluster system to predict parameters like percentage of system utilization, idle time, and network IO.

A relatively new technique on the rise is based on "rough set theory" [Pawlak et al. 1988], which is an approximation of conventional sets by providing a set of upper and lower values. Meng et al. [2007] combined rough set theory with wavelet networks in order to predict memory usage of a J2EE application server. In the experiments, memory usage was monitored every 12 min and the next monitoring value was predicted based on a series of previous monitoring values. From this one-step-ahead prediction of the monitoring variable, it follows that the lead time Δt_l equals the monitoring period (12 min in this case). The same authors published a similar paper about a combination of fuzzy logic and wavelet networks [Ning et al. 2006], which are called *fuzzy wavelet networks* (FWN) [Ho et al. 2001].

5.2.4.4. Other Potential Approaches. Signal processing techniques are of course related to methods that have already been described (e.g., Kalman filters in Category 2.3.5). Algorithms that fall into this category use signal processing techniques such as low-pass or noise filtering to obtain a clean estimate of a system resource measurement. For example, if free system memory is measured, observations will vary greatly due to the allocation and freeing of memory. Such measurement series can be seen as a noisy signal where noise filtering techniques can be applied in order to obtain the "true" behavior of free system memory: if it is a continuously decreasing function, software aging is likely in progress and the amount of free memory can be estimated for the near future by means of signal processing prediction methods such as low-pass filtering or frequency transformation (see Figure 15).

5.3. Detected Error Reporting (3)

There are two main groups of failure prediction approaches that analyze error reports: rule-based or error pattern-based approaches.

5.3.1. Rule-Based Approaches (3.1). Failure prediction methods in this category derive a set of rules where each rule consists of error reports.

To our knowledge, the first rule-based approach to failure prediction based on reported error events was published by Hätönen et al. [1996]. The authors described a system that identifies episode rules from error logs (the authors used the term *alarm*). Episode rules express temporal ordering of events in the form "if errors A and B occur within five seconds, then error C occurs within 30 s with probability 0.8." Several

parameters such as the maximum length of the data window, types of error messages, and ordering requirements have to be prespecified. However, the algorithm returned too many rules such that they had to be presented to human operators with system knowledge in order to filter out informative ones.

Weiss [1999] introduced a failure prediction technique called *timeweaver* that based on a genetic training algorithm. In contrast to searching and selecting patterns that exist in the database, rules are generated "from scratch" by the use of a simple language: error events are connected with three types of ordering primitives. The genetic algorithm starts with an initial set of rules (initial population) and repetitively applies crossing and mutation operations to generate new rules. The quality of the obtained candidates is assessed using a special fitness function that incorporates both prediction quality (based on a variant of the F-measure, that allows one to adjust the relative weight of precision and recall) and diversity of the rule set. After generating a rule set with the genetic algorithm, the rule set is pruned in order to remove redundant patterns. The approach was applied to telecommunication equipment failure data and results compared to three standard machine learning algorithms: C4.5rules [Quinlan 1993], RIPPER [Cohen 1995], and FOIL [Quinlan 1990]. Please refer to online Appendix G for further details on timeweaver.

Vilalta and Ma [2002] described a data-mining approach that is tailored to short-term prediction of Boolean data building on a concept termed *eventsets*. The method searches for predictive subsets of events occurring prior to a target event. In the terminology used here, events refer to error reports and target events to failures. The method addresses the issue of class skewness (failures occur very rarely in comparison to nonfailure events) by first considering only failure-prone sequences, and by incorporating nonfailure-prone data in a later step. More precisely, the method is divided into three steps: first, frequent subsets of error events preceding a failure are extracted. Second, subsets that occur frequently before failures but also frequently when no failure occurs are removed by a threshold on confidence and by applying a statistical test. In a final third step, overly general sets are removed. Please refer to online Appendix H for further details on this approach. The eventset method has been applied to failure prediction in a 350-node cluster system, as described in Sahoo et al. [2003] (Category 2.4.3).

5.3.1.1. Other Potential Approaches. Fault trees were developed in the 1960s and have become a standard tool reliability modeling. A comprehensive treatment of fault trees was given, for example, by Vesely et al. [1981]. The purpose of fault trees is to model conditions under which failures can occur using logical expressions. Expressions are arranged in the form of a tree, and probabilities are assigned to the leaf nodes, facilitating the computation of the overall failure probability. Fault tree analysis is a statical analysis that does not take the current system status into account. However, if the leaf nodes are combined with online fault detectors, and logical expressions are transformed into a set of rules, they can be used as a rule-based online failure predictor. Although such an approach has been applied to chemical process failure prediction [Ulerich and Powers 1988] and power systems [Rovnyak et al. 1994], we have not found the approach being applied to computer systems.

Bai et al. [2005] employed a Markov Bayesian Network for reliability prediction but a similar approach might work for online failure prediction as well. The same holds for decision tree methods: upcoming failures can be predicted if error events are classified using a decision tree approach similar to that proposed by Kiciman and Fox [2005] (see Category 2.2.3). In this case, however, the decision tree would have to classify error reports rather than monitored runtime paths.

Regarding data mining, several developments could potentially be used to further improve data mining-based online failure prediction: sequential pattern mining and

the use of ontologies, as described in, for example, Srikant and Agrawal [1996], or path traversal patterns (see, e.g., Chen et al. [1998]), which could be applied in transaction-based systems.

5.3.2. Cooccurrence (3.2). Levy and Chillarege [2003] analyzed data of an industrial voice mail system and identified three characteristics they called *principles*, two of which support the assumptions on which failure prediction approaches in this category are based: principle 1 ("counts tell") again emphasizes the property that the number of errors (since the article is about a telecommunication system, the authors used the term *alarm* for what is termed an *error*, here) per time unit increases before a failure. Principle 2 ("the mix changes") is described in Category 3.4. Principle 3 ("clusters form early") basically is the same as principle 1 but puts more emphasis on the fact that for common failures the effect is even more apparent if errors are clustered into groups. Similar observations have been made by Liang et al. [2006]: the authors analyzed jobs of an IBM BlueGene/L supercomputer and supported the following thesis: "On average, we observe that if a job experiences two or more nonfatal events after filtering, then there is a 21.33% chance that a fatal failure will follow. For jobs that only have one nonfatal event, this probability drops to 4.7%" (p. 444).

According to Siewiorek and Swarz [1998], page 262, Nassar and Andrews [1985] were one of the first to propose two ways of failure prediction based on the occurrence of error reports. The first approach investigates the distribution of error types. If the distribution of error types changes systematically (i.e., one type of error occurs more frequently), a failure is supposed to be imminent. The second approach investigates error distributions for all error types obtained for intervals between crashes. If the error generation rate increases significantly, a failure is looming. Both approaches result in the computation of threshold values upon which a failure warning can be issued.

The dispersion frame technique (DFT) developed by Lin and Siewiorek [1990] uses a set of heuristic rules on the time of occurrence of consecutive error events of each component to identify looming permanent failures. A detailed description of the DFT can be found in online Appendix I.

Lal and Choi [1998] showed plots and histograms of errors occurring in a UNIX Server. The authors proposed aggregating errors in an approach similar to tupling (c.f. Tsao and Siewiorek [1983]) and stated that the frequency of clustered error occurrence indicates an upcoming failure. Furthermore, they showed histograms of error occurrence frequency over time before failure.

More recently, Leangsuksun et al. [2004] presented a study where hardware sensors measurements such as fan speed, temperature, etc., are aggregated using several thresholds to generate error events with several levels of criticality. These events are analyzed in order to eventually generate a failure warning that can be processed by other modules. The study was carried out on data of a high-availability, high-performance Linux cluster.

5.3.3. Pattern Recognition (3.3). Pattern recognition techniques operate on sequences of error events trying to identify patterns that indicate a failure-prone system state.

Methods such as eventset [Vilalta et al. 2002] investigate the type of error reports (e.g., the error message ID) while methods such as the dispersion frame technique [Lin and Siewiorek 1990] focus on the time when errors are detected. If both parts of error reports—time and type—are considered together, the sequence of error reports turns into a temporal sequence. Salfner and colleagues proposed two failure prediction methods that identify patterns in temporal sequences.

Salfner et al. [2006] presented Similar Events Prediction (SEP), which is a semi-Markov chain model. Each error report event corresponds to a state in the semi-Markov

chain while the time between two reports is represented by the continuous-time state transition duration of the semi-Markov chain using uniform distributions. Sequence similarity is computed by the product of state traversal probabilities. Training of the model involves hierarchical clustering of error sequences leading to a failure and the computation of relative frequencies to estimate state transition probabilities. The article included experiments with data from an industrial telecommunication system.

Salfner and Malek [2007] proposed using hidden semi-Markov models (HSMM) in order to add one additional level of flexibility. Associating error detections with observations that are generated by the states of the HSMM, errors may be mapped to groups of hidden states. With HSMMs, similarity of error sequences can be computed by the use of the forward algorithm, which is an efficient dynamic programming algorithm to compute the sequence likelihood. One advantage of HSMMs in comparison to Salfner et al. [2006] is that HSMMs can handle permutations in the input event sequences. In addition to the classification step depicted in Figure 14, which is an evaluation of failure probability for some fixed point in time in the future, failure prediction can also be accomplished by computing the time-to-failure distribution (c.f. Salfner [2006]). Salfner [2008] provided more details on the HSMM method, including a comparative analysis of HSMM with DFT, eventset method, and SVD-SVM (see Category 3.5). The study also included a comparison of computational overhead both for the training and application of the methods.

5.3.3.1. Other Potential Approaches. Computing the similarity between sequences is one of the key tasks in biological sequence analysis [Durbin et al. 1998, Chapter 2], which is called *pairwise alignment*. Various algorithms have been developed such as the Needleman-Wunsch algorithm [Needleman and Wunsch 1970] the Smith-Waterman algorithm [Smith and Waterman 1981], or the BLAST algorithm [Altschul et al. 1990]. The outcome of such algorithms is usually a score evaluating the alignment of two sequences. If used as a similarity measure between the sequence under investigation and known failure sequences, failure prediction can be accomplished as depicted in Figure 14. One of the advantages of alignment algorithms is that they build on a substitution matrix providing scores for the substitution of symbols. In terms of error event sequences, this technique has the potential to define a score for one error event being "replaced" by another event, giving rise to the use of a hierarchical grouping of errors, as proposed in Salfner et al. [2004].

In Category 2.3.3, the method of Kiciman and Fox [2005] was described, which used probabilistic context-free grammars (PCFG) to analyze runtime paths. However, runtime paths contain symptoms rather than errors. Nevertheless, PCFGs could also be used to perform error report-based failure prediction: following the approach depicted in Figure 14, PCFGs can be used to compute sequence similarity to error report sequences that have lead to a failure in the training dataset, and to compute sequence similarity to error report sequences that usually do not lead to a failure.

A further well-known stochastic speech modeling technique is *n-gram models* [Manning and Schütze 1999, page 192]. n-grams represent sentences by conditional probabilities taking into account a context of up to n words in order to compute the probability of a given sentence.[8] Conditional densities are estimated from training data. Transferring this concept to failure prediction, error events correspond to words and error sequences to sentences. If the probabilities (the "grammar") of an n-gram model were estimated from failure sequences, high sequence probabilities would translate into "failure-prone" and low probabilities into "not failure-prone".

[8]Although, in most applications of statistical natural language processing, the goal is to predict the next word using $P(w_n|w_1, \ldots, w_{n-1})$, the two problems are connected via the theorem of conditional probabilities.

5.4. Statistical Tests (3.4)

Principle 2 ("the mix changes") in Levy and Chillarege [2003] mentioned in Category 3.2 delineates the discovery that the order of subsystems sorted by error generation frequency changes prior to a failure. According to the article, relative error generation frequencies of subsystems follow a Pareto distribution: most errors are generated by only a few subsystems while most subsystems generate only very few errors (which is also known as *Zipf's law* [Zipf 1949]). The proposed failure prediction algorithm monitors the order of subsystems and predicts a failure if it changes significantly, which basically is a statistical test. Using data of the voice mail application analyzed, the authors showed examples where the change in order can be observed; however, no results with respect to false negatives or false positives were reported.

5.5. Classifiers (3.5)

Classifiers usually associate an input vector with a class label. In this Category 3, input data consists of one or more error reports that have to be represented by a vector in order to be processed by a classification algorithm. A straightforward solution would be to use the error type of the first event in a sequence as the value of the first input vector component, the second type as the second component, and so on. However, it turns out that such a solution does not the work: if the sequence is shifted only one step, the sequence vector is orthogonally rotated in the input space and most classifiers will not judge the two vectors as similar. One solution to this problem has been proposed by Domeniconi et al. [2002]: SVD-SVM (Singular-Value-Decomposition and Support-Vector-Machine) borrows a technique known from information retrieval: the so-called "vector space model" representation of texts [Manning and Schütze 1999, page 539]. In the vector space model representation, there is a dimension for each word of the language. Each text is a point in this high-dimensional space where the magnitude along each dimension is defined by the number of occurrences of the specific word in the text. SVD-SVM applies the same technique to represent error event sequences: Each dimension of the vector corresponds to one event type and the magnitude is either a Boolean value indicating whether an error type is present or not, the number of occurrences of the event type in the data window, or a binary encoded timestamp. SVD-SVM performs two steps: the first step involves the simplification and decorrelation of event sequences by the use of singular value decomposition. The second step involves classification by the use of support vector machines. The approach has been applied to data of a production computer network where failures were defined to be the occurrence of an error event with severity level "critical" or "fatal." Further details on this method can be found in online Appendix K.

6. SUMMARY AND CONCLUSIONS

We have surveyed failure prediction methods that have been proposed and many of them used to predict failures of computer systems online. The goal of online failure prediction is to identify if a failure, which is a misbehavior of the system resulting in a deviation from the expected output, will occur in the near future. This can be accomplished by runtime observation and measurement of the current system state. Although failure prediction approaches exist that base predictions on upcoming failures without measuring the current system state (e.g., by lifetime probability distributions), these are beyond the scope of this survey.

We have developed a comprehensive taxonomy in order to structure and classify a wide spectrum of existing techniques dealing with online failure prediction to help potential users to easily find a method that might be attractive in their application

environment. Online failure prediction methods can be divided into four categories according to the type of input data that is processed. Specifically, the approaches evaluate (a) the times of previous failure occurrence (failure tracking), (b) monitoring data reflecting symptoms (side effects) of errors, (c) the detection of errors that have not yet evolved to become a failure, or (d) the search for undetected errors by means of auditing. Three of the categories have been divided further by the principle approach they employ and subsequently by the methods they use. Concepts have been briefly explained, followed by appropriate references.

We have surveyed almost 50 approaches from various research areas such as pattern recognition, control theory, and function approximation. The majority of described approaches focus on failure prediction concerning software but there are also many methods predicting failures of hardware, for example, Hughes et al. [2002], Hamerly and Elkan [2001], Murray et al. [2003], or Weiss [1999].

A comprehensive summary of all models/approaches together with demonstrated applications including appropriate references and classification can be found in Table III. In addition, the table provides a brief synopsis of each approach and its potential application areas, along with remarks about it. In order to shorten the table, the following numbers are used for text that appears several times across the table:

① Failure-based long term prediction;
② Failure prediction in distributed systems;
③ Prediction of resource-scarcity failures;
④ Prediction of service level failures;
⑤ Single request failure prediction;
⑥ Component failure prediction;
⑦ Event-based failure prediction (as a superset to error reports).

In order to provide better insight into categories, the online appendices describe various approaches that are representative for each category. Table IV online Appendix A lists the approaches that are described in online Appendices B to K. *Classification* refers to the category of our taxonomy the approach is assigned to; *application area* denotes the areas the method was or could be implemented in, the metrics indicates used to assess the quality of the corresponding failure prediction approach;' and, finally, *failure model* names the failures and/or faults that can be predicted by the examined technique. Most of these techniques have been tested and evaluated in practice.

Let us reiterate and summarize: dependability and resilience are and will remain a permanent challenge for the following reasons:

—ever-increasing systems complexity,
—ever-growing number of attacks and threats,
—novice users,
—third-party, open-source software, commercial-off-the-shelf (COTS) components,
—growing connectivity and interoperability,
—dynamicity (frequent configurations, reconfigurations, updates, upgrades and patches, ad hoc extensions), and
—systems proliferation to applications in all domains of human activity.

In such circumstances, proactive fault management by runtime monitoring and online failure prediction seem to be one of a very few alternatives for effective online dependability assessment and enhancement. In our opinion, proactive fault management

Table III. Overview of All Failure Prediction Techniques Surveyed

Category/reference	Model/approach	Area of application	Potential applications/remarks
1.1.1 Csenki [1990]	Jeilnski-Moranda model improved by observed interfailure times	Software reliability prediction	① Based on software quality
1.1.2 Pfefferman and Cernuschi-Frias [2002]	Failures modeled as nonstationary Bernoulli process	Software failures	① Adapts to changing system dynamics
1.2 Liang et al. [2006]	Temporal / spatial compression of failures	Extreme-scale parallel systems	② Focus on long-running applications
2.1.1 Vaidyanathan and Trivedi [1999]	Semi-Markov reward model to predict resource consumption	Memory consumption in UNIX system	③, ④ Focus on workload
2.1.1 Li et al. [2002]	ARX model for resource utilization using additional system variables as input	Prediction of Apache web server resource exhaustion	③, ④ Accounts for stationary and nonstationary effects
2.1.2 Andrzejak and Silva [2007]	Use of deterministic functions to approx. aging indicators as a function of workload metrics	Performance of Apache Axis SOAP server	③ Focus on workload-based input data
2.1.3 Hoffmann [2006]	Approximation of interval call availability by universal basis functions	Performance failures of a telecommunication system	③, ④ Also applied to response time and memory prediction in Apache web server [Hoffmann et al. 2007]
2.1.3 Kapadia et al. [1999]	Approximation of resource usage by averaging values of similar points in the training data	Resource consumption (e.g., CPU time) of programs based on invocation parameters	③, ④ Does not regard interference with other programs
2.1.3 / 1.2 Fu and Xu [2007]	Estimation of number of failures from CPU utilization and temporal and spatial correlation by use of neural networks	Failures of Wayne State University computing grid	②, ④ Focus on number of failures
2.1.3 Abraham and Grosan [2005]	Genetically generating code to approximate failure probability as a function of external stressors (e.g. temperature)	Power circuit failures of an electronic device	Applicable to systems where the root cause of failures can be assigned to a small set of stressors
2.2.1 Hamerly and Elkan [2001]	Naïve Bayes classification of SMART values	Hard-disk failures	Based on independence assumption of monitoring variables. ⇒ variable selection or decorrelation required
2.3.2 Murray et al. [2003]	Unsupervised clustering in order to determine areas of SMART values where the disk operates normally	Failures of hard disk drives	④ Number of clusters needs to be specified

Continued on next page

Table III. *Continued*

Category/reference	Model/approach	Area of application	Potential applications/remarks
2.2.1 Pizza et al. [1998]	Bayes-optimal discrimination between permanent and transient faults	Simulated component diagnosis	④ Failure prediction where short outages do not matter while permanent outages lead to a failure
2.2.2 Turnbull and Alldrin [2003]	Radial basis function network to classify a sequence of monitoring values	Motherboard failure prediction based on temperature and voltage	③, ④ Results refer to data set with equally likely failures and nonfailures
2.2.2 Berenji et al. [2003]	RBF rule base obtained by clustering from simulated training data	Hybrid combustion facility	Operates on continuous input and output of components
2.2.2 Murray et al. [2003]	Support vector machine classification of SMART values	Hard-disk failures	Achieves better recall but at the cost of higher false positive rate than Hughes et al. [2002]
2.2.2 Bodík et al. [2005]	Naïve Bayes classifier applied to hit frequencies of most frequently used Web pages	Detection of non-fail-stop failures in a large Internet application	②, ④ Focus on access statistics
2.2.3 Kiciman and Fox [2005]	Decision tree to classify whether requests are successful or not from recorded runtime paths	Identifies faulty components in J2EE application server	Based on runtime properties such as component IDs used, etc.
2.2.3 Daidone et al. [2006]	Use of HMMs to infer true state of components from sequences of component probing results	Identifies whether a component is faulty (simulations)	Based on unreliable probing
2.3.1 Elbaum et al. [2003]	Detection of unknown sequences of functional events such as function calls, etc.	Failures of email client "pine"	Single-threaded applications
2.3.1 Hughes et al. [2002]	Statistical rank sum test of monitored values in comparison to stored fault-free samples of SMART values	Failures of hard-disk drives	Monitoring variables must relate closely to failures; little computational overhead
2.3.1 Chen et al. [2002]	Pinpoint: identification of failure-correlated components by clustering of runtime paths	Failed components of a J2EE application	④, ⑤ Applicable if requests traverse many components
2.3.1 Bodík et al. [2005]	χ^2 test-based comparison of hit frequencies of Web pages to a "historically normal" distribution	Prediction of non-fail-stop failures of major Internet site	②, ④ Failure prediction based on usage statistics; applicable if significant number of usage counts is available
2.3.3 Ward et al. [1998]	Comparison of current measurement to normal distribution using a statistical test	TCP connections of proxy servers	③, ④ Works for failures indicated by a single variable; assumption of stationary process

Continued on next page

Table III. *Continued*

Category/reference	Model/approach	Area of application	Potential applications/remarks
2.3.3 Chen et al. [2004]	Probability of runtime paths computed using PCFGs	Request failure detection in J2EE appl. server, Tellme enterprise voice appl. network, and eBay online auction system	④, ⑤ Component-based systems
2.3.4 Kiciman and Fox [2005]	Comparison of component interactions to reference "peer model" using χ^2 test	Detects anomalous component instances in J2EE application server	⑥ Densly connected component interaction graph required
2.3.5 Singer et al. [1997]	Estimated state computed by MSET is compared to the measured state by sequential probability ratio test	Detection whether a memory consuming fault injector is active or not, and detections software aging in transaction processing servers	④ Focus on correlated multivariate system variables; details not publicly available due to patent restrictions
2.4.1 Garg et al. [1998]	Smoothing of monitored resources, seasonal Kendall test, trend estimation	Memory, file table size, process table size in UNIX system	③ Works for systems where time-based (rather than workload-based) approach is appropriate
2.4.1 Castelli et al. [2001]	Fitting of several curve types, selecting the best, extrapolation until resource exhaustion	IBM xSeries Software Rejuvenation Agent	③ Li et al. [2002] showed inferior performance compared to ARX modeling
2.4.1 Cheng et al. [2005]	Fuzzy logic to detect "unhealthy state," linear regression to determine mean time to resource exhaustion	High-availability cluster system	③ Sufficiently linear resource evolution required
2.4.2 Shereshevsky et al. [2003]	Detection of software aging by use of the Hölder exponent and subsequent Shewhart change detection algorithm	Predicts system crashes of Windows NT 4.0 and Windows 2000 servers	Predicts any software aging-related failures
2.4.3 Hellerstein et al. [1999]	Application of several time series models to remove stationary and nonstationary effects; predicts failure by threshold violation	Predict number of HTTP operations on production server	③ Requires sufficient amount of data in order to identify station-ary/nonstationary effects
2.4.3 Vilalta et al. [2002]	Stochastic time series model accounting for time-of-the-day, weekly, and monthly effects; generalized likelihood ratio test for change-point detection	Forecasting Web server workload by predicting the number of HTTP operations per second	③ Requires sufficient amount of data in order to identify station-ary /nonstationary effects
2.4.3 Sahoo et al. [2003]	Various linear time series models including ARMA to predict e.g., percentage of system utilization, idle time, network IO	350-node cluster	②, ③
2.4.3 Meng et al. [2007]	Rough set wavelet network to predict next monitoring value	Memory consumption of J2EE server	③ One step ahead prediction ⇒ lead-time equal to monitoring interval

Continued on next page

Table III. *Continued*

Category/reference	Model/approach	Area of application	Potential applications/remarks
2.4.3 Ning et al. [2006]	Fuzzy wavelet network to predict next monitoring value	Memory consumption of J2EE server	Similar to Meng et al. [2007]
3.1 Hätönen et al. [1996]	Episode rules determined by data mining followed by manual selection of rules	Telecommunication Alarm Sequence Analyzer (TASA)	⑦ Profound system knowledge required; not suitable for dynamic systems due to manual rule selection
3.1 Weiss [1999]	Timeweaver: rule base generation from scratch using genetic programming	Telecommunication equipment failure data	⑦ Events must contain several attributes; limited expressiveness in terms of temporal properties
3.1 Vilalta and Ma [2002]	Extraction of event sets that indicate an upcoming failure by data mining	Failure prediction in a 350-node cluster system	⑦ Focus on class imbalance
3.2 Nassar and Andrews [1985]	Detection of changes / trends in distribution of error types	Event log data of three DEC computing systems	⑦ Estimation of distributions requires long time windows ⇒ useful for slowly developing failures
3.2 Lin and Siewiorek [1990]	DFT: Heuristic rules investigating time of error occurrence	Error log of distributed machines running the Andrew file system at Carnegie Mellon University	② ⑦ Builds on system components; heuristic rules need to be adapted
3.2 Lal and Choi [1998]	Aggregation of errors and trend detection in the frequency of occurrence	Errors and failures in a UNIX server	⑦ Very simple method with little computational overhead
3.2 Leangsuksun et al. [2004]	Thresholds on sensor measurements generate events; predicts a failure if frequency of aggregated events exceeds threshold	High-availability, high-performance Linux cluster	Simple heuristic method; no results presented
3.3 Salfner et al. [2006]	SEP: error report sequences modeled using a semi Markov model	Performance failures of a telecommunication system	⑦ Includes both type and time of error reports
3.3 Salfner and Malek [2007]	Model error report sequences using hidden semi-Markov models (HSMM)	Performance failures of a telecommunication system	⑦ Includes both type and time of error reports; can handle permutations in event sequences
3.4 Levy and Chillarege [2003]	Detection of changes in subsystem order built from their error generation frequency	Comverse Voice Mail system	②, ④, ⑥ Significant amount of errors without successive failures needed for reliable estimation of component order
3.5 Domeniconi et al. [2002]	SVD-SVM: using techniques from natural language processing; classifying error sequences using support vector machines	Production computer network with 750 hosts	⑦

(where failure prediction plays a crucial role) will be the key enabler for the next generation of dependability improvement. Researchers have adopted various concepts from computer science resulting in a diverse landscape of approaches to online failure prediction. The goal of this survey is to provide an overview of existing approaches including key properties and fields of application in order to prepare researchers for the next challenge of proactive fault handling: the prediction-based triggering of effective countermeasures to enhance system availability by potentially an order of magnitude or more [Candea et al. 2004; Salfner et al. 2005] by using effective failure avoidance and downtime minimization methods.

ACKNOWLEDGMENTS

We would like to express our gratitude to the reviewers who provided several constructive comments that helped to improve this survey.

REFERENCES

ABRAHAM, A. AND GROSAN, C. 2005. Genetic programming approach for fault modeling of electronic hardware. In *Proceedings of the IEEE Congress on Evolutionary Computation* (CEC). Edinburgh, U.K., Vol. 2, 1563–1569.

AITCHISON, J. AND DUNSMORE, I. R. 1975. *Statistical Prediction Analysis*. Cambridge University Press, Cambridge, U.K.

ALTMAN, D. G. 1991. *Practical Statistics for Medical Research*. CRC Press, Boca Raton, FL.

ALTSCHUL, S., GISH, W., MILLER, W., MYERS, E., AND LIPMAN, D. 1990. Basic local alignment search tool. *J. Molec. Biol. 215*, 3, 403–410.

ANDRZEJAK, A. AND SILVA, L. 2007. Deterministic models of software aging and optimal rejuvenation schedules. In *Proceedings of the 10th IEEE/IFIP International Symposium on Integrated Network Management* (IM). 159–168.

AVIŽIENIS, A. AND LAPRIE, J.-C. 1986. Dependable computing: From concepts to design diversity. *Proc. IEEE 74*, 5 (May), 629–638.

AVIŽIENIS, A., LAPRIE, J.-C., RANDELL, B., AND LANDWEHR, C. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secure Comput. 1*, 1, 11–33.

BABAOGLU, O., JELASITY, M., MONTRESOR, A., FETZER, C., LEONARDI, S., VAN MOORSEL A., AND VAN STEEN, M., EDS. 2005. *Self-Star Properties in Complex Information Systems*. Lecture Notes in Computer Science, vol. 3460. Springer-Verlag, Berlin, Germany.

BAI, C. G., HU, Q. P., XIE, M., AND NG, S. H. 2005. Software failure prediction based on a Markov Bayesian network model. *J. Syst. Softw. 74*, 3 (Feb.), 275–282.

BASSEVILLE, M. AND NIKIFOROV, I. 1993. *Detection of Abrupt Changes: Theory and Application*. Prentice Hall, Englewood Cliffs, NJ.

BERENJI, H., AMETHA, J., AND VENGEROV, D. 2003. Inductive learning for fault diagnosis. In *Proceedings of the IEEE 12th International Conference on Fuzzy Systems* (FUZZ). Vol. 1.

BLISCHKE, W. R. AND MURTHY, D. N. P. 2000. *Reliability: Modeling, Prediction, and Optimization*. Probability and Statistics Series. John Wiley and Sons, New York, NY.

BODIK, P., FRIEDMAN, G., BIEWALD, L., LEVINE, H., CANDEA, G., PATEL, K., TOLLE, G., HUI, J., FOX, A., JORDAN, M. I., AND PATTERSON, D. 2005. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization. In *Proceedings of the IEEE International Conference on Autonomic Computing* (ICAC). IEEE Computer Society Press, Los Alamitos, CA, 89–100.

BROCKLEHURST, S. AND LITTLEWOOD, B. 1996. Techniques for prediction analysis and recalibration. In *Handbook of Software Reliability Engineering*, M. R. Lyu, Ed. McGraw-Hill, New York, NY, Chapter 4, 119–166.

BROWN, A. AND PATTERSON, D. 2001. Embracing failure: A case for recovery-oriented computing (ROC). In *Proceedings of the High Performance Transaction Processing Symposium*.

CANDEA, G., KAWAMOTO, S., FUJIKI, Y., FRIEDMAN, G., AND FOX, A. 2004. Microreboot—a technique for cheap recovery. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*. 31–44.

CANDEA, G., KICIMAN, E., KAWAMOTO, S., AND FOX, A. 2006. Autonomous recovery in componentized Internet applications. *Cluster Comput. 9*, 2, 175–190.

CANDEA, G., KICIMAN, E., ZHANG, S., KEYANI, P., AND FOX, A. 2003. Jagr: An autonomous self-recovering application server. In *Proceedings of the 5th International Workshop on Active Middleware Services* (Seattle, WA).

CASSIDY, K. J., GROSS, K. C., AND MALEKPOUR, A. 2002. Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers. In *Proceedings of the Conference on Dependable Systems and Networks* (DSN). 478–482.

CASTELLI, V., HARPER, R., P., H., HUNTER, S., TRIVEDI, K., VAIDYANATHAN, K., AND ZEGGERT, W. 2001. Proactive management of software aging. *IBM J. Res. Develop. 45*, 2 (Mar.), 311–332.

CAVAFY, C. P. 1992. But the wise perceive things about to happen. In *Collected Poems*, G. Savidis, Ed. Princeton University Press, Princeton, NJ.

CHEN, M., ACCARDI, A., LLOYD, J., KICIMAN, E., FOX, A., PATTERSON, D., AND BREWER, E. 2004. Path-based failure and evolution management. In *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation* (NSDI, San Francisco, CA).

CHEN, M., KICIMAN, E., FRATKIN, E., FOX, A., AND BREWER, E. 2002. Pinpoint: Problem determination in large, dynamic Internet services. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks, IPDS track* (DSN). IEEE Computer Society Press, Los Alamitos, CA, 595–604.

CHEN, M.-S., PARK, J. S., AND YU, P. S. 1998. Efficient data mining for path traversal patterns. *IEEE Trans. Knowl. Data Eng. 10*, 2, 209–221.

CHENG, F., WU, S., TSAI, P., CHUNG, Y., AND YANG, H. 2005. Application cluster service scheme for near-zero-downtime services. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 4062–4067.

CLEVELAND, W. ET AL. 1979. Robust locally weighted regression and smoothing scatterplots. *J. Amer. Statist. Assoc. 74*, 368, 829–836.

COHEN, W. W. 1995. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*. 115–123.

COLEMAN, D. AND THOMPSON, C. 2005. Model based automation and management for the adaptive enterprise. In *Proceedings of the 12th Annual Workshop of the HP OpenView University Association*. 171–184.

CROWELL, J., SHERESHEVSKY, M., AND CUKIC, B. 2002. Using fractal analysis to model software aging. Tech. rep. West Virginia University, Lane Department of CSEE, Morgantown, WV. May.

CSENKI, A. 1990. Bayes predictive analysis of a fundamental software reliability model. *IEEE Trans. Reliab. 39*, 2 (Jun.), 177–183.

DAIDONE, A., DI GIANDOMENICO, F., BONDAVALLI, A., AND CHIARADONNA, S. 2006. Hidden Markov models as a support for diagnosis: Formalization of the problem and synthesis of the solution. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems* (SRDS, Leeds, U.K.).

DEERWESTER, S., DUMAIS, S., FURNAS, G., LANDAUER, T., AND HARSHMAN, R. 1990. Indexing by latent semantic analysis. *J. Amer. Soc. Inform. Sci. 41*, 6, 391–407.

DENSON, W. 1998. The history of reliability prediction. *IEEE Trans. Reliab. 47*, 3 (Sep.), 321–328.

DISCENZO, F., UNSWORTH, P., LOPARO, K., AND MARCY, H. 1999. Self-diagnosing intelligent motors: a key enabler for nextgeneration manufacturing systems. In *Proceedings of the IEEE Colloquium on Intelligent and Self-Validating Sensors*.

DOMENICONI, C., PERNG, C.-S., VILALTA, R., AND MA, S. 2002. A classification approach for prediction of target events in temporal sequences. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'02)*, T. Elomaa, H. Mannila, and H. Toivonen, Eds. Lecture Notes in Artificial Intelligence, vol. 2431. Springer-Verlag, Heidelberg, Germany, 125–137.

DURBIN, R., EDDY, S. R., KROGH, A., AND MITCHISON, G. 1998. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, U.K.

ELBAUM, S., KANDURI, S., AND AMSCHLER, A. 2003. Anomalies as precursors of field failures. In *Proceedings of the 14th IEEE International Symposium on Software Reliability Engineering* (ISSRE). 108–118.

FARR, W. 1996. Software reliability modeling survey. In *Handbook of Software Reliability Engineering*, M. R. Lyu, Ed. McGraw-Hill, New York, NY, Chapter 3, 71–117.

FAWCETT, T. 2004. ROC graphs: Notes and practical considerations for researchers. *Mach. Learn. 31*, 1–38.

FLACH, P. A. 2004. The many faces of ROC analysis in machine learning. Tutorial at the International Conference on Machine Learning (ICML'04). `http://www.cs.bris.ac.uk/flach/ICML04tutorial/`.

FU, S. AND XU, C.-Z. 2007. Quantifying temporal and spatial fault event correlation for proactive failure management. In *Proceedings of the IEEE Symposium on Reliable and Distributed Systems* (SRDS).

GARG, S., VAN MOORSEL, A., VAIDYANATHAN, K., AND TRIVEDI, K. S. 1998. A methodology for detection and estimation of software aging. In *Proceedings of the 9th International Symposium on Software Reliability Engineering* (ISSRE).

GROSS, K. C., BHARDWAJ, V., AND BICKFORD, R. 2002. Proactive detection of software aging mechanisms in performance critical computers. In *SEW '02: Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop* (SEW-27'02). IEEE Computer Society Press, Los Alamitos, CA.

GROTTKE, M., MATIAS, R., AND TRIVEDI, K. S. 2008. The fundamentals of software aging. In *Proceedings of the IEEE Workshop on Software Aging and Rejuvenation* (Seattle, WA).

GROTTKE, M. AND TRIVEDI, K. 2007. Fighting bugs: Remove, retry, replicate, and rejuvenate. *IEEE Comput. 40*, 107–109.

GUYON, I. AND ELISSEEFF, A. 2003. An introduction to variable and feature selection. *J. Mach. Learn. Res. 3*, 1157–1182. Special Issue on Variable and Feature Selection.

HAMERLY, G. AND ELKAN, C. 2001. Bayesian approaches to failure prediction for disk drives. In *Proceedings of the Eighteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers, San Francisco, CA, 202–209.

HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer Verlag, Berlin, Germany.

HÄTÖNEN, K., KLEMETTINEN, M., MANNILA, H., RONKAINEN, P., AND TOIVONEN, H. 1996. Tasa: Telecommunication alarm sequence analyzer, or: How to enjoy faults in your network. In *Proceedings of the IEEE Network Operations and Management Symposium* (Kyoto, Japan). Vol. 2., 520–529.

HELLERSTEIN, J. L., ZHANG, F., AND SHAHABUDDIN, P. 1999. An approach to predictive detection for service management. In *Proceedings of the 6th IEEE International Symposium on Integrated Network Management*. 309–322.

HO, D. W. C., ZHANG, P. A., AND XU, J. 2001. Fuzzy wavelet networks for function learning. *IEEE Trans. Fuzzy Syst. 9*, 1, 200–211.

HOFFMANN, G. A. 2004. Adaptive transfer functions in radial basis function (RBF) networks. In *Proceedings of the 4th International Conference on Computational Science* (ICCS 2004), M. Bubak, G. D. van Albada, P. M. A. Sloot, et al., Eds. Lecture Notes in Computer Science, vol. 3037. Springer-Verlag, Berlin, Germany, 682–686.

HOFFMANN, G. A. 2006. *Failure Prediction in Complex Computer Systems: A Probabilistic Approach*. Shaker Verlag, Herzogexrath, Germany.

HOFFMANN, G. A. AND MALEK, M. 2006. Call availability prediction in a telecommunication system: A data driven empirical approach. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems* (SRDS), Leeds, U.K.

HOFFMANN, G. A., SALFNER, F., AND MALEK, M. 2004. Advanced failure prediction in complex software systems. Res. rep. 172, Department of Computer Science, Humboldt University, Berlin, Germany. `www.rok.informatik.hu-berlin.de/Members/salfner`.

HOFFMANN, G. A., TRIVEDI, K. S., AND MALEK, M. 2006. A best practice guide to resource forecasting for the Apache Webserver. In *IEEE Proceedings of the 12th International Symposium Pacific Rim Dependable Computing* (PRDC). University of California, Riverside, Riverside, CA.

HOFFMANN, G. A., TRIVEDI, K. S., AND MALEK, M. 2007. A best practice guide to resource forecasting for computing systems. *IEEE Trans. Reliab. 56*, 4 (Dec.), 615–628.

HORN, P. 2001. Autonomic computing: IBM's perspective on the state of information technology. Tech. rep. IBM, Yorktown Heights, NY.

HOTELLING, H. 1933. Analysis of a complex of statistical variables into principal components. *J. Educat. Psych. 24*, 417–441.

HUGHES, G., MURRAY, J., KREUTZ-DELGADO, K., AND ELKAN, C. 2002. Improved disk-drive failure warnings. *IEEE Trans. Reliab. 51*, 3 (Sep.), 350–357.

IEC: INTERNATIONAL TECHNICAL COMISSION, ED. 2002. *Dependability and Quality of Service*, 2nd ed. IEC, Geneva, Switzerland, Chapter 191.

IYER, R. K., YOUNG, L. T., AND SRIDHAR, V. 1986. Recognition of error symptoms in large systems. In *Proceedings of the 1986 ACM Fall Joint Computer Conference*. IEEE Computer Society Press, Los Alamitos, CA, 797–806.

JELINSKI, Z. AND MORANDA, P. 1972. Software reliability research. In *Statistical Computer Performance Evaluation*, W. Freiberger, Ed. Academic Press, New York, NY.

KAPADIA, N. H., FORTES, J. A. B., AND BRODLEY, C. E. 1999. Predictive application-performance modeling in a computational gridenvironment. In *Procedings of the 8th International IEEE Symposium on High Performance Distributed Computing*. 47–54.

KICIMAN, E. AND FOX, A. 2005. Detecting application-level failures in component-based Internet services. *IEEE Trans. Neural Netw. 16*, 5 (Sep.), 1027–1041.

KORBICZ, J., KOŚCIELNY, J. M., KOWALCZUK, Z., AND CHOLEWA, W., EDS. 2004. *Fault Diagnosis: Models, Artificial Intelligence, Applications*. Springer-Verlag, Berlin, Germany.

LAL, R. AND CHOI, G. 1998. Error and failure analysis of a unix server. In *Proceedings of the IEEE 3rd International High-Assurance Systems Engineering Symposium* (HASE). IEEE Computer Society Press, Los Alamitos, CA, 232–239.

LAPRIE, J.-C. AND KANOUN, K. 1996. Software reliability and system reliability. In *Handbook of Software Reliability Engineering*, M. R. Lyu, Ed. McGraw-Hill, New York, NY, Chapter 2, 27–69.

LEANGSUKSUN, C., LIU, T., RAO, T., SCOTT, S., AND LIBBY, R. 2004. A failure predictive and policy-based high availability strategy for Linux high performance computing cluster. In *Proceedings of the 5th LCI International Conference on Linux Clusters: The HPC Revolution*. 18–20.

LEVY, D. AND CHILLAREGE, R. 2003. 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*. IEEE Computer Society Press, Los Alamitos, CA.

LI, L., VAIDYANATHAN, K., AND TRIVEDI, K. S. 2002. An approach for estimation of software aging in a Web server. In *Proceedings of the Intenational Symposium on Empirical Software Engineering* (ISESE, Nara, Japan).

LIANG, Y., ZHANG, Y., SIVASUBRAMANIAM, A., JETTE, M., AND SAHOO, R. 2006. Bluegene/l failure analysis and prediction models. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks* (DSN). 425–434.

LIN, T.-T. Y. AND SIEWIOREK, D. P. 1990. Error log analysis: Statistical modeling and heuristic trend analysis. *IEEE Trans. Reliab. 39*, 4 (Oct.), 419–432.

LUNZE, J. 2003. *Automatisierungstechnik*, 1st ed. Oldenbourg, Munich, Germany.

LYU, M. R., ED. 1996. *Handbook of Software Reliability Engineering*. McGraw-Hill, New York, NY.

MANNING, C. D. AND SCHÜTZE, H. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

MELLIAR-SMITH, P. M. AND RANDELL, B. 1977. Software reliability: The role of programmed exception handling. *SIGPLAN Not. 12*, 3, 95–100.

MENG, H., DI HOU, Y., AND CHEN, Y. 2007. A rough wavelet network model with genetic algorithm and its application to aging forecasting of application server. In *Procedings of the IEEE International Conference on Machine Learning and Cybernetics*. Vol. 5.

MUNDIE, C., DE VRIES, P., HAYNES, P., AND CORWINE, M. 2002. Trustworthy computing. Tech. rep., Microsoft Corp., Redmond, WA. Oct.

MURRAY, J., HUGHES, G., AND KREUTZ-DELGADO, K. 2003. Hard drive failure prediction using non-parametric statistical methods. In *Proceedings of ICANN/ICONIP*.

MUSA, J. D., IANNINO, A., AND OKUMOTO, K. 1987. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, New York, NY.

NASSAR, F. A. AND ANDREWS, D. M. 1985. A methodology for analysis of failure prediction data. In *Proceedings of the IEEE Real-Time Systems Symposium*. 160–166.

NEEDLEMAN, S. B. AND WUNSCH, C. D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Molec. Biol. 48*, 3, 443–53.

NEVILLE, S. W. 1998. Approaches for early fault detection in large scale engineering plants. Ph.D. dissertation, University of Victoria, Victoria, B.C., Canada.

NING, M. H., YONG, Q., DI, H., YING, C., AND ZHONG, Z. J. 2006. Software aging prediction model based on fuzzy wavelet network with adaptive genetic algorithm. In *Proceedings of the IEEE 18th IEEE International Conference on Tools with Artificial Intelligence* (ICTAI). IEEE Computer Society Press, Los Alamitos, CA, 659–666.

PARNAS, D. L. 1994. Software aging. In *Proceedings of the 16th IEEE International Conference on Software Engineering* (ICSE). IEEE Computer Society Press, Los Alamitos, CA, 279–287.

PATTERSON, D. A., BROWN, A., BROADWELL, P., CANDEA, G., CHEN, M., CUTLER, J., ENRIQUEZ, P., FOX, A., KICIMAN, E., MERZBACHER, M., OPPENHEIMER, D., SASTRY, N., TETZLAFF, W., TRAUPMAN, J., AND TREUHAFT, N. 2002. Recovery-oriented computing (ROC): Motivation, definition, techniques, and case studies. Tech. rep. UCB//CSD-02-1175. Computer Science Department, University of California, Berkeley, Berkeley, CA. March.

PAWLAK, Z., WONG, S. K. M., AND ZIARKO, W. 1988. Rough sets: Probabilistic versus deterministic approach. *Internat. J. Man-Mach. Stud. 29*, 81–95.

PETTITT, A. 1977. Testing the normality of several independent samples using the anderson-darling statistic. *Appl. Statist. 26*, 2, 156–161.

PFEFFERMAN, J. AND CERNUSCHI-FRIAS, B. 2002. A nonparametric nonstationary procedure for failure prediction. *IEEE Trans. Reliab. 51*, 4 (Dec.), 434–442.

PIZZA, M., STRIGINI, L., BONDAVALLI, A., AND DI GIANDOMENICO, F. 1998. Optimal discrimination between transient and permanent faults. In *Proceedings of the IEEE 3rd International High-Assurance Systems Engineering Symposium* (HASE). IEEE Computer Society Press, Los Alamitos, CA, 214–223.

QUINLAN, J. 1990. Learning logical definitions from relations. *Mach. Learn. 5*, 3, 239–266.

QUINLAN, J. 1993. *C4. 5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, CA.

RABINER, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE 77*, 2 (Feb.), 257–286.

ROVNYAK, S., KRETSINGER, S., THORP, J., AND BROWN, D. 1994. Decision trees for real-time transient stability prediction. *IEEE Trans. Power Syst. 9*, 3, 1417–1426.

SAHNER, R. A., TRIVEDI, K. S., AND PULIAFITO, A. 1996. *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package (The Red Book)*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

SAHOO, R. K., OLINER, A. J., RISH, I., GUPTA, M., MOREIRA, J. E., MA, S., VILALTA, R., AND SIVASUBRAMANIAM, A. 2003. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD). ACM Press, New York, NY, 426–435.

SALFNER, F. 2006. Modeling event-driven time series with generalized hidden semi-Markov models. Tech. rep. 208. Department of Computer Science, Humboldt-Universität zu Berlin, Berlin, Germany. `http://edoc.hu-berlin.de/docviews/abstract.php?id=27653.`

SALFNER, F. 2008. *Event-based Failure Prediction: An Extended Hidden Markov Model Approach*. dissertation.de—Verlag im Internet GmbH, Berlin, Germany. `http://www.rok.informatik.hu-berlin.de/Members/salfner/publications/salfner08event-based.pdf.`

SALFNER, F., HOFFMANN, G. A., AND MALEK, M. 2005. Prediction-based software availability enhancement. In *Self-Star Properties in Complex Information Systems*, O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, van Moorsel A., and M. van Steen, Eds. Lecture Notes in Computer Science, vol. 3460. Springer-Verlag, Berlin, Germany.

SALFNER, F. AND MALEK, M. 2007. Using hidden semi-Markov models for effective online failure prediction. In *Proceedings of the IEEE 26th International Symposium on Reliable Distributed Systems* (SRDS).

SALFNER, F., SCHIESCHKE, M., AND MALEK, M. 2006. Predicting failures of computer systems: A case study for a telecommunication system. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium* (IPDPS, Rhodes Island, Greece).

SALFNER, F., TSCHIRPKE, S., AND MALEK, M. 2004. Comprehensive logfiles for autonomic systems. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium* (IPDPS)*, Workshop on Fault-Tolerant Parallel, Distributed and Network-Centric Systems* (FTPDS). IEEE Computer Society Press, Los Alamitos, CA.

SEN, P. K. 1968. Estimates of the regression coefficient based on Kendall's tau. *J. Amer. Statist. Assoc. 63*, 324 (Dec.), 1379–1389.

SHERESHEVSKY, M., CROWELL, J., CUKIC, B., GANDIKOTA, V., AND LIU, Y. 2003. Software aging and multifractality of memory resources. In *Proceedings of the International Conference on Dependable Systems and Networks* (DSN). IEEE Computer Society Press, Los Alamitos, CA, 721–730.

SIEWIOREK, D. P. AND SWARZ, R. S. 1998. *Reliable Computer Systems*, 3rd ed. A. K. Peters, Ltd., Wellesley, MA.

SINGER, R. M., GROSS, K. C., HERZOG, J. P., KING, R. W., AND WEGERICH, S. 1997. Model-based nuclear power plant monitoring and fault detection: Theoretical foundations. In *Proceedings of the Conference on Intelligent System Application to Power Systems* (ISAP, Seoul, Korea). 60–65.

SMITH, T. AND WATERMAN, M. 1981. Identification of common molecular subsequences. *J. Molec. Biol. 147*, 195–197.

SRIKANT, R. AND AGRAWAL, R. 1996. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology, EDBT*, P. M. G. Apers, M. Bouzeghoub, and G. Gardarin, Eds. Lecture Notes in Computer Science, vol. 1057. Springer-Verlag, Berlin, Germany, 3–17.

TANG, D. AND IYER, R. 1993. Dependability measurement and modeling of a multicomputer system. *IEEE Trans. Comput. 42*, 1 (Jan.), 62–75.

TROUDET, T., MERRILL, W., CENTER, N., AND CLEVELAND, O. 1990. A real time neural net estimator of fatigue life. In *Proceedings of the IEEE International Joint Conference on Neural Networks* (IJCNN). 59–64.

TSAO, M. M. AND SIEWIOREK, D. P. 1983. Trend analysis on system error files. In *Proceedings of the 13th International Symposium on Fault-Tolerant Computing* (Milano, Italy). 116–119.

TURNBULL, D. AND ALLDRIN, N. 2003. Failure prediction in hardware systems. Tech. rep. University of California, San Diego, CA. `http://www.cs.ucsd.edu/~dturnbul/Papers/ServerPrediction.pdf`.

ULERICH, N. AND POWERS, G. 1988. On-line hazard aversion and fault diagnosis in chemical processes: The digraph+fault-tree method. *IEEE Trans. Reliab. 37*, 2 (Jun.), 171–177.

VAIDYANATHAN, K. AND TRIVEDI, K. S. 1999. A measurement-based model for estimation of resource exhaustion in operational software systems. In *Proceedings of the International Symposium on Software Reliability Engineering* (ISSRE).

VAN RIJSBERGEN, C. J. 1979. *Information Retrieval*, second ed. Butterworth, London, U.K.

VAPNIK, V. N. 1995. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, NY.

VESELY, W., GOLDBERG, F. F., ROBERTS, N. H., AND HAASL, D. F. 1981. Fault tree handbook. Tech. rep. NUREG-0492. U.S. Nuclear Regulatory Commission, Washington, DC.

VILALTA, R., APTE, C. V., HELLERSTEIN, J. L., MA, S., AND WEISS, S. M. 2002. Predictive algorithms in the management of computer systems. *IBM Syst. J. 41*, 3, 461–474.

VILALTA, R. AND MA, S. 2002. Predicting rare events in temporal domains. In *Proceedings of the IEEE International Conference on Data Mining* (ICDM). IEEE Computer Society Press, Los Alamitos, CA, 474–482.

WARD, A., GLYNN, P., AND RICHARDSON, K. 1998. Internet service performance failure detection. *SIGMETRICS Perform. Eval. Rev. 26*, 3, 38–43.

WARD, A. AND WHITT, W. 2000. Predicting response times in processor-sharing queues. In *Proceedings of the Fields Institute Conference on Communications Networks*, P. W. Glynn, D. J. MacDonald, and S. J. Turner, Eds.

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

WEISS, G. 2002. Predicting telecommunication equipment failures from sequences of network alarms. In *Handbook of Knowledge Discovery and Data Mining*, W. Kloesgen and J. Zytkow, Eds. Oxford University Press, Oxford, U.K., 891–896.

WONG, K. C. P., RYAN, H., AND TINDLE, J. 1996. Early warning fault detection using artificial intelligent methods. In *Proceedings of the Universities Power Engineering Conference*.

ZIPF, G. K. 1949. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley Press, Cambridge, MA.

Received July 2007; revised June 2008; accepted October 2008