25th IEEE Symposium on Reliable Distributed Systems (SRDS 2006), 2-4 October 2006 in Leeds, United Kingdom

Call Availability Prediction in a Telecommunication System: A Data Driven Empirical Approach

Guenther Hoffmann and Miroslaw Malek
Humboldt-Universität zu Berlin
Institut für Informatik
[gunho|malek@informatik.hu-berlin.de]

Abstract

Availability prediction in a telecommunication system plays a crucial role in its management, either by alerting the operator to potential failures or by proactively initiating preventive measures. In this paper, we apply linear (ARMA, multivariate, random walk) and nonlinear (Radial and Universal Basis Functions) regression techniques to recognize system failures and to predict the system's call availability up to 15 minutes in advance. Secondly we introduce a novel nonlinear modeling technique for call availability prediction. We benchmark all five techniques against each other. The applied modeling methods are data driven rather than analytical and can handle large amounts of data. We apply the modeling techniques to real data of a commercial telecommunication platform. The data used for modeling includes a) time stamped event-based log files and b) continuously measured system states. Results are given in terms of a) receiver operator characteristics (AUC) for classification into classes of failure and non-failure states and b) as a cost-benefit analysis. Our findings suggest a) high degree of nonlinearity in the data, b) statistically significant improved forecasting performance and cost-benefit ratio of nonlinear modeling techniques, and finally finding that c) log file data does not contribute to improve model performance with any modeling technique.

1 Introduction

Over the past decades software systems, including telecommunication systems, have grown in complexity up to a point where their behavior is in parts unpredictable. Software related failures have now become common. The increasing complexity of these systems is seen as a major threat to the benefits these systems aim to provide. Nonetheless, human lives and organizations with considerable economic impact are dependent on the availability of exactly these software infrastructures. Industrial software systems inevitably exhibit failures. Availability prediction during run time is one prerequisite to decrease the system's failure rate and thus increase its availability, e.g. by proactively triggering preventive measures. We identify three steps involved in building smart high availability systems:

1. *Observe*: develop methods that capture and select *essential* data of software system, not all of the many hundreds or even thousands of variables that could be observed actually contribute to failure forecasting.
2. *Reason*: develop methods that interpret that data, recognize malicious system states and predict future system states.
3. *React*: reaction schemes which build on these predictions and help self-manage the system.

In this paper, we focus on the *second step* in this process. We present a modeling framework, a novel observation based modeling technique and cross-benchmark the employed methods. Step 1 we have covered in [18] and is briefly discussed in Section 5.1. Failure avoidance schemes (Step 3) will need to be developed as Steps 1 and 2 prove to be successful. We will assess and benchmark the following nonlinear and linear modeling techniques: a) univariate linear random walk models, b) univariate linear ARMA models, c) multivariate linear regression models, d) nonlinear Radial Basis Function models and e) nonlinear Universal Basis Function models.

The paper is organized as follows: In Section 2 we review related work on software availability and empirical modeling approaches. In Section 3 we describe the modeling objectives and give a formal description of the forecasting task. In Section 4 we introduce a novel nonlinear modeling approach we call Universal Basis Functions (UBF) which is an extension of the widely discussed Radial Basis Function (RBF) approach. Before we present results in Section 6 we describe the data used, the metric and the benchmark strategies in Section 5. In Section 7 we briefly summarize our conclusions and discuss future work.

2 Related Work and Motivation

2.1 Availability of Software System

In the past a number of strategies have been developed to increase the availability of software systems. First, we can try to minimize the number of faults in a software component during the development process. After all there is a sense of that a piece of software or soft-

ware component is deterministic. The idea is: either it is fault free, then it will not fail, or it contains faults, thus circumstances that make it fail once will make it fail all the time. In actuality the software failure process arises in the context of executing successive inputs to the system. Also, the environment in which the execution takes place might have an impact. Testing the system is one way of finding malicious input patterns or flaws in their execution. However, testing only allows establishing the presence of errors. It cannot assure their absence [9]. For large systems we cannot predict with certainty all future inputs into the system during system tests and we do not know the systems faults. So of the inputs we have not investigated we do not know which of them produce a failure if executed [4]. If we knew, we could use this knowledge to fix that bug before releasing the software. Another way of testing a software system is by fault injection. This way the system's behavior may be verified if stressed in unusual ways. Some of the inherent limitations of this approach are due to heavy human involvement. Test cases have to be specified and test scenarios evaluated. For large systems the complexity of these tests may quickly become prohibitive. Also testing is strongly product based. Test cases specified for one system may not simply be transferred to another system.

Other approaches aiming to ensure specified behavior include rigorous analysis. These methods imply formal specification of the properties of a systems intended behavior and verify that the system conforms to that specification. Examples of rigorous methods include temporal logic [28] and process algebra [29]. However, formal approaches can quickly become impractical when confronted with the degree of complexity and degrees of freedom of complex computer systems. We believe that formal methods are too rigorous to scale up to enterprise systems and the unpredictable environment we find in industry does not allow the rigor we find in traditional approaches. This has been leading to a change in the way software systems are perceived and to changing concerns from bug centric - *whether* a system will work to *how well* it will work.

A realistic line of thought is to accept the fact that software systems or parts of them inevitably do fail as discussed in [25] and [33]. In fact this is our operational framework in this paper. Arguably the most followed path within this framework is to recover rapidly into a failure free state after a failure has occurred. Decreasing the reboot time has been a major research concern ([14], [35]). However, rebooting the entire system can cause nontrivial service disruption or downtime even when clusters and failover mechanisms are employed. Thus the concept of rebooting has been further refined to only restarting affected parts of the system. This procedure recovers most of the same failures as full reboots, but does so an order of magnitude faster. Most of these concepts are employed in a *reactive – post mortem* framework. Meaning after the fact that a failure occurred, measures are taken to correct it.

We often reboot our computers preemptively before mishap happens. We find the machine is behaving strangely or because parts like printing or network connections do subtly not work the way we expect them to work. It can be small deviations from normal behavior, but nonetheless we observe them and act accordingly. This can be seen as preventive maintenance, triggered by our human observation. It is our belief that by automating this process and by embedding it into a *proactive – preventive* framework we can contribute to enhanced software availability. A promising direction is to think in terms of constant adaptation to changing conditions rather than waiting for abnormal behavior to happen and then to recover. To do so, we need systems that anticipate the likely evolution of their availability or other nonfunctional properties. Based on a prediction of its future health status, the system would initiate preventive measures itself to either totally avoid the looming failure or at least to be prepared and recover more rapidly. Within this line of thought [20], [11] and [2] discuss the concept of software aging which has subsequently been further refined by a number of authors (e.g. [5], [37]). The authors argue that due to resource exhaustion, data corruption or numerical error accumulation the status of the system may degrade with time. They propose to gracefully terminate the affected application or the system while it is still in a manageable state. They call this process *rejuvenation*. The problem then becomes determining the optimal time for rejuvenating. The authors propose, among other techniques, stochastic measurement-based models. This line of thought has been reflected in the specific research area of systems which manage themselves and *anticipate* the likely evolution of their dynamics. These systems are also called *self-** systems, where the star is a place holder for repairing, configuration, healing, optimization and protection to name a few. Although some *self-** approaches like self-repairing date back to the late 1960's [1] more recently *autonomic computing* has been used synonymously [19] and *self-** systems have been investigated in a broader context (e.g. [8]).

2.2 Probabilistic Modeling Approaches

How can we build the *anticipating* part of *self-** software systems? It is our belief that the uncertainty about a systems availability, mainly introduced by its complexity, can be captured by probabilistic representation of the system. To achieve this goal we propose to apply a data-driven methodology based on a novel nonlinear modeling technique

There is a well-established theory and a large body of tools which let us characterize any given system which can be described by a set of *linear* equa-

tions. It is being used in virtually every scientific discipline. Where the assumption of linearity breaks down there is a lack of a uniform theory. However, despite this lack it is remarkable how much theoretical insight we can gain on the dynamics of a nonlinear system. For example, the reconstruction theorem [36], also known as the embedding theorem, provides the theoretical means to model highly nonlinear behavior of arbitrary physical systems with hidden dynamics. It means that for almost any scalar function, for example, observations we take from a software system, we can answer a wide range of questions about the dynamics of the original system by examining the dynamics in a space defined by delayed values of this function. We can compute dynamical invariants, and we can even make predictions by interpolating in the so-called delay embedding space[1]. Furthermore, the reconstruction theorem detects low-dimensional structure in a high-dimensional data space, modeling the *effective* degrees of freedom of a system rather than *all* degrees of freedom. This has important implications for our approach: in the case of software systems it implies that it should be sufficient to model the larger picture of effective internal states and output signals, rather than representing every single module. However, the reconstruction theorem has its drawbacks. Models can easily become unstable for large embedding dimensions. Also the availability of large amounts of highly precise data is assumed [26]. This is difficult to achieve in the presence of noise and limited data access in practically any real-world system. The reconstruction theorem has influenced a number of methods aiming at circumventing these challenges. These methods automatically detect the functional relationship of data in the presence of noise, limited data accessibility and possibly nonlinear dependencies among observations. Methods which have been shown to be able to handle these scenarios include Radial Basis Functions [24], Multi-Layer Perceptrons [31], Projection Pursuit [21], Hidden Markov Models [30] and Support Vector Machines [6]. These methods are known under a variety of names such as learning networks or nonlinear, stochastic and probabilistic regression. In this paper, we will synonymously use the latter terms as well as machine learning and empirical models where applicable. One method which has been extensively utilized and studied in conditional density estimation and function approximation is Radial Basis Functions (RBF). This method allows data-driven, nonlinear modeling. RBF are computationally less expensive than Multi-Layer Perceptrons, Projection Pursuit or Hidden Markov Models and can be translated into Multi-Layer Perceptrons [13]. Also Radial Basis Functions are universal approximators, which means they can approximate any continuous function given enough degrees of freedom [27]. Although these methods were originally developed in different contexts for seemingly different purposes, they can be viewed as probabilistic approaches to the problem of nonlinear regression.

Concluding we can say that rigorous analytical study is imperative for systems needing absolute guarantees such as nuclear power plants, airplanes or mars robots. At the same time it is impractical for generic enterprise computing. These less critical systems could significantly benefit from empirical modeling approaches based on observing the system at runtime and extracting information about its behavior. This is the approach we will focus on in this paper.

3 Formal Description of Forecasting Task

The modeling and forecasting task in our scenario is straightforward. Given a set of labeled observations $\mathbf{x} = \{x \mid x = \langle f_1,...,f_n \rangle\}$ of our target system we compute a classifier *Cl* that predicts from the observed features $\langle f_1,...,f_{n-1} \rangle$ the target class label f_n which is either "*failure*" or "*no failure*". Each element $f \in \mathbf{x}$ is a vector of features where we denote $\langle f_1,...,f_{n-1} \rangle$ as the input features and f_n as the target class label. f_n=1 denotes "*no failure*", f_n=0 denotes "*failure*". Given a new pattern or a previously unseen observation $\mathbf{x}_{new}$ with an unknown class label we obtain $f_n = Cl(\mathbf{x}_{new})$. We say a prediction at time t_1 is correct if the target event occurs at least once within the *prediction period* Δt_p. The prediction period occurs some time after the prediction is made, we call this the *lead time* Δt_l. This *lead time* is necessary for a prediction to be of any use.

Figure 1: Time segments for modeling and prediction

The prediction period defines how far the prediction extends into the future. The value of the *lead time* Δt_l critically depends on the problem domain, e.g. how long it takes to restart a component or to initiate a fail over sequence. The values are defined as follows:

$$\textit{lead time} \qquad \Delta t_l = t_2 - t_1 \qquad (1)$$

$$\textit{prediction period} \qquad \Delta t_p = t_3 - t_2 \qquad (2)$$

The *embedding dimension* Δt_e which is denoted as

$$\textit{embedding dimension} \qquad \Delta t_e = t_1 - t_0 \qquad (3)$$

[1] Number of axes of a return map sufficient to describe the properties of the corresponding phase space. The return map is a plot of a time series as a function of the current and of the previous values.

specifies how far the labeled observations $f \in \mathbf{x}$ extend into the past.

4 Nonlinear Empirical Modeling

In this paper we focus on the application of linear as well as nonlinear empirical modeling techniques to real measurements taken from a telecommunication system. Linear regression procedures have been described in great detail over the past decades. So in this section we revisit the *nonlinear* data driven modeling technique Radial Basis Functions (RBF) and will then build on this framework to derive and introduce the novel and more general Universal Basis Function (UBF) approach.

Techniques for classification, function approximation and nonlinear regression have proliferated over the past two decades. Radial Basis Function (RBF) networks are one of the primary tools for interpolating multidimensional scattered data and are arguably one of the most popular methods for nonlinear regression. This lends basically to its proximity to linear modeling techniques which are widely used and well-understood and also to its ability to handle arbitrarily scattered data and to easily generalize to several space dimensions. RBF are also well motivated in the framework of regularization theory [27]. They have been applied to a number of seemingly disparate domains such as financial data, state space reconstruction in physics and function approximation [38] as well as classification of medical and biological data sets [34].

However, the generalization quality of RBF models strongly depends on issues in their architecture, learning algorithms, initialization heuristics and regularization techniques. RBF are typically used with a priori fixed kernel functions and little attention has been given to the effect of mixture kernels on model quality and efficiency of parameter optimization. In particular, it has been shown that some data distributions, such as heavy tailed ones, cannot be well approximated by Gaussian functions [23] and that depending on the specific problem at hand, adapting the transfer functions according to the underlying data can improve model quality significantly [15].

In the next Sections we will introduce a novel concept for domain specific mixture functions in the RBF framework which overcomes some of these limitations. We call this novel modeling technique Universal Basis Function (UBF) networks to reflect their proximity to Radial Basis Function.

4.1 Radial Basis Functions

Suppose we obtain a set of data

$$\mathbf{m} = \left\{ \left(\mathbf{x}_j, y_j \right) \in \Re^d \times \Re^1 \right\} \tag{4}$$

with $j \in [1, N]$ by random sampling from a function f (see Equation 5) in the presence of noise. Usually, we are interested in recovering f from our sampled data $\mathbf{m}$. Note that our sampled data $\mathbf{m}$ consists of observation vectors $\mathbf{x}_j$, in linear statistics also called regressors, and the corresponding target value y_j. Where j is the index of the current observation, N the total number of observations and d is the dimension of the input vector.

$$y_j = f\left(\mathbf{x}_j \right) \tag{5}$$

This problem of finding the function f is ill-posed since it has an infinite number of solutions [16]. To reconstruct any particular solution we clearly need some a priori knowledge about the function in question. The most common assumption in these cases is that the function is smooth. Smoothness in this context can be thought of as two similar inputs giving rise to similar outputs, i.e. the principle of strong causality must hold which says that a small change in the inputs results in a small change in the output. The underlying principle is regularization theory [27]. In this approach a functional H[f] is formulated as in Equation (6). The objective is then to find a function f which is close to our sampled data and is smooth according to a chosen smoothness criterion [13]. Such a function f would minimize the following functional

$$\mathrm{H}[f] = \sum_{j=1}^{N} \left(f\left(\mathbf{x}_j \right) - y_j \right)^2 + p\left(\mathbf{x}_j \right) \tag{6}$$

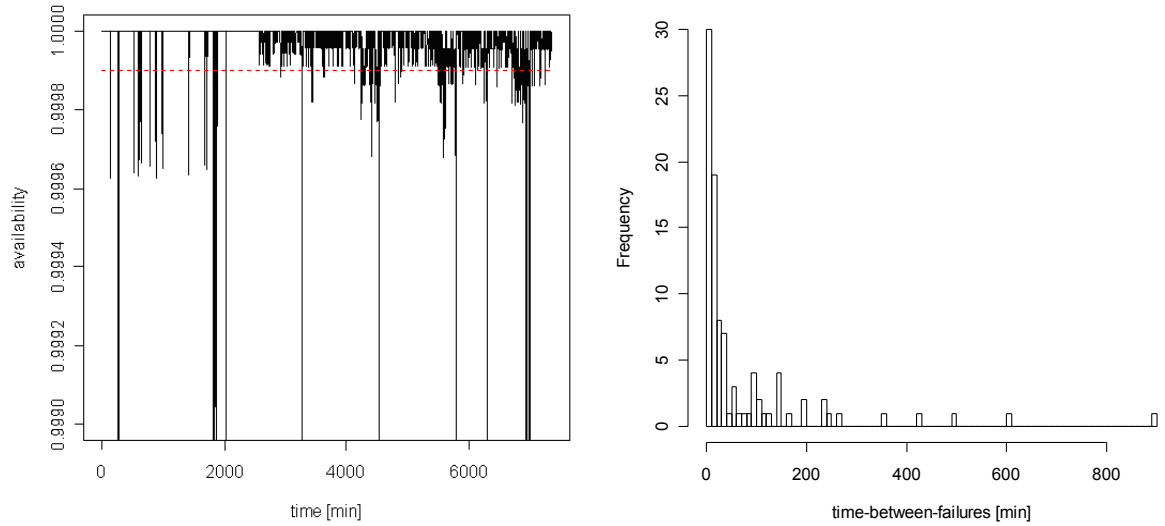
The first term $\sum_{j=1}^{N} \left(f\left(\mathbf{x}_j \right) - y_j \right)^2$ enforces closeness to our data, the second polynomial term $p(\mathbf{x}_j)$ smoothness. The polynomial is typically taken to be just a linear or constant term [22], the solution for f is then given by [13] as

$$f(\mathbf{x}) = \sum_{j=1}^{N} c_j G\left(\mathbf{W}\mathbf{x} - \mathbf{W}\mathbf{x}_j \right) \tag{7}$$

where c's are coefficients, $\mathbf{W}$ is a $d \times d$ (see Equation 4) weight matrix which rotates and resizes the input space to reflect possible linear combinations of original input variables and G is some nonlinear transformation function [27]. In this scheme we have the weight matrix $\mathbf{W}$ and the coefficients c as free parameters. This can result in an approximation task where more parameters have to be estimated than there are data points. To escape from this dilemma [13] describe the scheme of *Generalized Regularization Networks* (GRN) in which centers $\mathbf{t}$ coincide with the data points and the weight matrix $\mathbf{W}$ is fixed to be the identity $\boldsymbol{I}$. This results in

$$f(\mathbf{x}) = \sum_{j=1}^{N} c_j G\left(\mathbf{x} - \mathbf{t}_j \right) \tag{8}$$

The one-to-one correspondence between the observations $\mathbf{x}$ and the transfer function $G(\mathbf{x};\mathbf{x}_j)$ of this type of



(a) (b)

Figure 2: In (a) we report the target data which is the system's interval call availability (A). The dotted line indicates a 99.99% availability limit. Any drop below that threshold is defined as a failure. Our objective is to model and predict the timely appearance of these failures. The system's interval call availability is reported in consecutive five minute intervals and is calculated as the number of successful calls over the total number of calls in this interval. In (b) we report the distribution of time between failures (TBF).

network (see Equation 7) can make it prohibitively expensive to compute for large N. Also the computation of the linear coefficients c in Equation (8) requires the inversion of an N-by-N matrix which grows roughly as $O(N^3)$. This has lead to so called *Hyperbf* where the centers are considered to be free parameters as well. The number of centers n (Equation 9) is then chosen so that it is much smaller than the number of available observations N such that $n << N$. The linear coefficients c_j can straightforwardly be calculated for example by singular value decomposition. This scheme has the advantage of being computationally less expensive while retaining the form of the regularization solution depicted in Equation (6). The solution to Equation (8) is given by [27] as

$$f(\mathbf{x}) = \sum_{i=1}^{n} c_i G(\|\mathbf{x} - \mathbf{t}_i\|) \tag{9}$$

where $\|\bullet\|$ denotes a geometric distance measure, such as the Euclidean. In fact this is the solution we will focus on when deriving generalized Universal Basis Functions.

4.2 Generalized Universal Basis Functions

In this Section we introduce a more general concept of basis functions by replacing G with a flexible function (i.e. not necessarily Gaussian) to adapt to specifics of the data space. Let

$$\mathbf{r} = \|\mathbf{x} - \mathbf{t}_i\| \tag{10}$$

then $G(\|\mathbf{x} - \mathbf{t}_i\|)$ can be rewritten as $G(\mathbf{r})$. We focus on mixtures of activation functions such as Gaussian, sigmoid and multiquadratics. We would like to crossover smoothly from one activation function to the other covering all states in the continuum between the two extremes. For convenience we define ω which regulates the dominance of one activation function Φ_1 over the other Φ_2.

$$G(\mathbf{r}, \boldsymbol{\omega}) = \boldsymbol{\omega}\Phi_1(\mathbf{r}) + (1 - \boldsymbol{\omega})\Phi_2(\mathbf{r}) \tag{11}$$

Note that ω can be any real value. Our desired value range is [0,1]. Thus, we introduce a standard nonlinear mapping

$$\omega' = \frac{1}{2}(\tanh(\omega) + 1) \tag{12}$$

The value of ω' now controls the behavior of the transfer function in the fringe regions and maps it to the unit interval $G : [-\infty, \infty] \rightarrow [0,1]$. This way we get a smooth transfer from one activation function to another.

$$G(\mathbf{r}, \boldsymbol{\omega}') = \boldsymbol{\omega}'\Phi_1(\mathbf{r}) + (1 - \boldsymbol{\omega}')\Phi_2(\mathbf{r}) \tag{13}$$

with Φ being any of the following functions Φ':

$$\Phi'_1 = e^{-r^2/2\sigma^2} \tag{14}$$

$$\Phi'_2 = \tanh(r/\sigma^2) \tag{15}$$

$$\Phi'_3 = \sqrt{r^2 + \sigma^2} \tag{16}$$

and σ is a function specific constant which is set following a heuristic which will be detailed in Section *4.3*. We could try many more transfer functions, however, as detailed in the previous sections, the listed transfer function Equation(s) (14), (15) and (16) are pervasively used in literature and our believe is we should start with some known transfer function before we apply more exotic ones. The degree of shape variation for Gaussian, respectively the steepness of the sigmoid, can be adjusted by adapting the functions width σ. Inserting equation (10) into equation (13) and replacing *G* in equation (9) by equation (13) yields

$$f(\mathbf{x}) = \sum_{i=1}^{n} c_i \, \mathrm{G}\left(\left\|\mathbf{x} - \mathbf{t}_i\right\| \boldsymbol{\omega}_i'\right) \tag{17}$$

If we do have knowledge about the data distribution we should give a bias towards a specific activation function by initializing the slider value ω' accordingly. However, as frequently is the case knowledge about the data distribution is either difficult to obtain or not available at all. In this case we start with an educated guess. If no further information is available to justify a bias towards a particular activation function we start with a Gaussian RBF approach and set $\omega' = 1$. Subsequently we optimize ω' by stepwise decreasing its value.

4.3 Parametrization

Assume we are given the centers $\mathbf{t}_i$ in Equation (9) then we can find the coefficients c_i. How do we choose the centers though? There are a number of ways such as random sampling from the data set, by least squares or by clustering, e.g. by *k*-means [22]. When applying random sampling the sample should reflect the distribution of the data. There are only a few theoretical results available, however, [27] proved that a solutions exists using this technique. The main problems are a) the optimal number of the centers and b) the placement of these centers. There is no general answer known to these problems and cross validation techniques are considered a reasonable choice [10]. When using least squares or moving centers the problem becomes non-convex and multiple local minima are to be expected [27]. This technique is in principle very powerful but can become computationally expensive for large *N*. Clustering is commonly used, even though no theoretical results exist that prove feasible solutions. However, sensibly applied it is a good starting point [22]. In fact clustering is the method we apply in this paper. The solution of Equation (17) with respect to the vector of coefficients **c** yields ([27])

$$\mathbf{c} = \mathbf{G}^{+}\mathbf{y} \tag{18}$$

with

$$\mathbf{y} = \left[y_1,\ldots,y_N\right]^T \tag{19}$$

the vector of coefficients

$$\mathbf{c} = \left[c_1,\ldots,c_n\right]^T \tag{20}$$

and

$$\mathbf{G} = \begin{bmatrix} G(\mathbf{x}_1;\mathbf{t}_1) & \ldots & G(\mathbf{x}_1;\mathbf{t}_n) & x_1^1 & \ldots & x_1^d & 1 \\ \vdots & & \vdots & \vdots & & \vdots & \vdots \\ G(\mathbf{x}_N;\mathbf{t}_1) & \ldots & G(\mathbf{x}_N;\mathbf{t}_n) & x_N^1 & \ldots & x_N^d & 1 \end{bmatrix} \tag{21}$$

where *d* is the dimension of the input vector as defined above, x_N^d denotes the *d*-th component of the *N*-th observation, $\mathbf{x}_N$ is the *N*-th observation vector and $\mathbf{t}_n$ is the *n*-th kernel vector. The middle part of the matrix **G** including x_1^1 to x_N^d denotes optional direct linear connections between the input and the output layer, while the last column denotes the constant bias term. It is interesting to note that if we leave out the nonlinear transformation $G(\mathbf{x}_1;t_1)$ to $G(\mathbf{x}_N;t_n)$ but keep the direct linear connections between the input and the output layer we obtain a classical linear equation system. To obtain all variables to solve Equation (15) we need $\mathbf{G}^+$ which is called the pseudo-inverse[2] of the matrix **G** [27], thus

$$\mathbf{G}^{+} = \left(\mathbf{G}^T\mathbf{G} + \lambda\mathbf{I}\right)^{-1}\mathbf{G}^T \tag{22}$$

where ^{T} in the exponent denotes the transposed matrix and $^{-1}$ in the exponent denotes the inverse. However, inverting $\mathbf{G}^T\mathbf{G}$ can be subject to numerical problems because of the likelihood in our high dimensional nonlinear setting of the matrix being ill-conditioned. [7] propose a solution to this problem by preconditioning the **G** matrix thru the use of an iterated Laplacian operator. This also makes the diagonal of the square matrix $\mathbf{G}^T\mathbf{G}$ dominant by multiplying the identity **I** by $\lambda > 0$ and thus incorporates a smoothness assumption [27]. This is the solution we adopt. Note that *real* problems are usually over determined. This means we have more data points available than free parameters. The matrix **G** is therefore not square ($N > n$) and consequently no unique inverse exists. For this reason we use the pseudo inverse approach as shown in Equation (17) where $\mathbf{G}^+$ is the pseudo inverse matrix of **G**. $\mathbf{G}^T\mathbf{G}$ becomes a square matrix in its own right with a unique inverse of its own. Substituting Equation(s) (15) and (8) into Equation (14) we get a vector of coefficients **c** which in

[2] $\mathbf{A}^+=(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ is called the pseudo inverse matrix of **A**. Here, $\mathbf{A}^+\mathbf{A}=\mathbf{I}$ holds, where I is an identity matrix.

	AUC out-of-sample Confidence levels		
Model	**Median**	**Lower**	**Upper**
Random Walk (TBF)	0,5000000	0,5000000	0,5000000
ARMA(1,0) (TBF)	0,5000000	0,5000000	0,5000000
Multivariate linear	0,8070540	0,8017753	0,8123326
RBF nonlinear only	0,6230982	0,6226479	0,6235485
UBF nonlinear only	0,7375519	0,7264591	0,7486447
RBF	0,8257261	0,8122181	0,8392342
UBF	0,9024896	0,8881837	0,9167955

Table 1: In this table we present the median out-of-sample AUC performance for failure recognition. The random walk and ARMA(1,0) models are based on time-between-failure (TBF) data. We also report the AUC performance of multivariate linear, RBF and UBF models. For RBF and UBF models we also report AUC performance for models built only with their respective nonlinear part.

addition to the location of the centers **t** and the value of ω' completes the specification of the UBF model.

5 Experiment: An industrial Telecommunication System

The system we consider is an industrial telecommunication platform which handles mobile originated calls (MOC) and value adding services such as short message services (SMS) and multimedia messaging services (MMS). It operates with the Global System for Mobile Communication (GSM) and General Packet Radio Service (GPRS). The systems architecture follows strict design guidelines considering reliability, fault tolerance, performance, efficiency and compatibility issues. We focus on one specific system which, at the time we took our measurements, consisted of somewhat more than 1.6 million lines of code, approximately 200 components[3] and 2000 classes[4]. It is designed to be operated distributed over two to eight nodes for performance and fault tolerance reasons. The system we model consists of two nodes operating in GPRS mode. Both nodes form one cluster. We call them *node1* and *node2*. The system is designed to be operated nonstop. We will focus on modeling and predicting system events (i.e. calls) which take longer time to be processed than some guaranteed threshold value. We call these events *failures* or *target events*.

To a) parameterize, b) validate and c) test the generalization capabilities of our models, we split the data into three equally large segments. We use the first and second segment to parameterize and cross-validate the models as described in Section *4.3. Parametrization*, the third segment of data, which is *not* used in the model building process, we use to derive the models *generalization* or *out-of-sample* performance. In this paper we report the *out-of-sample* performance of the models.

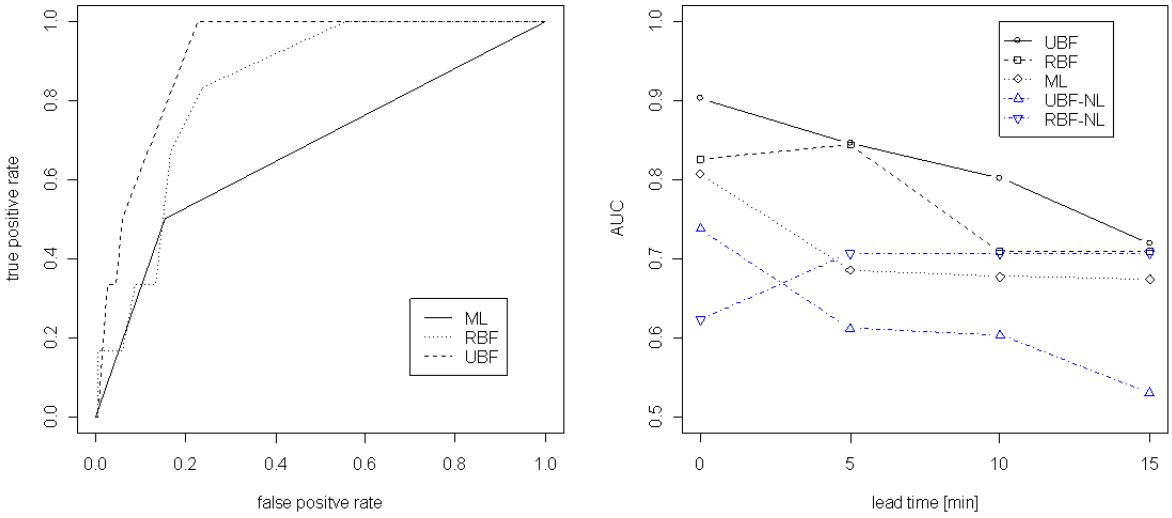
5.1 Data, Notation and Variable Selection

The data we use to build and verify our models consists of a) equidistant time-triggered continuous variables and b) time stamped event-driven log file entries. To record the numeric values of system variables we use the *system activity reporter* (SAR) utility running under the UNIX operating system. It samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. We gather numeric values of 46 system variables once per minute and per node. This yields 92 variables in a time series describing the evolution of the internal states of the system. In a 24-hour period we collect a total of 132.480 readings (92 variables times 24 hours times 60 minutes per hour with one reading per variable and minute). In total we collect roughly 1.3 million system variable observations.

It is difficult if not impossible to evaluate the influence of each variable on the predictive quality of a model due to combinatorial explosion and potentially nonlinear correlations. In fact including too few, too many or unfavorable variable combinations into the model building process can degrade model performance significantly ([12],[17]). This problem is known under a variety of names such as *variable selection, dimension reduction* or *feature detection*. The problem is finding the smallest subset of input variables which are necessary or suffice to perform a modeling task. In most cases ad hoc theories or gut feeling is employed to derive a plausible set of explaining variables. To eliminate the latent ambiguity of this approach, techniques are needed to automatically find the most predictive and meaningful variables to observe. In [18] we have benchmarked four techniques for variable selection for the same data set we use in this experiment. We have benchmarked the well documented *forward selection* and *backward elimination* procedures as well as a probabilistic wrapper approach and a set of vari-

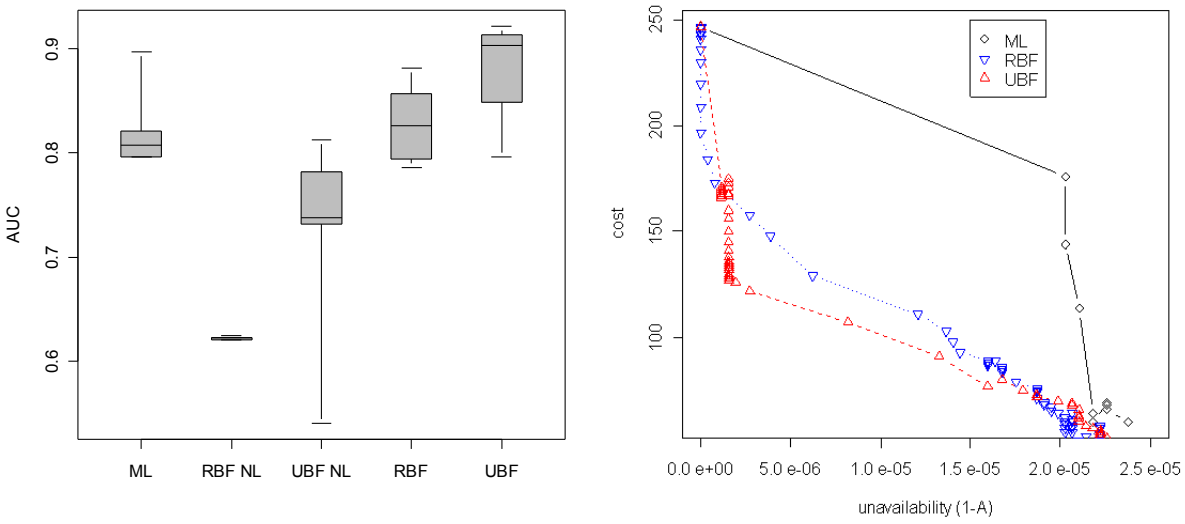
[3] a system element offering a predefined service and able to communicate with other components

[4] classes are used to group related variables and functions



(a) (b)

Figure 3: Failure recognition results are shown in (a). The UBF model (AUC=0.9024) outperforms the RBF (AUC=0.8257) and ML (AUC=0.807) approach. AUC characteristics plotted over a number of lead times for each modeling technique are shown in (b). All values are t-tested against each other at a 95% confidence level.



(a) (b)

Figure 4: Nonlinear UBF models outperform linear as well as nonlinear RBF benchmark methods. Statistical significance we established using *t*-testing. In (a) we present box-whisker charts showing the median and the lower and upper error quartiles of out-of-sample performance across five modeling approaches for multivariate linear (ML), nonlinear only RBF and RBF, as well as RBF and UBF with linear parts included. Figure (b) shows cost vs. unavailability. We display cost-unavailability ratios by single characters and connect the points for clarity. For cost formula and further explanation see text.

ables hand-picked by system specialists. We identified two out of the 96 variables as being most predictive. These are 1) *sema/s* (number of semaphore operations per second) and 2) *alloc* (amount of memory that the kernel memory allocator has allocated). To avoid potential biasing the variables were identified by segmenting the data the same way as described earlier in this Section. The variable selection process only made use of the first two segments of the data, all results reported in this paper are then calculated for the third *out-of-sample* set which had been set aside for generalization purposes. It is interesting to note that the specialist selected set of variables was outperformed by all other algorithmic procedures. This seems to confirm our view that specialist based variable selection can be ad-hoc and sub-optimal.

In this paper we first focus on observations of these two variables and will then include log file data in an attempt to improve our models. Log file data has been identified as a potential source with predictive power with respect to impending failures ([3], [32]). We gather log file entries in the same time period as we gather SAR data. The log file entries are written by the operating system and the application into text files. The entries are written event-driven and are partially non-numeric. The number of entries per time segment ranges from several hundred entries per second to a few entries per minute. Each log file entry consists of a number of labels which contain information of the source which has written the entry, date and time information as well as the severity and nature of the entry. The log file entries are classified into 195 predefined classes according to their area of relevance (e.g. database, network). Each log file event is labeled by its class name. Considering our two nodes we will have to handle 390 class variables (195 classes at each node). In total we gather roughly four million log file entries.

5.2 Failure Definition

For our purposes we introduce *interval call availability* $A_c(\Delta t)$, which is the probability that calls within a specific time interval Δt will be handled by the system within a given deadline. *Interval call availability* is calculated as the number of calls completed $n_{completed}$ over the total number of calls n_{total} in the interval Δt. $A_c(\Delta t)$ can also be written as one minus the number of failed calls n_{failed} over the total number of calls in this interval. In our scenario we derive $A_c(\Delta t)$ as

$$A_c(\Delta t) = 1 - \frac{n_{failed}}{n_{total}} \tag{23}$$

The values of n_{failed} and n_{total} are given to us in time stamped log files. The interval Δt is given as five minutes. A failed call is defined as a call which is not handled within a given amount of time. A *failure* is defined as 0.01% or more calls not being processed within the given deadline. This corresponds to $A_c(\Delta t)$ dropping below 99.99%. In this paper we use the term *availability* synonymously to *interval call availability*. For simplicity we also write $A = A_c(\Delta t)$. See Figure 2 (a) for a plot.

5.3 Metric

When making predictions about the system's future state we must take into account true positive (TP), false positive (FP), true negative (TN) and false negative (FN) classifications. Focusing on TP alone may substantially bias a model. A metric which takes all four prediction outcomes into a account is true positive rate (*Tprate*) and false positive rate (*FPrate*).

$$TPrate = \frac{TP}{TP + FN} \tag{24}$$

$$FPrate = \frac{FP}{FP + TN} \tag{25}$$

To express the interdependence of *FPrate* and *TPrate* frequently the *Receiver Operating Characteristic (ROC)* is calculated. The ROC curve is used for diagnosis by assessing the ability of our model to discriminate between failures and nonfailures. If the model discriminates perfectly, the ROC curve passes through the coordinates (0,1) and the *Area Under the Curve (AUC)* is one. If the model has no discriminating ability, we see a straight line from (0,0) to (1,1). Each point on the ROC curve provides the false positive rate (*FPrate*) and true positive rate (*TPrate*) associated with a threshold in the probability scale which allows classification of each observation. See Figure 3 (a) for details. The ROC curve shows the trade-off between *FPrate* and *TPrate.* Any increase in *TPrate* will be accompanied by an increase in *FPrate*. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate is our model. In fact we use AUC intensely to analyze and compare our models.

5.4 Models and Benchmark Strategies

How well do traditional linear modeling techniques perform and can we improve the performance of these models by adding nonlinearities? We investigate this problem in two steps:

1. time-driven approach: first we build bottom-of-the-line random walk and linear autoregressive moving average (ARMA) models. Prediction in this case is dependent on *time between failures* (tbf) only.
2. State-dependent approach: We then extend our approach to multivariate linear models, subsequently introduce nonlinearities to our models and benchmark them against the linear models and against each other.

Exemplary we report the UBF's parametrization. In the UBF model the number of centers n is five. This parameter choice was made based on n-fold cross-validation. The system's availability based on our UBF model thus is defined as

$$\mathbf{A} = G\left(\frac{-\|\mathbf{x}-\mathbf{t}_i\|^2}{0.06324555^2}\right)\mathbf{c} \tag{26}$$

The kernel vectors **t** which feed into **G** and the coefficient vector **c** are parameterized as follows:

$$\mathbf{t}_1 = [0.01416387, 0.04101540, 0.06209718, 0.10428004, 0.20404461]$$
$$\mathbf{t}_2 = [0.5536791, 0.5622841, 0.5658562, 0.6298298, 0.6350998]^T \tag{27}$$

$$\mathbf{c} = \begin{bmatrix} 0.035458364, 0.022254254, 0.020798340, 0.047735738, \ldots \\ \ldots 0.097423457, 0.097427984, 0.003668041, 0.057793757 \end{bmatrix} \tag{28}$$

The transfer function $G(\bullet)$ is

$$G(\bullet) = \omega' \Phi_1(\bullet) + (1-\omega')\Phi_2(\bullet) \tag{29}$$

The UBF type is Gauss / sigmoid with Φ_1 and Φ_2 being the Gaussian and sigmoid function respectively. The UBF slider $\omega' = \tanh(0.5\omega)$ with $\omega = -0.8$.

6 Results

In this section we will first give the failure recognition results at $\Delta t_l = 0$, we will then present results for lead times from five to 15 minutes. We will continue with including log files into the model building process. Finally we will propose and evaluate a cost metric for cost-benefit analysis of our models.

6.1 Failure Recognition Results

In Table 1 we present out-of-sample AUC performance for failure recognition. These results are the median out-of-sample values of 20 models we have built on bootstrapped datasets [10]. In Figure 3 (a) we present the ROC chart for *out-of-sample* failure recognition data with $\Delta t_l = 0$. We compared the performance of the UBF modeling approach with that of traditional empirical modeling techniques such as ARMA, random walk, multivariate linear (ML) models and nonlinear Radial Basis Functions (RBF). The ARMA and random walk approach did not recognize any failure. This is an expected result because these univariate models are based on time between failures and do not consider other variables. The distribution of the time between failures is heavy tailed with irregular spikes. Also, there is practically no autocorrelation which could be exploited by ARMA models, see Figure 5.

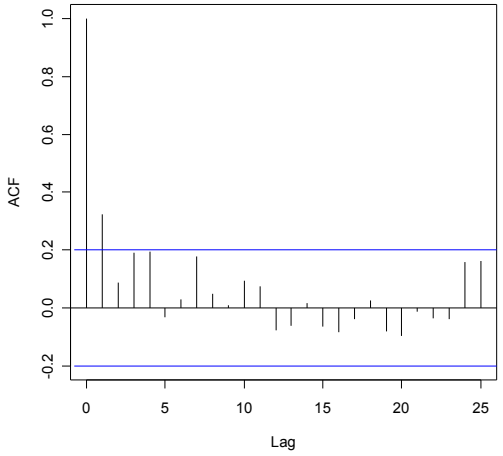


Figure 5: Time-between-failures (TBF) autocorrelation function (ACF). Horizontal lines indicate approximate 95% confidence intervals.

In the next step we resort to multivariate approaches. We calculated error bounds and statistical significance of all models. UBF clearly and significantly outperformed ML and RBF (compare Table 1). We also investigated the effect of focusing on nonlinear modeling only. For that purpose we cut all linear parts in UBF and RBF. The result clearly indicates superior model quality of UBF over RBF. However, the quality of nonlinear only models does not reach that of linear models or that of mixture models which include nonlinear as well as linear parts. We therefore conclude that the data of our telecommunication system has a high fraction of linear correlation. However, adding nonlinearities to the models significantly improves overall failure recognition performance.

The UBF model outperforms all other modeling techniques with an AUC of 0.9025 on the generalization data set (see Table 1). It is followed by the RBF approach with an AUC value of 0.8257 and the ML model with an AUC of 0.8071 on the generalization data set. It is interesting to note that RBF and UBF models which we built using only their nonlinear part were worse than the ML model. These modeling approaches yield an AUC of 0.6230 and 0.7376 respectively.

6.2 Failure Prediction Results

In the previous sections we have looked into the capability of our models to recognize failures at the time they occur. We now turn to models with a lead time greater than zero for failure prediction. We use the same input data as in the previous sections. Now, we present our target value with a time lag of $\Delta t_l \in \{5,10,15\}$ minutes. In Figure 3 (b) we present AUC

characteristics plotted over a number of lead times for each modeling technique. The reported values are medians of their respective distribution. UBF models outperformed all other approaches significantly. Interestingly, the UBF advantage narrows for models predicting further into the future, nonetheless the difference in prediction quality is statistically significant. It seems that model performance of UBF, RBF and ML converges to error bounds within a small interval for predicting failures 15 or more minutes ahead (compare Figure 3 (b)), indicating that nonlinear data correlations play less a role the further we look ahead. The presented results clearly indicate that empirical models for failure prediction can successfully be constructed based on historic observations of the systems behavior.

6.3 Including Log File Data

We built predictive models using additional log file data. We used the same target failure data as in the previous sections with $\Delta t_l = 5$ minutes. We report results in Figure 6 and Table 2. The models did poorly compared to SAR-fed models with corresponding lead times, compare Table 1 and Figure 4 (a). The UBF model yields a median AUC of 0.5958 on the generalization data, whereas the UBF model fed with SAR data yields an AUC of 0.85. In Figure 6 we present median and quartiles of the respective models for out-of-sample AUC in a box-whisker plot. In fact our standard *t*-test indicates no significant difference in distributions among all three models.

	AUC out-of-sample performance		
Model	***Median***	***Lower***	***Upper***
ML	0,61542	0,60126	0,62957
RBF	0,57167	0,53086	0,61247
UBF	0,59583	0,57717	0,61449

Table 2: Median, lower and upper bounds of out-of-sample AUC for ML, RBF and UBF models based on log file data.

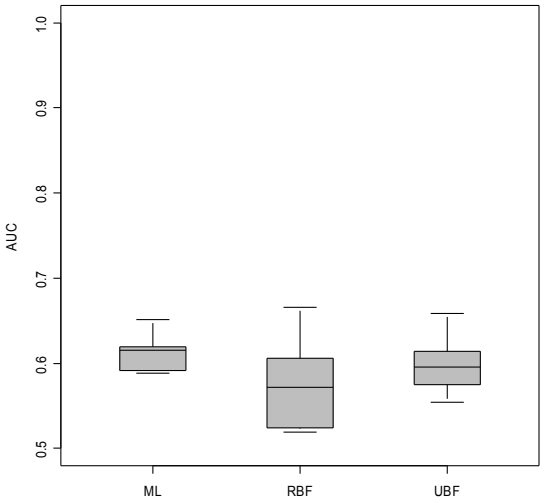


Figure 6: Box-Whisker plots for out-of-sample AUC values of linear (ML), RBF and UBF models. Models built based on log file data.

6.4 Cost-Benefit Analysis

Cost-benefit analysis is the nontrivial part of deciding what level of involvement is sensible to reach a certain level of system availability. The methods described in this paper involve cost in terms of computational and human resources and on the other hand promise an improvement in availability. In some ultra-reliable systems the cost for increasing the systems availability may be taken to extremes. In our case we would like to develop an understanding of what cost is associated with what level of availability improvement. We would like to compare the status quo situation, i.e. known cost and availability levels with options offered by our prediction method.

Benefit in our case can be defined as the increase in system availability A or decrease in unavailability (1-A). Cost is more difficult to define because it depends on variables which we can only estimate upfront. Matters are further complicated by the fact that we are focused on modeling and predicting failures, which can be used for preventive maintenance actions, which in turn may increase availability. However, the cost associated with these preventive measure covers a broad range we can hardly assess without empirically measuring the effects. Thus, we approach this challenge by depicting cost as a function of the models performance as we can assess it, namely by its contingency matrix. If, for example, missing a failure (*FN*) costs ten times the value of making a false positive (*FP*) statement we can reflect this by introducing the respective ratio in our cost function. Thus for each preventive

maintenance method triggered by our prediction algorithm we should be able to parameterize the cost function according to its specifics. Once we have clamped the parameters of the cost function we can find the optimal operating point of our model with respect to its cost/benefit ratio by riding along the AUC curve. One way of approaching this topic is to let the cost function be the sum of costs for each entry in the contingency matrix weighted by the number of its occurrences.

$$cost = n_{TP}C_{TP} + n_{TN}C_{TN} + n_{FP}C_{FP} + n_{FN}C_{FN} \quad (30)$$

The number n_i with $i \in \{TP,TN,FP,FN\}$ for each cell of the contingency matrix we derive by counting. The proper parametrization of C obviously strongly depends on the strategy we employ to avoid a predicted failure. In Figure 4 (b) we present the cost curve for $C=\{1,0,1,10\}$. This cost vector implies that a missed failure (*FN*) is ten times as cost intensive as a falsely predicted failure (*FP*). This is arguably an arbitrary value and should be understood as an example only for demonstration purposes. We caution the reader to apply this metric blindly. In Figure 4 (b) we depict the cost / unavailability ratio for ML, RBF and UBF models for out-of-sample generalization data. The UBF model shows a more favorable cost / unavailability ratio which is an expected result given that UBF predictions clearly outperform other modeling techniques. The ML model yields a rather inflexible frame for choosing a particular cost / unavailability point by only offering few operational points (the circles in Figure 4).

7 Conclusions and Future Work

In this paper, we have firstly applied linear as well as nonlinear data driven techniques to model and then forecast call availability and failures of an industrial telecommunication system. This is in contrast to commonly used theory or first principle driven modeling. We predicted call availability up to 15 minutes ahead of time and recognized failures as they occurred. Secondly, we presented, applied and assessed a novel nonlinear empirical modeling technique we call Universal Basis Functions (UBF). We have cross-benchmarked five empirical modeling techniques, these include the linear ARMA (autoregressive moving average), multivariate and random walk techniques, as well as the nonlinear Radial Basis Functions (RBF) and Universal Basis Functions (UBF).

From the telecommunication system we have gathered real data during runtime from two different sources which are a) equidistant numerical data as time series and b) time stamped textual log file data. Overall results clearly show that empirical multivariate modeling techniques can be effectively used for modeling and prediction in our telecommunication system. We did find a fair amount of nonlinearity in the system's data. In fact nonlinear modeling techniques such as UBF and RBF significantly outperformed linear techniques in failure recognition and prediction. Statistical significance was established based on *t*-testing. The nonlinear UBF approach significantly outperformed RBF.

Contrary to conventional wisdom including log file data did *not* improve model quality, on the contrary, model quality deteriorated when we included log file data. We attribute this effect to a particularly high degree of noise in the data and most probably limited information contained in the log files. Even though this result may not be generalized at this point it may be a promising starting point for future research to get insights into better structuring log files and to evaluate what type of information should be included into log files.

We also proposed a parameterizable cost function for cost-benefit analysis which can reflect the cost associated with a particular prevention step after a failure has been predicted. Future work will include an in-depth analysis on closing the control loop, which is integrating prediction techniques with particular preventive maintenance methods such as failover, checkpointing or rejuvenation to name a few.

References

[1]: Avizienis A. *An Experimental Self-Repairing Computer*. NASA-TR-32-1356, Jet Propulsion Laboratory, Pasadena, CA , p. E30, 1968

[2]: Pfening A., Garg S., Puliafito A., Telek M., Trivedi K. *Optimal rejuvenation for tolerating soft failures*. Performance Evaluation, Vol. 27 & 28, pp. 491-506, 1996

[3]: Ascher, H.E. Lin, T.-T.Y. Siewiorek, D.P.. *Modification of: error log analysis: statistical modeling and heuristic trend analysis*. IEEE Transactions on Reliability, Vol. 41(4), 1992

[4]: Bev Littlewood , Lorenzo Strigini,. *Software reliability and dependability: a roadmap*. Proceedings of the conference on The future of Software engineering, p.175-188, Limerick, Ireland, 2000

[5]: Bobbio, Garg, Gribaudo, Horvath, Sereno, Telek. *Modeling Software Systems with rejuvenation, restoration and checkpointing through fluid petri nets*. Proc. 8th Int. Workshop on Petri Net and Performance Models (PNPM'99), Zaragoza, Spain, pp. 82-91, 1999

[6]: Cortes C. and V. N. Vapnik. *Support vector networks*. Machine Learning, 20: 273-297, 1995

[7]: Dyn Nira, Levin David. *Iterative Solutions of systems originating from integral equations and surface interpolation*. Siam J. of Numer. Anal. 20(2), 1983

[8]: Ozalp Babaoglu, Mark Jelasity, Alberto Montresor, Christof Fetzer, Maarten van Steen, Aad van Moorsel, Stefano Leonardi (Eds.). *Self-Star Properties in Complex Information Systems*. Proceedings of the International

Workshop on Self-Star Systems, Bertinoro (Forli), Italy, May-June, 2004

[9]: Edsger Wybe Dijkstra. *Notes On Structured Programming*. Technical Report 70-WSK-03, Technological University Eindhoven, Department of Mathematics (http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF), 1970

[10]: Efron B., Tibshirani R.. *An Introduction to the Bootstrap*. Monographs on statistics and applied probability 57, Chapman & Hall, 1993

[11]: Garg S. , Y. Huang, C. Kintala and K. S. Trivedi.. *Time and Load Based Software Rejuvenation : Policy, Evaluation and Optimality*. First Fault Tolerance Symposium, FTS-95, 1995

[12]: Geman, S., Bienenstock, E. and Doursat, R.. *Neural Networks and the Bias/Variance Dilemma*. Neural Computation, 4, pp. 1-58, 1992

[13]: Girosi Federico, Jones Michael, Poggio Tomaso. *Regularization Theory and Neural Networks*. MIT Cambridge, AI Memo 1430, 1993

[14]: Gray J.. *A census of Tandem system availability between 1985 and 1990*. IEEE Transactions on reliability 39(4), 1990

[15]: Hastie T., Tibshirani R.. *Discriminant adaptive nearest neighbor classification*. IEEE PAMI 18, pp. 607-616, 1996

[16]: Hertz J., Krogh P., Palmer R.. *Introduction to the Theory of Neural Computation*. A Lecture Notes Volume in the Santa Fe Institute Studies of Complexity, Vol. I, 1991

[17]: Hochreiter Sepp, Obermayer Klaus, Isabelle Guyon, Steve Gunn, Masoud Nikrav. *Nonlinear Feature Selection with the Potential Support Vector Machine*. Feature extraction, Foundations and Applications, Springer, 2004

[18]: Hoffmann G. A.. *Failure Prediction in Complex Computing Systems: A Probabilistic Approach*. Dissertation at Humboldt University Berlin, ROK Group, 2005

[19]: Horn, P.. *Autonomic Computing: IBM's perspective on the state of Information Technology*. IBM Research Report, 2001

[20]: Huang Y., C. Kintala, N. Kolettis and N. Fulton. *Software Rejuvenation: Analysis, Module and Applications*. In Proc. of the 25th IEEE Intl. Symp. on Fault Tolerant Computing (FTCS-25), Pasadena, CA, 1995

[21]: Huber P. J.. *Data Analysis and Projection Pursuit - A Tutorial Introduction*. Dept. of Mathematics, MIT Boston, 1990

[22]: Hutchinson James M.. *A Radial Basis Function Approach to Financial Time Series Analysis*. Dissertation, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1994

[23]: Joao F. G. de Freitas. *Bayesian Methods for Neural Networks*. Trinity College, University of Cambridge and Cambridge University Engineering Department, PhD Thesis, 1999

[24]: M. J. D. Powell. *Radial basis functions for multivariable interpolation: A review.*. J. C. Mason and M. G. Cox, editors, Algorithms for Approximation of Functions and Data, pp. 143-167, 1987

[25]: Miroslaw Malek, Felix Salfner, Günther Hoffmann. *Self-Rejuvenation - an Effective Way to High Availability*. SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems, Bertinoro, Italy, 2004

[26]: Peitgen, H., Jurgens, H., Saupe, D. *Lyapunov exponents and chaotic attractors in Choas and fractals - new frontiers of science*. Springer, New York, 1992

[27]: Poggio T., Girosi F.. *A Theory of Networks for Approximation and Learning*. Proc. of IEEE 78(9), 1990

[28]: Prior, A. N. *Past, Present and Future*. Oxford: Clarendon Press, 1967

[29]: R. Milner. *Communication and Concurrency*. Prentice Hall, 1989

[30]: Rabiner L. R.. *A Tutorial on Hidden Markov Models*. Proc. of IEEE, Vol.77, No.2, 1989

[31]: Rumelhart, Hinton, Williams. *Learning internal representation by error propagation*. In Rumelhart, McCleland eds. Parallel Distributed Processing Volume I, MIT Press, 1986

[32]: Salfner F., Tschirpke S., Malek M.. *Comprehensive Logfiles for Autonomic Systems*. IEEE Proceedings of IPDPS 2004 (International Parallel and Distributed Processing Symposium), 2004

[33]: Salfner, F. and Hoffmann, G.A. and Malek, M.. *Prediction-based Software Availability Enhancement*. Self-Star Properties in Complex Information Systems; Editors: O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel and M. van Steen, LNCS, vol. 3460, hot topics series; Springer-Verlag, 2005

[34]: Schoelkopf B., Smola A.. *Learning with Kernels*. MIT Press, 2002

[35]: Siewiorek, Swarz. *Reliable Computer Systems Design and Evaluation*. The Digital Press, 2nd edition, 1992

[36]: Takens Floris. *Detecting strange attractors in turbulence.*. Eds.: D.A. Rand and L.S. Young, Dynamical Systems and Turbulence, volume 898 of Lecture Notes in Mathematics, pages 366-381, New York, Springer, 1981

[37]: Trivedi Kishor S., Vaidyanathan Kalyanaraman and Goseva-Popstojanova Kater. *Modeling and Analysis of Software Aging and Rejuvenation*. Center for Advanced Computing & Communication, Dept. of Electrical & Computer Engineering, Duke University, Durham, NC 27708, USA, 2000

[38]: Weigend A. S., Gershenfeld N. A., eds.. *Time Series Prediction*. Proceedings of the Santa Fe Institute, Vol. XV, 1994