A Best Practice Guide to Resource Forecasting for Computing Systems

Guenther A. Hoffmann, Kishor S. Trivedi, *Fellow, IEEE*, and Miroslaw Malek

***Abstract*—Recently, measurement-based studies of software systems have proliferated, reflecting an increasingly empirical focus on system availability, reliability, aging, and fault tolerance. However, it is a nontrivial, error-prone, arduous, and time-consuming task even for experienced system administrators, and statistical analysts to know what a reasonable set of steps should include to model, and successfully predict performance variables, or system failures of a complex software system. Reported results are fragmented, and focus on applying statistical regression techniques to monitored numerical system data. In this paper, we propose a best practice guide for building empirical models based on our experience with forecasting Apache web server performance variables, and forecasting call availability of a real-world telecommunication system. To substantiate the presented guide, and to demonstrate our approach in a step by step manner, we model, and predict the response time, and the amount of free physical memory of an Apache web server system, as well as the call availability of an industrial telecommunication system. Additionally, we present concrete results for a) variable selection where we cross benchmark three procedures, b) empirical model building where we cross benchmark four techniques, and c) sensitivity analysis. This best practice guide intends to assist in configuring modeling approaches systematically for best estimation, and prediction results.**

***Index Terms*—Apache web server, failure forecasting, monitoring, non-parametric modeling, prediction of resource utilization, quantitative analysis, statistical modeling, telecommunication systems.**

ACRONYM[1]

ARMA	autoregressive moving average
AR	autoregressive
AUC	area under curve
PWA	probabilistic wrapper approach
RBF	radial basis functions
SVM	support vector machines
UBF	universal basis functions

Manuscript received January 15, 2007; revised May 1, 2007; accepted June 4, 2007. Associate Editor: Y. Dai.

G. Hoffmann is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA, on leave from the Computer Architecture and Communication Group, Department of Computer Science, Humboldt University of Berlin, Berlin, Germany (e-mail: gunho@ee.duke.edu).

K. Trivedi is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: kst@ee.duke.edu).

M. Malek is with the Computer Architecture and Communication Group, Department of Computer Science, Humboldt University of Berlin, 10099 Berlin, Germany (e-mail: malek@informatik.hu-berlin.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2007.909764

[1]The singular and plural of an acronym are always spelled the same.

I. INTRODUCTION

OVER the past several decades, computing performance has increased by orders of magnitude. At the same time, the software driving these (frequently networked) systems has grown in complexity up to the point where its behavior is in parts unpredictable. Software related failures are now predominant, and can be seen as a threat to the benefits that computing systems aim to provide. Classical approaches to increase dependability such as testing, formal methods, and fault injection can become too rigorous to scale up to enterprise systems, and are not always sufficient to provide the level of dependability needed for real-world applications [13], [34]. Also, system patches, and updates can transform an already tested system into a new, untested system, or introduce bugs. Furthermore, unpredictable industrial environments make it difficult if not impossible to apply the rigor we find in traditional approaches [1], particularly given real-world resource constraints such as time, and money.

A number of reasons have led to an increased complexity of software systems; for example, existing systems are extended, and integrated with legacy infrastructure. New functionality is added in an ad-hoc manner without fully understanding the impact on the entire system. Software systems tend to be embedded in complex, highly dynamic, and frequently decentralized organizations. The support, maintenance, and premises of these systems can change frequently, which provokes ad-hoc reconfiguration. Fewer custom-made software systems, and components are constructed. Instead, generic commercial-off-the-shelf (COTS) components are built, and integrated to provide the intended overall services. The focus of these components is often on interoperability, and re-usability. Thus, software components with a great variance in their parameterization, and application domains are becoming more wide spread. These components may come from third party vendors, and may not be fully specified, and tested.

Additionally, industrial computing systems frequently have significant nonfunctional constraints on their operations. They must meet temporal constraints, be fault-tolerant, and deliver high performance. Architectural features such as caches, pipelines, out-of-order execution, and branch prediction are implemented for performance reasons. Failover, checkpointing, restart, and rejuvenation techniques, to name a few, are implemented for increased dependability. By introducing these features, stochastic dynamics is induced in the overall system.

Following these trends, there is strong reason to believe that the complexity of future software systems will keep increasing, and thus dependability as well as performance issues will keep demanding more attention. This situation demands an empirical probabilistic perspective of complex software systems as re-

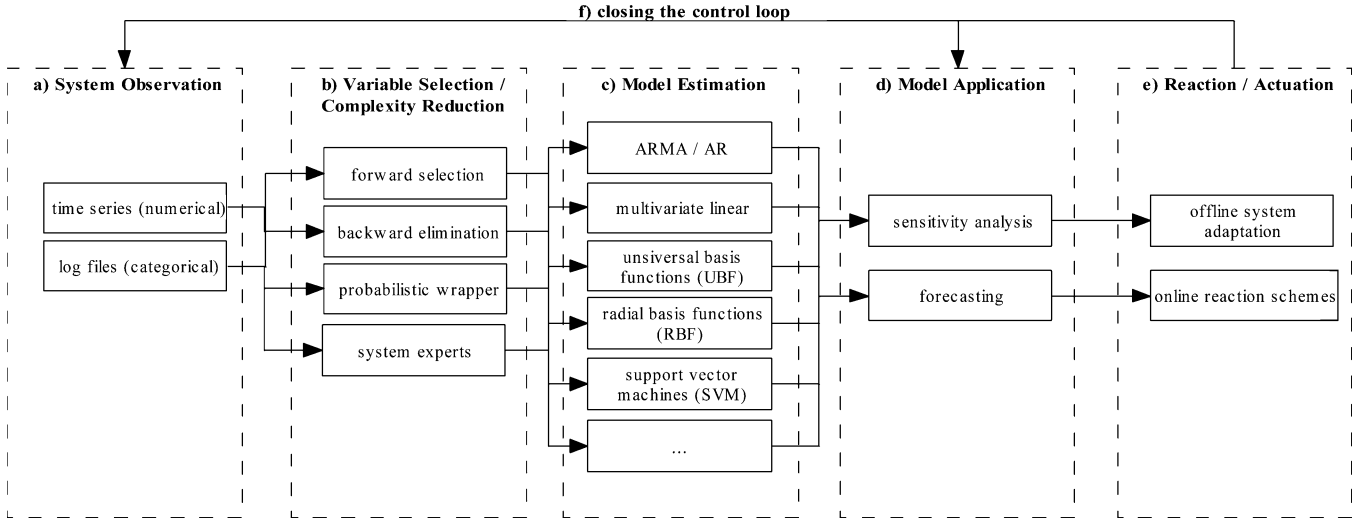

Fig. 1. Building blocks for modeling, and forecasting performance variables, as well as critical events in complex software systems either during runtime (online) or during offline testing.

flected in numerous *measurement-based* studies [15], [20], [22], [26], [27].

However, procedures, and methods detailed in these studies mainly focus on statistical regression of monitored system data. Issues pertaining to

a) the type of data to use,
b) the type of modeling technique to use,
c) the type of parameter optimization technique to apply, and
d) the type of variable selection procedure to employ

are frequently treated with low priority. Piecing the plethora of techniques in these respective subtopics together is more like an art than a science. However, we believe there is much to be gained by integrating these dispersed techniques into one coherent approach. This best practice guide is based on the experience we have gained when investigating these topics:

a) complexity reduction, showing that selecting the most predictive subset of variables contributes more to model quality than selecting a particular linear or nonlinear modeling technique [26], [28];
b) information gain of using numerical vs. categorical data [26], finding that including log file data into the modeling process counter intuitively degrades model quality for prediction of call availability of a telecommunication system; and
c) data-based empirical modeling of complex software systems [27] cross benchmarking five linear, and nonlinear modeling techniques, finding nonlinear approaches to be consistently superior than linear approaches, however, not always significantly.

In this paper, we integrate these steps into one coherent approach. To help guide the practitioner in reaching a feasible modeling & forecasting solution (vs. finding the optimal solution), we summarize our experience gained so far in a *best practice guide,* and substantiate these steps with three examples. Firstly, we model, and forecast the *response time* of an Apache web server in the short (5 minutes ahead), and long (48 hours ahead) timescales. Secondly, we model, and forecast the amount of *free physical memory* of the Apache web server in the short timescale. We also present results for call availability prediction of an industrial telecommunication system.

The paper is organized as follows. In Section II, we review related work in empirical modeling, and variable selection. In Section III, we introduce the basic terminology. In Section IV, we introduce our proposed best practice guide, which we put to the test in Section V. In Section VI we present the results from the empirical part of the paper. In Section VII we summarize our findings.

II. Related Work

The primary goal of empirical studies of complex software systems is to gain better understanding of the way the system works. Based on observations about its past behavior, statistical models are derived to capture its dynamics, and to make forecasts about its future behavior. This approach is in fact similar to long standing approaches in financial engineering, gene expression analysis, or physics [51]. The methods include in some form or the other one of the following (see Fig. 1):

— *Monitoring/collecting data* describing the temporal evolution of the system. This can include *numerical time series data,* or *categorical log files* (Fig. 1(a)).
— *Estimating statistical models* based on these observations. This is sometimes called *regression* or *function approximation* (Fig. 1(c)).
— *Forecasting future events,* or realizations of specific variables. This includes performance variables (e.g. memory or response times), or significant events (e.g. failures) (Fig. 1(d)).
— *Closing the control loop* by implementing some reaction/ actuation scheme to help self-manage the system (e.g. failover, load shedding, rejuvenation and others) (Fig. 1(f)).

System observations (Fig. 1(a)) include numerical time series data, and/or categorical log files. The variable selection process (Fig. 1(b)) is frequently handled implicitly by system experts' ad-hoc theories, or gut feeling; rigorous procedures are applied infrequently. In recent studies, attention has been focused on

the model estimation process (Fig. 1(c)). Univariate, and multivariate linear regression techniques have been at the center of attention. Some nonlinear regression techniques, such as universal basis functions, or support vector machines, have been applied as well. While forecasting has received a substantial amount of attention, sensitivity analysis (Fig. 1(d)) of system models has been largely marginalized. Closing the control loop (Fig. 1(f)) is still in its infancy. Choosing the right reaction scheme (Fig. 1(e)) as a function of quality of service, and cost is nontrivial.

The main idea is that, once a model is in place, it can predict future events in advance, which could be used to trigger preventive measures. Three issues that are important in the overall concept have been largely ignored in the published literature:

— *Complexity reduction*, synonymously also called *variable selection* (Fig. 1(b));
— Sensitivity analysis (Fig. 1(d)); and
— Choice of reaction/ actuation schemes (Fig. 1(e)).

We revisit these building blocks briefly in the following sections.

A. Modeling Techniques

A number of modeling techniques have been applied to software systems: probability models, linear & nonlinear statistical models, expert system-based models, and fractal-based models. In the following sections, we briefly review some of these techniques together with some of their applications.

Probability Models: Queuing models of various types, Markov/semi-Markov chains, stochastic Petri nets, and their combinations have been used for performance modeling. Besides throughput and response time, blocking probability are the measures computed from these models [6]. When a call is blocked in a telecommunication system due to congestion, we refer to the event as having caused call unavailability, even though there is no actual failure in the system. See Section V-C for an example.

Reliability block diagrams, fault trees, Markov/semi-Markov chains, stochastic Petri nets, and their combinations have been used for reliability and availability modeling [47]. Such probability models can sometimes be solved in closed-form, but will commonly be solved numerically, and sometimes by discrete-event simulation.

The key difficulty with such models is the parameterization & validation. Clearly, measurement data are necessary for these steps. If the model is to be used in a closed-loop fashion, then it should be able to take the input data online, track changes in system dynamics, quickly compute the desired measures, and initiate the new control setting. Normally, the structure of the model is derived from understanding the system structure/behavior, but it is also possible to derive the models structure from the data collected from a real system [49].

Specifically in the context of software aging & rejuvenation, a number of probability models have been proposed. In [20], a Markov model of a software system which serves transactions is presented. Due to aging, the service rate of the system in question decreases over time, and the software itself experiences hangs & crashes, which results in unavailability. In [30], a Markov model for a long running server-client type telecommunication system is presented. The authors express downtime, and the cost induced by downtime, in terms of the model parameters. The authors in [14], and [15] relax the assumptions made in [30] of exponentially distributed sojourn times, and build a semi-Markov model. This way they find a closed form expression for the optimal rejuvenation time. In [5], and [18], stochastic Petri Nets are developed by allowing the rejuvenation trigger to start in a robust state. This way the authors calculate the rejuvenation trigger numerically. The key idea in [15] is to take the time between failure data directly (without fitting it to a distribution) to determine the optimal rejuvenation trigger interval. The model is thus suitable for closed-loop control. The main idea in [49] is that the model structure (besides model input parameters) is derived from the measured data.

Linear Statistical Approaches: To get a grip on the complexity of real systems, heuristics and data-driven modeling approaches have been proposed by a number of authors [32]. In [2], the dispersion frame technique to distinguish transient errors from intermittent errors, and to predict the occurrence of intermittent errors, is proposed. However, the applied traditional statistical methods are still too rigid to fit distributions where the data were not identically distributed. Other authors suggest heuristics to rejuvenate software components regularly at certain times [48]. This time-triggered approach basically reflects the assumption of a linear model describing software aging, and proposes that regular intervention will increase the system's availability. This method can yield increased availability of systems with static characteristics. However, it is difficult to employ this approach with systems changing their characteristics more dynamically [31].

Literature on applying linear modeling techniques to software systems has been dominated by approaches based on a single, or a limited number of variables. Most models are either based on observations of workload, time, memory, or file tables. Ref. [19] proposes a time-based model for detection of software aging. Ref. [49] proposes a workload-based model for prediction of resource exhaustion in operating systems such as Unix. Ref. [22] collects data from a web server which the authors expose to an artificial workload. They build time series ARMA (autoregressive moving average) models to detect aging, and estimate resource exhaustion times. Ref. [10] proposes the Pinpoint system to monitor network traffic. The authors include a cluster-based analysis technique to correlate failures with components, and determine which component is most likely to be in a faulty state.

The explicit modeling of an adequate point in time for rejuvenation, which would optimize system availability, and at the same time conserve system resources, would allow adaptation to changing system dynamics, is discussed in [14]. This approach would further allow an event-driven initialization of preventive measures. To achieve this goal, data-driven methods have been applied to model complex computer systems. Ref. [22] proposes linear autoregressive moving average (ARMA) models to predict resource exhaustion times in the Apache web server running in a Linux environment based on measured time series of system variables. They plan to use the prediction of resource exhaustion to time-optimize the rejuvenation of software components. They compare their approach to a number of linear regression models

described in [9]. The authors indicate that they found nonlinear data relationships, and therefore suggest the use of nonlinear modeling techniques.

Nonlinear Statistical Modeling: Another approach to statistical analysis is nonlinear, nonparametric data analysis [4], [41]. This modeling technique addresses some of the shortcomings of classical statistical techniques, in particular:

— the need to model nonlinear dependencies among system variables,
— to cope with incomplete, noisy data,
— to handle changing dynamics,
— to cope with inconclusive data, and
— the need to retrofit a system.

In particular, the need to retrofit a system (i.e. take the data available at runtime without further interfacing the system) has become an important aspect given that most legacy systems are difficult to interface after they are deployed. Most industrial software systems have some form of fault detection at runtime. This is typically implemented in the form of limit checking of some parameters observed in real time, which are compared against some preset value. If this predetermined health corridor is violated, the system will give an alarm. Sensors have typically been implemented during the development of the source code, and tend to be difficult to change later on, not to mention the implementation of new sensors if the system is in the production state. In some software systems, counters are special implementations of sensors, and are used synonymously in this context. Additionally, the implementation of sensors in real systems can become prohibitively expensive in terms of system load, which further complicates their implementation; i.e. data must be stored for online or offline retrieval. The overhead of monitoring, storage, and retrieval operations can easily become prohibitively large.

Statistical models predicting resource demand of an algorithm based on observations of its historic behavior are developed in [25]. In reference [17], the author introduces *Stochastic Time Demand Analysis* for determining a lower bound on the rate at which deadlines are met in fixed-priority systems. An example of a probabilistic discrimination of transient versus permanent faults is given in [40]. The authors present a procedure based on applying Bayesian inference to observed events to diagnose a system at run time. They report an optimal procedure for discrimination of transient versus permanent faults. In ref. [33] the authors report the application of a number of nonlinear learning algorithms to model & predict run-specific resource usage as a function of input parameters supplied to the models at run time. The author applies his models to optimize resource management by estimating resource requirements before a scheduling decision is made. Ref. [49] proposes a strategy to estimate the rate of exhaustion of operating system resources as a function of time, and system workload. They construct a semi-Markov reward model. Ref. [50] predicts response times of an M/G/1 processor-sharing queue. A similar approach is considered in [38]. The authors propose using nonlinear regression techniques to predict the optimal point in time for rejuvenation. Data obtained from a software system by measurement are subjected to statistical analysis in [7]. The authors model computation times, and extend their approach to provide probabilistic scheduling in [8]. Statistical analysis of non-critical computing systems has been carried out in [46]. The authors analyze run-time traces, and conclude that they can automatically extract temporal properties of applications by tracing a reasonably small set of system events. Prediction-based software availability enhancement is proposed in [37]. The authors present modeling approaches based on nonlinear regression techniques, and Markov chains to predict upcoming failures in a software system. A novel nonlinear modeling technique universal basis functions (UBF) was introduced in [29]. In fact, model quality was shown to be more sensitive to the right variable mix than to a particular linear or nonlinear regression technique.

Expert Systems: An expert system approach is taken in [52], [53]. In the *Timeweaver* project, the author predicts failures in telecommunication equipment. The author identifies log files as a source of system state description. In his case study, he uses genetic algorithms to evolve rules based on log files which have been collected during the runtime of a telecommunication system. The rules found showed clear predictive capabilities. Another noteworthy result is presented in [42]. The authors analyze log files as a potential resource for system state description, and calculate the entropy of log files.

Fractal-Based Modeling: Chaotic behavior in data selected from servers used in regular office environments such as file & print servers is reported in [44]. The authors collected a set of data consisting of two variables: *unused memory pages*, and *free swap space*. The authors calculate the Hölder exponent [39] of the data, which is an important mathematical concept for studying chaotic behavior in other disciplines such as medical sciences, signal processing, and economics. They report multi-fractal behavior in both time series, and higher self similarity for higher workloads. The fractal approach is further developed in [12], and is turned into a predictive system in [45]. The authors report an algorithm with which they successfully predicted upcoming system crashes in 80% of their experiments. However, they do not give other metrics such as residual errors, area under curve (AUC), precision, recall, or F-Measure to make their approach comparable.

B. Variable Selection

Given the complexity of industrial software systems, the number of observable variables can easily reach an order of magnitude which makes it difficult if not impossible to assess the influence of each variable on the predictive quality of the model due to combinatorial explosion. In fact, including unfavorable variable combinations can degrade model performance significantly [3], [21], [24]. This problem is known under a variety of names such as *variable selection*, *dimension reduction*, or *feature detection* which are used synonymously. The problem is to find the smallest subset of input variables which are necessary or suffice to perform a modeling task. This type of problem is one of the most prevalent topics in the machine learning & pattern recognition community, and has been addressed by a number of authors such as [36], and more recently by [23].

In the context of software systems, we have previously applied variable selection techniques to optimize failure prediction in a telecommunication system [27], and to identify the most

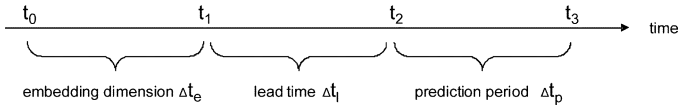

Fig. 2. Time frames for modeling and prediction.

predictive variables for resource prediction in an Apache web server [28].

C. Sensitivity Analysis

One important application of an empirical model beyond prediction is sensitivity analysis with respect to the model's input variables. By investigating each variable's contribution to the overall quality of the model, and by evaluating potentially nonlinear relationships between a variable and the target (e.g. response time, or system failure) variable, constellations may be identified that yield potential insights into the root cause of a failure or misbehavior of a performance variable. For example, in earlier work [26], we identified two distinct variables, the number of semaphores per second, and the amount of memory that was allocated by the kernel, which only at a particular parameterization made the system's call availability drop below a threshold value. This provided system engineers with additional information to track down the root cause of this behavior.

III. Terminology

The modeling & forecasting tasks we look at are straightforward. We are given a matrix $\mathbf{x}$ of observations $\mathbf{x} = \{x | x = \langle f_1, \ldots, f_n \rangle\}$. Based on $\mathbf{x}$, we compute a function Cl that predicts the target variable f_n from the observed features $\langle f_1, \ldots, f_{n-1} \rangle$. The target variable is in our case either *response time*, or *free physical memory*. These data have been monitored from an Apache web server system. Each element $f \in \mathbf{x}$ is a vector of features where we denote $\langle f_1, \ldots, f_{n-1} \rangle$ as the input features, and f_n as the target variable. Given a previously unseen observation matrix $\mathbf{x}_{new}$ with an unknown target value f_n, we obtain $f_n = Cl(\mathbf{x}_{new})$. We calculate the prediction error at time t_3 which is $t_1 + \Delta t_l + \Delta t_p$. The *prediction period* Δt_p occurs some time after the prediction is made, and covers the time frame in which the prediction is valid. In our experiments, we use $\Delta t_p = 5$ minutes, and $\Delta t_p = 60$ minutes. The *lead time* Δt_1 defines how far the prediction extends into the future. We predict the scalar value of the target variables at time $t + \Delta t_l$ in the long term with $\Delta t_l = 48$ hours, and in the short term with $\Delta t_l = 5$ minutes (i.e. one step ahead).

$$lead\,time \qquad \Delta t_l = t_2 - t_1 \tag{1}$$

$$prediction\,period \qquad \Delta t_p = t_3 - t_2 \tag{2}$$

The *embedding dimension* Δt_e which is denoted as

$$embedding\,dimension \qquad \Delta t_e = t_1 - t_0 \tag{3}$$

specifies how far the labeled observations $f \in \mathbf{x}$ extend into the past (Fig. 2).

For model building & testing, we split the dataset into three equally large segments (Fig. 3(b)). We use the first segment for parameterizing the model, the second segment to cross-validate the model and limit over-fitting, and the third segment to measure the models generalization performance on data which has not been presented to the model for parameter estimation.

IV. Proposed Best Practice Guide

To help guide the practitioner, we summarize the experience we gained, while handling data of an industrial telecommunication system, and a networked Apache web server cluster; and we cast it into the following *best practice guide*.

1) *Data splitting*: split your data set into three distinct subsets. Use the first subset to parameterize your model. Use the second subset to validate your model, and to prevent overfitting. Apply the model to the third subset, which has not been presented to the model before (Fig. 3(b)). We call this third subset the *prediction*, *generalization*, or *testing* subset. If the residual errors vary significantly from data set to data set, there may be a significant change in dynamics hidden in the data.
2) *Data preprocessing*: if the observed features are not commensurate, standardize them to zero mean, and unit variance. Experiment with a number of different data preprocessing techniques, and embedding dimensions (see Fig. 2). Even though *time series data* have been found yielding more predictive models in earlier experiments [26], [27], this step may heavily depend on characteristics of the data at hand.
3) *Variable ranking:* rank the observed features by their correlation coefficient with the target, build correlation coefficients for
   1) time series data
   2) class labels, such as 'failure,' and 'no failure.' Map the class labels to numeric values.
4) *Variable set reduction:* Reduce the variable set with a technique for complexity reduction, e.g. forward selection, backward elimination [23], or probabilistic wrapper (PWA) [28].
5) *Variable selection:* Filter the most important variables
   1) in conjunction with a linear classifier by forward selection,
   2) in conjunction with a linear classifier by backward elimination,
   3) by encouraging experts to contribute their knowledge, or
   4) by using the PWA method particularly when suspecting non-obvious nonlinear interdependence of variables.
6) *Model building for classification:* if the distribution of the minority to majority class is unbalanced, adjust it to be evenly distributed, e.g. by re-sampling. Obtaining data in a form that is suitable for machine learning can be prohibitively costly. Events of interest may occur infrequently up to the point where the trade off between creating a useful data set by obtaining, cleaning, transporting, storing, and labeling the data, and the potential benefits from predicting rare events may become unattractive [26].
7) *Model building linear:* start with simple models, e.g. by using multivariate linear models.

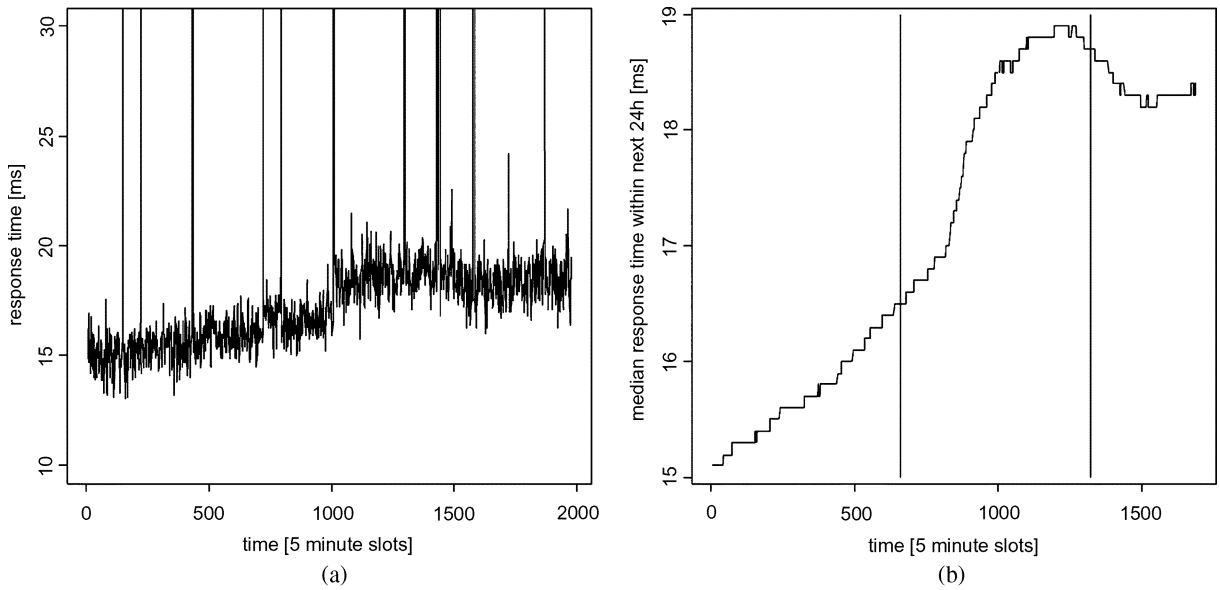

Fig. 3. (a) Response time of Apache server with spikes. (b) To filter out spikes and to detect resource exhaustion more easily, we calculate the median of the response times for the next 24 hours over a sliding window. Vertical solid lines segment the data into a training set (first segment), a validation set (second segment), and a generalization set (third segment).

8) *Model building nonlinear:* only when suspecting nonlinear dynamics, use the variables found in the previous step as input to nonlinear models, e.g. universal basis functions (UBF) or support vector machines (SVM).
9) Experiment with both *data types* if available, a) numerical time series, and b) categorical log file data. We found time series data to be a better predictor than log file data [26]. However, this may ultimately depend on the quality of the log file data available for the system in question [2], [42].
10) Calculate *model confidence*, e.g. by using bootstrapping techniques [16], and verify statistical significance, e.g. by t-testing.

V. Case Studies: Modeling and Predicting

To substantiate our proposed guide, we apply it to three case studies. First, we use the data set introduced in [22] to model & predict two Apache performance variables:

a) *response time*, and
b) *free physical memory*.

Secondly, we use the data captured in [27] to model & predict the call availability of an industrial telecommunication system. The authors in [22] captured data from a server running Apache, and a client connected via an Ethernet local area network. The data set contains 102 variables measured continuously once every five minutes over a period of about 165 hours (i.e. 1979 measurements). We enhance those data by including the first differences, which doubles the number of variables. Thus, we work with 204 variables in total. Also, we present results of modeling & forecasting the call availability of an industrial telecommunication system.

A. Apache: Forecasting Response Time

Data Preprocessing: Response time is the interval from the time a client sends out the first byte of a request until the client receives the first byte of the reply. In Fig. 3(a), we show a plot of the measured *response time*. Two characteristics attract our attention. First, there are significant spikes. Second, the response time increases over time to flatten out in a plateau. Reference [22] concludes that the latter is caused by resource exhaustion. By looking at the time between spikes (see Table I for the complete list), we find that they occur approximately in multiples of 6 hours.

TABLE I
Lag Times Between Spikes.
The left column shows the number of time slots between spikes measured in 5 minute intervals. This translates into the minutes shown in the middle column. The approximate hours are shown in the right column.

# of 5 minute lag intervals	Minutes	~hours
73 and 74	369	6
150	750	13
213	1065	18
286	1430	24

However, that does not give us the root cause; but it indicates some areas for further research. Potential explanations include a) cron-jobs in additive 6 hour intervals, b) additive effects of update cycles of many machines affecting the network, or c) broadcast messages. More theories concerning the root cause could be developed based on this finding. However, from here on, human experts will have to probe further into the system to find the root cause. Statistical analysis is a potential tool to narrow down the areas to look for a root cause. The increasing response time over time has been explained previously by resource exhaustion [22]. The question remaining is can we forecast the response time such that we could close the control loop, and trigger a preventive maintenance scheme based on this forecast. In the next step, we cross benchmark three variable selection techniques to filter out the most predictive variables.

TABLE II
ROOT MEAN SQUARE ERROR (RMSE) FOR SHORT TERM PREDICTIONS ($\Delta t_p = 24$ hours, AND $\Delta t_l = 5$ minutes) OF RESPONSE TIME FOR THREE CROSS BENCHMARKED VARIABLE SELECTION PROCEDURES: FORWARD SELECTION, BACKWARD ELIMINATION, AND PROBABILISTIC WRAPPER APPROACH (PWA)

	RMSE		
	forward selection	backward elimination	PWA
parametrization	0,012195	0,013896	0,014602
validation	0,046748	0,336004	0,024502

Variable Selection: To filter out the spike effect, we set $\Delta t_p = 24$ hours. This means that, instead of making point predictions, we are rather looking at the median *response time* over the next 24 hours; see Fig. 3(b) for a plot. The vertical solid lines split the data into three sets:

1) a *training* or *parameterization* set which is used to parameterize the statistical models (first segment);
2) a *validation* set which is used to validate the models, and to avoid over-fitting (second segment); and
3) a *generalization* or *testing* set which is used to test the model on previously unseen data (third segment).

In this section, we cross benchmark three procedures for variables selection:

— forward selection,
— backward elimination, and
— probabilistic wrapper (PWA).

These procedures have been introduced, described in detail, and applied in a number of articles. For an introduction to forward selection, and backward elimination, see [23]. For the PWA, see [35]; or see [26], [28] for a more recent application. We investigate short term predictions with $\Delta t_l = 5$ minutes, and long term predictions with $\Delta t_l = 2$ days. In Table II, we report short term results for $\Delta t_p = 24$ hours, and $\Delta t_l = 5$ minutes.

We report the root-means-square-error (RMSE) for the *parameterization*, and the *validation* data set in Table II. Consistent with our earlier findings in [26], and [28], forward selection, and PWA outperform backward elimination by an order of magnitude. PWA shows an RMSE of 0.025 on the validation data set, while backward elimination shows an $\mathrm{RMSE} = 0.336$. As a result of the variable selection step, we choose the variables identified by the PWA variable selection procedure which scored the smallest RMSE on the validation data set. These variables are shown in Table III. The *number of new processes*, the *number of blocks written to the disk*, and the *number of Ethernet packets transported* were identified as the top three variables.

In the next step, we apply the same three variable selection procedures to a long term prediction task with $\Delta t_p = 24$ h, and $\Delta t_l = 2$ days. Results for this setting are shown in Table IV. Forward selection, and PWA clearly outperform backward elimination on the parameterization as well, as on the validation data set. PWA scored the smallest error with $\mathrm{RMSE} = 0.01$. The resulting set of variables is shown in Table XI.

It is interesting to note that the most predictive variable sets identified for short, and long term prediction share an intersection of core variables which are

a) *NewProcesses*,
b) *eth0_recv_pkts*,
c) *ContextSwitches*, and
d) *si_size_256*

TABLE III
THE ACTUAL VARIABLES SELECTED BY THE PWA PROCEDURE FOR (a) SHORT TERM AND (b) LONG TERM PREDICTION OF RESPONSE TIME. NOTE THAT VARIABLE NAMES IN ITALIC FONT INDICATE FIRST DIFFERENCES RATHER THAN THE ABSOLUTE VALUE OF THAT VARIABLE.

Variable Nr.	Variable Name
30	NewProcesses
23	DiskWrittenBlocks
89	eth0_trans_pkts
65	lo_recv_pkts
27	SwapOutCounter
63	si_slab_cache
59	si_size_256
29	*ContextSwitches*
6	*CachedMemory*

(a)

Variable Nr.	Variable Name
30	NewProcesses
81	eth0_recv_pkts
29	ContextSwitches
18	TimeIdleMode
72	lo_trans_bytes
26	SwapInCounter
59	si_size_256
62	si_size_32
29	*ContextSwitches*

(b)

TABLE IV
ROOT MEAN SQUARE ERROR (RMSE) FOR LONG TERM PREDICTIONS ($\Delta t_p = 24$ hours AND $\Delta t_l = 2$ days) OF *response time* FOR THREE CROSS BENCHMARKED VARIABLE SELECTION PROCEDURES: FORWARD SELECTION, BACKWARD ELIMINATION, AND PWA

	RMSE		
	forward selection	backward elimination	PWA
parametrization	0,006096	0,018397	0,005312
validation	0,011146	0,716490	0,010160

In addition to this base set of variables, more counters and variables are added as needed by the PWA procedure (compare Table III(a) to III(b)).

Model Building: In the previous step, we have identified the most predictive variables for long term (Table III(b)), and short term (Table III(a)) predictions of *response time*. We have identified these variables based on cross benchmarking three variable selection techniques (see Tables II and IV). We selected the variable set yielding the smallest RMSE on the validation data set. We now turn to model building, and forecasting. We use the variable sets identified in the previous step to cross benchmark four modeling techniques for short, and long term predictions. These modeling techniques are

— multivariate linear regression (ML),
— support vector machines (SVM),
— radial basis functions (RBF), and
— universal basis functions (UBF).

Again, these techniques have previously been described in detail, and have been applied to numerous applications. For an introduction to multivariate linear regression (ML), consult any basic text book on statistics. Support vector machines (SVM) have been introduced in [11], and kernel based nonlinear regression techniques such as SVM. and radial basis functions (RBF) have recently been discussed in detail in [43]. Universal basis

TABLE V
ROOT MEAN SQUARE ERROR (RMSE) FOR SHORT TERM PREDICTION ($\Delta t_p = 24$ hours AND $\Delta t_l = 5$ minutes) OF *response time*. WE CROSS BENCHMARK FOUR MODELING TECHNIQUES: MULTIVARIATE LINEAR REGRESSION, SUPPORT VECTOR MACHINES, UNIVERSAL, AND RADIAL BASIS FUNCTIONS. WE REPORT RMSE FOR THE THREE DATA SEGMENTS OF PARAMETERIZATION, VALIDATION, AND TESTING. THE BEST MODEL IS SELECTED BY ITS PERFORMANCE ON THE VALIDATION DATA SET. IN THIS CASE, SVM PRODUCES THE SMALLEST ERROR WITH RMSE $= 0.012$.

	ML	SVM	UBF	RBF
parametrization	0,012540	0,006885	0,008867	0,008116
validation	0,018170	0,012256	0,014236	0,015242
testing	0,084692	0,073457	0,048701	0,076342

TABLE VI
ROOT MEAN SQUARE ERROR (RMSE) FOR LONG TERM PREDICTION ($\Delta t_p = 24$ hours, AND $\Delta t_l = 2$ days) OF *response time*

	ML	SVM	UBF	RBF
parametrization	0,007714	0,006818	0,007985	0,006663
validation	0,009323	0,006852	0,008147	0,008139
testing	0,428725	0,126498	0,064269	0,135337

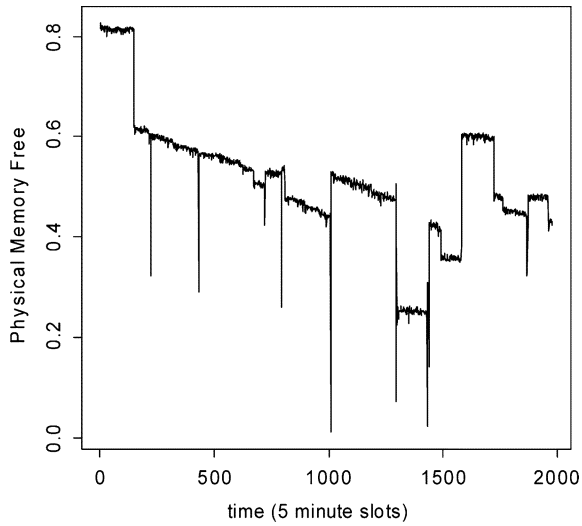

Fig. 4. *Free physical memory* (normalized values) for the Apache web server.

functions (UBF) are an extension of RBF; and we have previously developed, discussed, and applied them in [29], and in the context of forecasting call availability of an industrial telecommunication system in [27].

In Table V, we list RMSE values for *short term* prediction ($\Delta t_p = 24$ hours, and $\Delta t_l = 5$ minutes) of *response time* for the three data segments of parameterization, validation, and testing. In Table VI, we list RMSE values for *long term* prediction ($\Delta t_p = 24$ hours, and $\Delta t_l = 2$ days).

SVM outperform all other modeling techniques for *short term* (Table V), as well as for *long term* predictions (Table VI). For short term predictions, SVM produce a RMSE of 0.012; and 0.007 for long term predictions. Both values are generated from the validation data set. However, this performance does not translate into an equally low forecast error on the testing data set. In the short term, SVM are outperformed on the *testing* (i.e. *generalization*) data set by UBF, and by UBF and RBF in the long term. However, in reality, at the time of model building, we only have the parameterization, and the validation data available. The multivariate linear approach is consistently outperformed by the nonlinear contenders SVM, RBF, and UBF. Based on error results from the *validation set*, we choose the support vector approach (SVM) for forecasting.

Sensitivity Analysis: How sensitive is the outcome of our models to each variable? The knowledge of this sensitivity factor can potentially help increase the knowledge about how the system under scrutiny works.

In this step, we take each of the four models built in the previous *model building* step (ML, SVM, UBF, RBF), and remove each of the variables found in the *variable selection* step one at a time. We then calculate the change in model error RMSE for the validation, and the generalization data set. We report the percentage change in RMSE in Table X. The upper part of Table X shows the data for the validation set, while the lower part shows data for the generalization data set. For the validation set, we also report the linear correlation coefficient of each variable with the target value *response time*. It is interesting to note that variables to which the model is highly sensitive are not necessarily highly correlated with the target. For example, by removing the variable *IO_recv_pkts* from the SVM model, RMSE increases by 23% on the validation data set. However, removing the number of new processes increases the RMSE only by 3.5%.

B. Apache: Forecasting Free Physical Memory

In this section, we use the same approach described in Section V-A. to model the variable *free physical memory of the* Apache server. Fig. 4 shows a plot over time; this variable first decreases, then increases again stepwise. This behavior cannot readily be explained by resource exhaustion due to software aging. In the Apache server, *free physical memory* cannot be lower than a preset threshold value. If physical memory approaches the lower limit, the systems frees up memory by paging. Thus, we get an irregular utilization pattern.

Data Preprocessing: Free physical memory shows similar spikes as *response time* does. However, only a few of the spikes coincide, making a mutual root cause less likely. Besides the spikes, we observe a rather linear behavior except for some abrupt changes when the server frees physical memory. Our ultimate goal is to forecast future realizations of this variable. We start by looking at the first differences of this variable (Fig. 5(a)), and its autocorrelation function (Fig. 5(b)). However, there is not much autocorrelation in the first differences that could be used for modeling. Autocorrelation depletes at the first step, which corresponds to a 5 minute horizon.

Variable Selection: We continue with variable selection, and cross benchmark again with our three selection procedures for $\Delta t_p = 5$ minutes, $\Delta t_l = 5$ minutes, and $\Delta t_e = 5$ minutes. In Table VII, we list the RMSE results for the parameterization, and the validation data set for this step.

Based on the lowest RMSE error values generated on the validation data set, we select the variable set produced by the PWA mechanism. Variables are listed in Table VIII. Given the autocorrelation at lag $t-1$, not surprisingly the *free physical memory* at $t-1$ is in the variable set.

Model Building: We cross-benchmark again the variable set identified in the previous step with the four modeling techniques

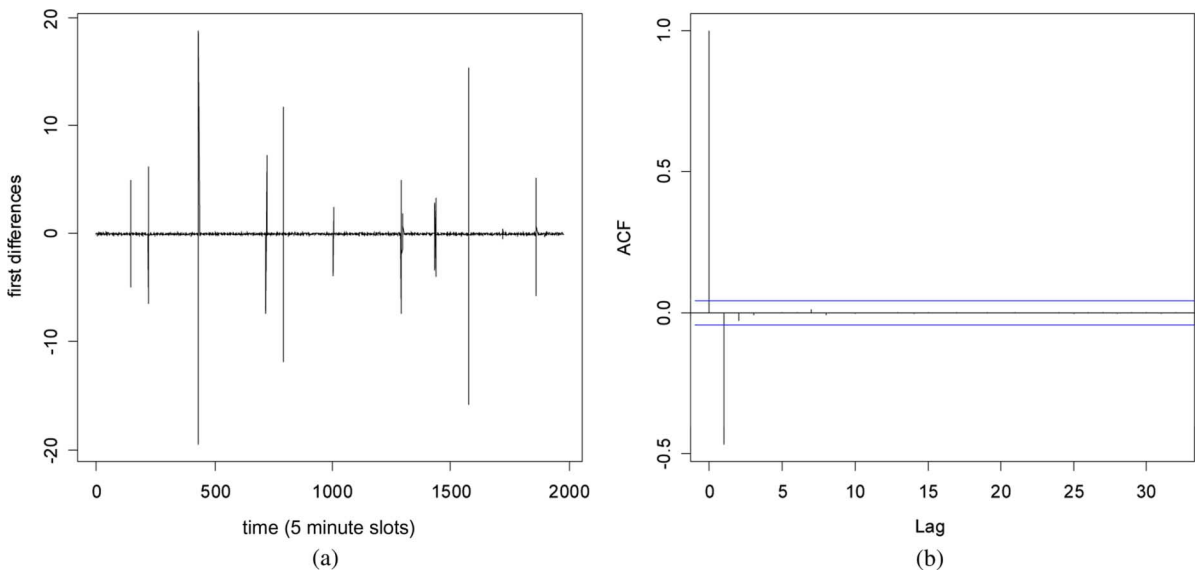

Fig. 5. (a) First differences of the variable *free physical memory*. (b) autocorrelation function of first differences.

TABLE VII
VARIABLE SELECTION: ROOT MEAN SQUARE ERROR (RMSE) FOR $\Delta t_p = 5$ minutes AND $\Delta t_l = 5$ minutes OF THE VARIABLE *free available memory*

	RMSE		
	forward selection	backward elimination	PWA
parametrization	0,026648	0,028293	0,026934
validation	0,019938	0,220954	0,018614

TABLE VIII
THE ACTUAL VARIABLES SELECTED BY THE PWA PROCEDURE. NOTE THAT VARIABLES IN ITALICS DENOTE THE FIRST DIFFERENCE RATHER THAN THE ABSOLUTE VALUE.

Variable Nr.	Variable Name
3	PhysicalMemoryFree at time t-1
15	TimeUserMode
30	*NewProcesses*
48	si_files_cache

TABLE IX
MODEL BUILDING: ROOT MEAN SQUARE ERROR (RMSE) FOR $\Delta t_p = 5$ minutes AND $\Delta t_l = 5$ minutes OF FREE AVAILABLE MEMORY

	ML	SVM	UBF	RBF	ARMA
parametrization	0,017971	0,017580	0,017638	0,017620	0,02231004
validation	0,026998	0,027021	0,026706	0,026759	0,03517245
testing	0,018513	0,051723	0,017123	0,059156	0,02235754

introduced earlier: multivariate linear, support vector machines, radial basis functions, and universal basis functions for $\Delta t_p = 5$ minutes, $\Delta t_l = 5$ minutes, and $\Delta t_e = 5$ minutes. Results are shown in Table IX.

Sensitivity Analysis: In this step, we remove one variable at a time from each model (ML, SVM, UBF, RBF), and calculate the change in the models RMSE for the validation & generalization data set. In Table XI we show the models RMSE for each variable removed, and the percentage change in RMSE. All models show that prediction performance is particularly sensitive to two variables which are *PhysicalMemoryFree* (Table X) at time $t-1$, and *si_files_cache*. Given the high autoregressive correlation, *PhysicalMemoryFree* is an expected result. However, *si_files_cache* shows only a small linear correlation of 0.075 with the target variable; however, its removal would increase the models RMSE by 26% for ML models, and 15.8% for UBF models. In the generalization data set (Table XI, lower part), this effect becomes even more clear. The removal of *si_files_cache* would increase model errors by 54% for ML, and 139% for UBF models.

C. Telecommunication System Call Availability

In this section, we present results for interval call availability prediction of an industrial telecommunication system. We use the data monitored, collected, and preprocessed in [27]. The system handles mobile originated calls; and value added 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). At the time we took the measurements, the system consisted of somewhat more than 1.6 million lines of code, approximately 200 components, and 2000 classes.

Interval call availability $A_c(\Delta t)$ 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 attempted 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 attempted calls in this interval. In our scenario, we derive $A_c(\Delta t)$ as

$$A_c(\Delta t) = 1 - \frac{n_{failed}}{n_{total}} \tag{4}$$

The values of n_{failed}, and n_{tota} 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 $A = A_c(\Delta t)$.

TABLE X
RESULTS FOR SENSITIVITY ANALYSIS FOR SHORT TERM APACHE RESPONSE TIME FORECASTING.
THE UPPER PART SHOWS THE DATA FOR THE VALIDATION DATA SET; THE LOWER PART SHOWS DATA FOR THE GENERALIZATION DATA SET. FOR THE VALIDATION SET, WE ALSO REPORT THE LINEAR CORRELATION COEFFICIENT OF EACH VARIABLE WITH THE TARGET VALUE *response time*. THE LEFT COLUMN LISTS ALL VARIABLES IDENTIFIED IN THE VARIABLE SELECTION STEP. THE NEXT COLUMN LISTS RMSE VALUES WHEN THE VARIABLE IN THE CORRESPONDING ROW HAS BEEN REMOVED FROM THE MODEL. THE FOLLOWING COLUMN LISTS THE PERCENTAGE CHANGES IN MODEL ERROR FOR THIS CASE. CLEARLY *IO_recv_pkts* IS THE ONE VARIABLE WHICH IS MOST INFLUENTIAL IN PREDICTION. ITS REMOVAL INCREASES MODEL ERROR BY 23% FOR SVM MODELS. ON THE GENERALIZATION DATA SET (DISPLAYED IN THE LOWER PART), ITS REMOVAL WOULD STILL INCREASE THE SVM'S MODEL ERROR BY 8%. SEE TEXT FOR FURTHER EXPLANATION.

Sensitivity Analysis

		RMSE on validation data				RMSE change in %				Correlation
Variable Nr.	**Variable Name**	**ML**	**SVM**	**UBF**	**RBF**	**ML**	**SVM**	**UBF**	**RBF**	coefficient
	all listed variables	0,018170	0,012256	0,014236	0,015242					
30	NewProcesses	0,019426	0,012683	0,014612	0,015545	6,91%	3,48%	2,64%	1,99%	0,956424
23	DiskWrittenBlocks	0,019316	0,012656	0,014634	0,015539	6,30%	3,26%	2,79%	1,94%	0,955246
89	eth0_trans_pkts	0,018384	0,012679	0,016104	0,016512	1,18%	3,45%	13,12%	8,33%	0,953614
65	lo_recv_pkts	0,020455	0,015131	0,016818	0,018063	12,57%	23,46%	18,14%	18,51%	0,937351
27	SwapOutCounter	0,021763	0,012855	0,015363	0,017086	19,77%	4,89%	7,92%	12,10%	0,755900
63	si_slab_cache	0,018176	0,012288	0,014059	0,015207	0,03%	0,26%	-1,25%	-0,23%	0,682239
59	si_size_256	0,019173	0,013181	0,012582	0,015144	5,52%	7,55%	-11,62%	-0,64%	0,458101
29	*ContextSwitches*	0,018207	0,011776	0,014161	0,015234	0,20%	-3,92%	-0,53%	-0,05%	0,220309
6	*CachedMemory*	0,018175	0,011837	0,014025	0,014977	0,03%	-3,42%	-1,48%	-1,74%	0,058488
	adjusted set	0,018170	0,011238	0,013654	0,015240	0,00%	-8,31%	-4,09%	-0,02%	

Sensitivity Analysis		RMSE on unseen test data				RMSE change in %			
Variable Nr.	**Variable Name**	**ML**	**SVM**	**UBF**	**RBF**	**ML**	**SVM**	**UBF**	**RBF**
	all listed variables	0,084692	0,073457	0,048701	0,076342				
30	NewProcesses	0,069045	0,068437	0,091184	0,108526	-18,47%	-6,83%	87,23%	42,16%
23	DiskWrittenBlocks	0,153435	0,070006	0,092068	0,105964	81,17%	-4,70%	89,05%	38,80%
89	eth0_trans_pkts	0,231876	0,076665	0,207850	0,085982	173,79%	4,37%	326,79%	12,63%
65	lo_recv_pkts	0,040331	0,079400	0,051371	0,021927	-52,38%	8,09%	5,48%	-71,28%
27	SwapOutCounter	0,270449	0,046728	0,097349	0,160244	219,33%	-36,39%	99,89%	109,90%
63	si_slab_cache	0,091150	0,068403	0,028253	0,053292	7,63%	-6,88%	-41,99%	-30,19%
59	si_size_256	0,061327	0,082852	0,116621	0,023679	-27,59%	12,79%	139,46%	-68,98%
29	*ContextSwitches*	0,084956	0,065106	0,040642	0,085192	0,31%	-11,37%	-16,55%	11,59%
6	*CachedMemory*	0,083107	0,088105	0,042010	0,081243	-1,87%	19,94%	-13,74%	6,42%
	adjusted set	0.084692	0.089188	0.028132	0.035020	0.00%	21.42%	-42.24%	-54.13%

Variable Selection: The data available to us is made up of 46 system variables sampled once per minute, per node, with two nodes. This yields 92 variables in a time series describing the evolution of the internal states of the system. In a 24-hour period a total of 132,480 readings were collected. In total roughly 1.3 million system state observations were collected. Additionally, we made use of 195 failure classes which were collected from textual log files. Considering our two nodes we have available 390 class variables. In total, the data set consisted of approximately four million log file entries.

We employed the widely used area under curve (AUC) metric for model performance. A value close to one signals excellent prediction performance, while a value equal to one half signals a random selection of variables. We subjected both data sets, the numerical time series data as well as the categorical log file classes, to three variable selection techniques: the probabilistic wrapper, forward selection, and backward elimination. Additionally, we asked the experts programming the system to name system variables they considered important to forecast call availability. Results are shown in Fig. 6. The probabilistic wrapper (PWA) approach applied to time series data clearly outperforms all other approaches. Interestingly, the expert selected variables were outperformed by all other variable selection techniques performing close to random sampling. The two variables selected by the PWA approach are

a) *alloc*: The amount of memory, in bytes, that the KMA (Kernel Memory Allocator) has allocated from its large memory request pool to large memory requests.
b) *sema/s*: The number of semaphore operations per second. This figure will usually be zero (0.00), unless you are running an application that uses messages or semaphores.

Prediction Results: In the next step, we turned to failure prediction. We calculated error bounds, and statistical significance for models predicting failures five to 15 minutes $\Delta t_l \in \{5, 10, 15\}$ ahead of their appearance. For comparison, we also report failure recognition performance at $\Delta t_l = 0$.

We investigated models using the two predictive variables identified in the variable selection step. 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 Fig. 7), indicating that nonlinear data correlations play less of a role the further we look ahead. The reported values are medians of their respective distributions.

Sensitivity Analysis: An important application of nonlinear models beyond failure prediction is sensitivity analysis with respect to the model's input variables. System call availability as a function of the two system variables *alloc* and *sema/s* is depicted in Fig. 8.

These two variables have been identified in the previous variable selection step. The UBF model clearly shows a hot zone in which system availability is more sensitively dependent on

TABLE XI
SENSITIVITY ANALYSIS FOR APACHE'S PERFORMANCE VARIABLE *freePhysicalMemory*.
THE UPPER TABLE SHOWS DATA FOR THE VALIDATION DATA SET; THE LOWER TABLE COVERS THE GENERALIZATION DATA SET. FOR THE VALIDATION DATA SET, WE ALSO SHOW THE LINEAR CORRELATION COEFFICIENT OF EACH VARIABLE FOR THE TARGET VARIABLE *freePhysicalMemory*. THE LEFT COLUMN SHOWS ALL VARIABLES IDENTIFIED IN THE VARIABLE SELECTION STEP. THE NEXT COLUMN SHOWS RMSE VALUES WHEN THIS VARIABLE IS REMOVED FROM THE MODEL. THE NEXT COLUMN LISTS CHANGES TO THE MODELS PERFORMANCE IN THIS CASE.

		RMSE on validation data				RMSE change in %				Correlation
Variable Nr.	**Variable Name**	**ML**	**SVM**	**UBF**	**RBF**	**ML**	**SVM**	**UBF**	**RBF**	coefficient
	all listed variables	0,026998	0,027021	0,026706	0,026759					
3	PhysicalMemoryFree at t-1	0,057083	0,050432	0,045837	0,041312	111,44%	86,64%	71,63%	54,39%	0,96355239
15	TimeUserMode	0,027136	0,027309	0,027022	0,026967	0,51%	1,06%	1,18%	0,78%	0,80725707
30	*NewProcesses*	0,027927	0,026295	0,027616	0,027589	3,44%	-2,69%	3,41%	3,10%	0,17648617
48	si_files_cache	0,034094	0,027987	0,030931	0,029154	26,28%	3,57%	15,82%	8,95%	0,07487987
	adjusted set	0,026998	0,026295	0,026706	0,026759	0,00%	-2,69%	0,00%	0,00%	

Sensitivity Analysis		RMSE on unseen test data				RMSE change in %			
Variable Nr.	**Variable Name**	**ML**	**SVM**	**UBF**	**RBF**	**ML**	**SVM**	**UBF**	**RBF**
	all listed variables	0,018513	0,051723	0,017123	0,059156				
3	PhysicalMemoryFree at t-1	0,182444	0,135516	0,370749	0,184623	885,47%	162,00%	2065,19%	212,09%
15	TimeUserMode	0,015504	0,019340	0,016559	0,015847	-16,26%	-62,61%	-3,30%	-73,21%
30	*NewProcesses*	0,018822	0,047455	0,023734	0,088290	1,67%	-8,25%	38,61%	49,25%

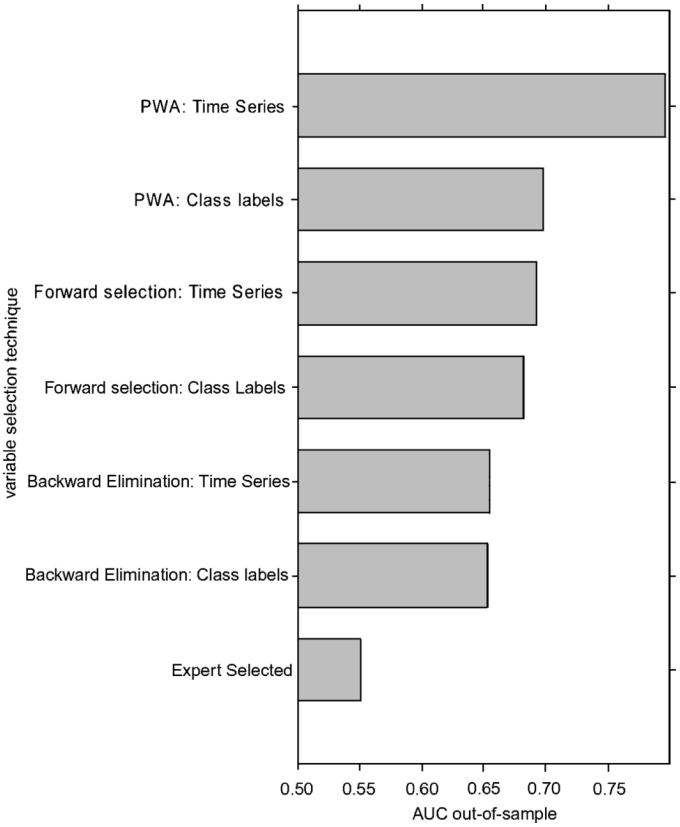

Fig. 6. AUC results for seven variable selection techniques. All results were derived from *generalization data* which had not been used for model parameterization.

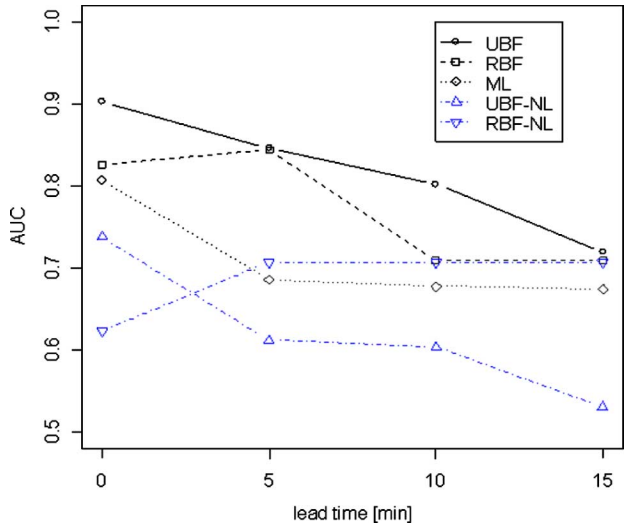

Fig. 7. AUC characteristic plotted over a number of lead times for each modeling technique.

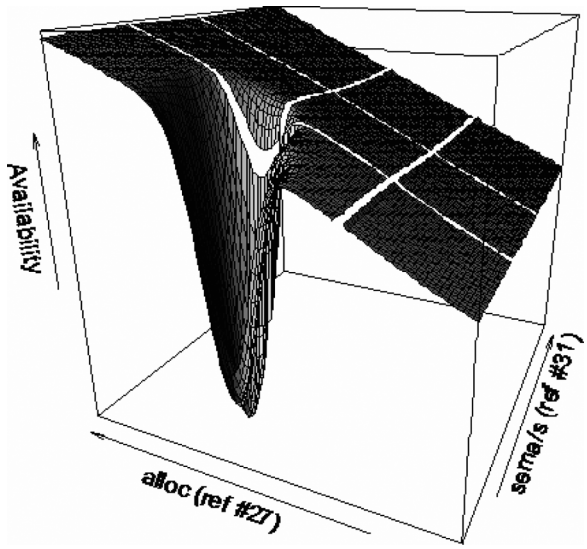

Fig. 8. Call availability as a function of the two SAR variables *alloc* and *sema/s* identified by the variable selection step.

the two regressors than in other areas. The variables have been normalized to unit root. Changing *alloc* considerably influences the system's availability. Changing *sema/s*, however, only moderately influences the system's availability, except for one specific area when the UBF model shows a high sensitivity of the system's availability to *sema/s*. This information may guide the system engineer to fine tune the system with respect to these two variables, or support root cause analysis by guiding the practitioner.

VI. Results

In this section, we summarize our results. In Section III, we introduced the terminology. Based on earlier work, we detailed a best practice guide for empirical modeling of complex software systems in Section IV. In this guide, we focused on variable selection, and sensitivity analysis, two aspects that have received little attention in current literature. In Section V, we put our guide to the test by modeling & predicting two performance variables of the Apache web server **response time**, and *freePhysicalMemory*. Additionally, we presented results for modeling & predicting the call availability of an industrial telecommunication system.

A. Best Practice Guide

It is a complicated, arduous, and time-consuming task for even experienced system administrators, and statistical analysts, to know what a reasonable set of steps should comprise to model, and successfully predict performance variables or system failures of a complex digital system. Thus, this first, and by no means complete attempt of a best practice guide intends to assist in configuring modeling approaches systematically for best prediction results. In four examples, we detail aspects of our guide focusing on variable selection (i.e. *complexity reduction*), modeling, and sensitivity analysis.

B. Variable Selection

The objective of variable selection is threefold: 1) improve the prediction result of the underlying model, 2) provide faster, more resource effective (e.g. time, cost) models, and c) provide a better understanding of the underlying process that generated the data [23]. Regarding the *first* aspect, we have cross benchmarked three popular variable selection procedures (forward selection, backward elimination, and probabilistic wrapper) for four distinct applications (Apache *response time* short, and long term prediction, short term *free physical memory* prediction, and *call availability* of a telecommunication system).

Our results suggest that forward selection, and the probabilistic wrapper approach (PWA) significantly, and consistently outperform the backward elimination procedure by an order of magnitude. This is due to the fact that backward elimination starts out with *all* available variables, and then reduces this set to minimize the model's error. This procedure is easily affected by the curse of dimensionality. This result is also a strong indicator that variable selection should be preferred over an approach which blindly uses *all* available variables. Regarding the *third* aspect, we found that, when predicting the *response time* of the Apache web server, there is a set of basis variables that remains constant, even if the lead time varies from minutes to days. This core variable set can be further used to understand the underlying processes, and for theory building or verification.

C. Statistical Modeling

In our experiments, we cross benchmarked multivariate linear (ML), and multivariate nonlinear (support vector machines, radial, and universal basis functions) data driven modeling techniques. When predicting *response time*, the support vector machine (SVM) approach outperformed other techniques. When predicting *free physical memory*, the universal basis function (UBF) approach turns out to be superior. In all cases, the winning margin is far from being an order of magnitude. This result suggests that focusing resources on variable selection *instead* of the type of modeling technique may yield faster, less costly results.

D. Sensitivity Analysis

We have performed sensitivity analysis for models that predict short term response times, and that predict the *free physical memory* of the Apache web server. We removed each variable from the models, one at a time, and calculated the models change in RMSE. As expected, the RMSE increases for the multivariate linear models with each variable removed. However, this is not the case with nonlinear models. This result suggests that variable selection with a nonlinear wrapper may yield improved model quality. In fact, when we rebuilt SVM, UBF, and RBF models with this new set of variables, we decreased the models error by up to 8% on the validation data set, and by up to 54% on the generalization data set.

VII. Conclusions and Future Work

In this paper, we propose a best practice guide for building empirical models for the prediction of performance variables of computing systems. In three case studies, we validate its applicability by modeling & forecasting Apache web server response time, and *free available memory*, as well as the *call availability* of a telecommunication system. We obtained encouraging results regarding forecasting quality, sensitivity analysis, and complexity reduction (variable selection).

Recent work in empirical modeling of complex computing systems has addressed the modeling problem from the pragmatic point of view by applying statistical regression techniques to monitored data. The focus has been on trend analysis, linear, and nonlinear regression techniques to sometimes high dimensional input spaces. While the step from trend analysis to multivariate regression techniques in some cases showed a significant improvement, the step from multivariate linear to nonlinear procedures has not always resulted in a notable increase in prediction quality. One reason for this observation may be that multivariate techniques working in high dimensional data spaces easily suffer from the curse of dimensionality, and may over-fit the data. We found that applying first a method that automatically reduces the complexity of the input space (i.e. variable selection) yields a) improved prediction performance, and b) a more compact set of variables which can serve as a cornerstone for further system and sensitivity analysis.

The methods we cross-benchmarked in regression techniques and variable selection are diverse, and motivated by various theoretical arguments, but a unifying theoretical framework is lacking. Because of these shortcomings, it is important to have some guidelines before tackling a prediction problem. To that end, we have summarized the experience we gained when modeling & predicting performance variables, and critical events in the Apache web server, and in a commercial telecommunication system in this best practice guide. We recommend focusing the modeling efforts on the variable selection step because it yields consistently the largest improvement (one order of magnitude)

in prediction performance in our test cases. Results from sensitivity analysis suggest that variable selection with a nonlinear wrapper approach may improve model quality further. Favoring nonlinear over linear multivariate regression techniques yields consistent improvements, however they may not always be significant. Additionally, the question of optimal reaction schemes has to be addressed in an effort to close the loop. Also, the question of causality inference (i.e. finding root cause) must be further investigated.

REFERENCES

[1] J. Arlat, Y. Crouzet, and J. Karlsson, "Comparison of physical and software-implemented fault injection techniques," *IEEE Trans. Computers*, pp. 1115–1133, 2003.

[2] H. E. Ascher, T.-T. Y. Lin, and D. P. Siewiorek, "Modification of: error log analysis: Statistical modeling and heuristic trend analysis," *IEEE Trans. Reliability*, vol. 41, no. 4, 1992.

[3] E. Baum and D. Haussler, "What size net gives valid generalization?," *Neural computation*, vol. 1, no. 1, pp. 151–160, 1989.

[4] C. M. Bishop*, Neural Networks for Pattern Recognition*. London: Clarendon Press, 1995.

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

[6] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi*, Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications (The White Book)*, second ed. New York, NY: John Wiley, 2006.

[7] A. Burns, "Predicting computation time for advanced processor architectures," presented at the Euromicro RTS. 12th Euromicro Conf. on Real-Time Systems, 2000.

[8] A. Burns, G. Bernat, and I. Broster, "A probabilistic framework for schedulability analysis," in *Proc. of the Third Int. Embedded Software Conf., EMSOFT, Lect. Notes in Computer Science*, 2003, pp. 1–15.

[9] Castelli, Harper, Heidelberger, Hunter, Trivedi, Vaidyanathan, and Zeggert, "Proactive management of software aging," in *IBM JRD*, 2001, vol. 45, pp. 311–332.

[10] M. Chin, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic systems," in *Proc. of Int. Performance and Dependability Symp.*, Washington, DC, 2002.

[11] Cortes and Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[12] J. Crowell, M. Shereshevsky, B. Cukic, V. Gandikota, and Y. Liu, "Software aging and multifractality of memory resources," presented at the Int. Conf. on Dependable Systems and Networks (DSN '03), 2003.

[13] D. E. Wybe, "Notes on structured programming," Technological University Eindhoven, Department of Mathematics, Technical Report 70-WSK-03, 1970 [Online]. Available: http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF

[14] T. Dohi, K. Goseva-Popstojanova, and K. S. Trivedi, "Analysis of software cost models with rejuvenation," in *Proc. of the IEEE Intl. Symp. on High Assurance Systems Engineering, HASE*, 2000.

[15] T. Dohi, K. Goseva-Popstojanova, and K. S. Trivedi, "Statistical nonparametric algorithms to estimate the optimal software rejuvenation schedule," in *Proc. of the Pacific Rim Int. Symp. on DependableComputing*, Los Angeles, 2000.

[16] B. Efron and B. R. Tibshirani*, An Introduction to the Bootstrap, Monographs on Statistics and Applied Probability*. : Chapman & Hall, 1993, vol. 57.

[17] M. Gardner, "Probabilistic analysis and scheduling of critical soft real-time systems," Ph.D. dissertation, , Univ. of Illinois, Urbana-Champaign, 1999.

[18] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, "Analysis of software rejuvenation using Markov regenerative stochastic Petri net," in *Proc. of the Sixth IEEE Intl. Symp. on Software Reliability Engineering*, Toulouse, France, 1995.

[19] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi, "A methodology for detection and estimation of software aging," in *Proc. of the Ninth IEEE Intl. Symp. on Software Reliability Engineering*, Paderborn, Germany, 1998.

[20] S. Garg, A. Puliofito, M. Telek, and K. S. Trivedi, "Analysis of preventive maintenance in transaction based software systems," *IEEE Trans. Computers*, vol. 47, no. 1, pp. 96–107, 1998.

[21] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1–58, 1992.

[22] M. Grottke, L. Lie, K. Vaidyanathan, and K. Trivedi, "Analysis of software aging in a web server," *IEEE Trans. Reliability*, vol. 55, no. 3, pp. 411–420, September 2006, 2006.

[23] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research 3*, pp. 1157–1182, 2003.

[24] S. Hochreiter, K. Obermayer, I. Guyon, S. Gunn, and M. Nikrav*, Nonlinear Feature Selection with the Potential Support Vector Machine, Feature Extraction, Foundations and Applications*. : Springer, 2004.

[25] G. Hoffmann, "A radial basis function approach to modeling resource demands in distributed computing systems," presented at the ISMIP—International Symposium on Multi Technology Information Processing, Taiwan, 1996.

[26] G. Hoffmann, "Failure prediction in complex computer systems: A probabilistic approach," Ph.D. dissertation, Humboldt Univ, Berlin, 2005.

[27] G. Hoffmann and M. Malek, "Call availability prediction in a telecommunication system: A data driven empirical approach," presented at the IEEE Symposium on Reliable Distributed Systems (SRDS 2006), 2006.

[28] G. Hoffmann, K. S. Trivedi, and M. Malek*, Invariant Detection for Apache Server Systems*. , NC: Duke University, 2006.

[29] G. Hoffmann and M. Malek, "Data specific mixture kernels in basis function networks," presented at the 1th Int. Conf. on Neural Information Processing Systems (ICONIP) 2004, Calcutta, 2004.

[30] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, "Software rejuvenation: Analysis, module and applications," in *Proc. of the 25th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-25)*, Pasadena, CA, 1995.

[31] Y. Huang, Chung, Wang, and Liang, "NT-SwiFT: Software implemented fault tolerance on windows NT," in *Proceedings of the 2nd USENIX Windows NT Symposium*, 1998.

[32] R. K. Iyer and D. Tang, "Experimental analysis of computer system dependability," in *Fault-Tolerant Computer System Design*, D. K. Pradhan, Ed., 2nd ed. : Prentice Hall, 1996.

[33] Kapadia, Fortes, and Brodley, "Predictive application-performance modeling in a computational grid environment," presented at the Eighth IEEE Int. Symp. on High Performance Distributed Computing, 1999.

[34] B. Littlewood and L. Strigini, "Software reliability and dependability: A roadmap," in *Proc. of the Conference on the Future of Software Engineering*, Limerick, Ireland, 2000, pp. 175–188.

[35] H. Liu and R. Setiono, "A probabilistic approach to feature selection—A filter solution," in *Proc. of Machine Learning: ICML '96*, Bari, Italy, 1996, pp. 319–327.

[36] D. J. C. MacKay, "Bayesian interpolation," *Neural Computation*, vol. 4, no. 3, pp. 415–447, 1992.

[37] F. Salfner, G. Hoffmann, and M. Malek, "Prediction-based software availability enhancement," in *Hot Topics Vol. of the Springer Lect. Notes in Comp. Science (LNCS)*, 2005.

[38] M. Malek, F. Salfner, and G. Hoffmann, "Self-rejuvenation–an effective way to high availability," presented at the Int. Workshop on Self-Star Properties in Complex Information Systems, Bertinoro (Forli), Italy, 2005, Univ. of Bologna.

[39] T. S. Parker and L. O. Chua*, Practical Numerical Algorithms for Chaotic Systems*. : Springer Verlag, 1989.

[40] Pizza, Strigini, Bondavalli, and Giandomenico, "Optimal discrimination between transient and permanent faults," presented at the 3rd IEEE High Assurance System Engineering Symposium, 1998.

[41] Rumelhart, Hinton, and Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing Volume I*, R. McCleland, Ed. : MIT Press, 1986.

[42] F. Salfner, S. Tschirpke, and M. Malek, "Comprehensive logfiles for autonomic systems," in *IEEE Proceedings of IPDPS 2004 (Int. Parallel and Distributed Processing Symp.)*, 2004.

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

[44] M. Shereshevsky, J. Crowell, and B. Cukic, "Multifractal description of resource exhaustion data in real time operating system," West Virginia Univ., 2001, Tech. Rep..

[45] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu, "Software aging and multifractality of memory resources," presented at the Int. Conf. on Dependable Systems and Networks (DSN 2003), San Francisco, 2003.

[46] T. Andres and B. Guillem*, Extracting Temporal Properties From Real-Time Systems by Automatic Tracing Analysis*. : Tech. Univ. Valencia, Real-time systems research group University of York, 2003.

[47] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. New York: John Wiley and Sons, 2001.
[48] K. S. Trivedi, K. Vaidyanathan, and K. Gosseva-Popstojanova, "Modeling and analysis of software aging and rejuvenation," in *Proc. of the 33rd Ann. Simulation Symp.*, Washington, DC, 2000, pp. 270–279.
[49] K. Vaidyanathan and K. S. Trivedi, "A comprehensive model for software rejuvenation," *IEEE Trans. Dependable and Secure Computing*, vol. 2, no. 2, pp. 124–137, April–June 2005.
[50] A. Ward and W. Whitt, "Predicting response times in processor-sharing queues," in *Proceedings of the Fields Institute Conference*, 2000, Stanford Univ., Dept. of Management Science and Engineering, AT&T Labs (Shannon Lab).
[51] A. S. Weigend and N. A. Gershenfeld, Eds., "Time series prediction," *Proceedings of the Santa Fe Institute*, vol. XV, 1994.
[52] G. M. Weiss, *Timeweaver: A Genetic Algorithm for Identifying Predictive Patterns in Sequences of Events*. : Rutgers Univ. and AT&T Labs, 1999.
[53] G. M. Weiss, *Handbook of Data Mining and Knowledge Discovery*, W. Kloesgen and J. Zytkow, Eds. : Oxford University Press, 2001.

Guenther A. Hoffmann received his Ph.D. from Humboldt University Berlin, Germany in computer science, and holds a M.Sc. degree from Technical University Berlin, Germany. His research interests cover dependable distributed systems including empirical approaches to modeling complex computing systems, failure forecasting, and self-* systems. In particular, he is working with machine learning techniques, stochastic optimization, and feature detection. He has been a guest researcher at Massachusetts Institute of Technology Artificial Intelligence Laboratory, and is currently a visiting scholar at Duke University, Durham, NC, Electrical and Computer Engineering Department.

Kishor S. Trivedi holds the Hudson Chair in the Department of Electrical and Computer Engineering at Duke University, Durham, NC. He has been on the Duke faculty since 1975. He is the author of a well-known text entitled Probability and Statistics with Reliability, Queuing, and Computer Science Applications with a thoroughly revised second edition being published by John Wiley. His research interests are in reliability and performance assessment of computer and communication systems. He has made seminal contributions in software rejuvenation, solution techniques for Markov chains, fault trees, stochastic Petri nets, and performability models. He has actively contributed to the quantification of security and survivability. He is a fellow of the IEEE.

Miroslaw Malek is professor and holder of Chair in Computer Architecture and Communication at the Department of Computer Science at Humboldt University in Berlin, Germany. His research interests focus on dependable distributed systems including failure prediction, and service availability. He authored, and co-authored over 150 publications; and founded, organized, and co-organized numerous workshops and conferences, and recently co-founded the International Service Availability Symposium. Malek received his Ph.D. in Computer Science from the Technical University of Wroclaw in Poland; spent 17 years as professor at the University of Texas at Austin; and was also, among others, visiting professor at Stanford, and guest researcher at Bell Laboratories and IBM T. J. Watson Research Center.