Predictive algorithms in the management of computer systems

by R. Vilalta
C. V. Apte
J. L. Hellerstein
S. Ma
S. M. Weiss

Predictive algorithms play a crucial role in systems management by alerting the user to potential failures. We report on three case studies dealing with the prediction of failures in computer systems: (1) long-term prediction of performance variables (e.g., disk utilization), (2) short-term prediction of abnormal behavior (e.g., threshold violations), and (3) short-term prediction of system events (e.g., router failure). Empirical results show that predictive algorithms can be successfully employed in the estimation of performance variables and the prediction of critical events.

An important characteristic of an intelligent agent is its ability to learn from previous experience in order to predict future events. The mechanization of the learning process by computer algorithms has led to vast amounts of research in the construction of predictive algorithms. In this paper, we narrow our attention to the realm of computer systems; we demonstrate how predictive algorithms enable us to anticipate the occurrence of events of interest related to system failures, such as CPU overload, threshold violations, and low response time.

Predictive algorithms can play a crucial role in systems management. The ability to predict service problems in computer networks, and to respond to those warnings by applying corrective actions, brings multiple benefits. First, detecting system failures on a few servers can prevent the spread of those failures to the entire network. For example, low response time on a server may gradually escalate to technical difficulties on all nodes attempting to communicate with that server. Second, prediction can be used to ensure continuous provision of network services through the automatic implementation of corrective actions. For example, prediction of high CPU demand on a server can initiate a process to balance the CPU load by rerouting new demands to a back-up server.

Several types of questions are often raised in the area of computer systems:

- What will be the disk utilization or CPU utilization next month (next year)?
- What will be the server workload in the next hour (n minutes)?
- Can we predict a severe system event (e.g., router failure) in the next n minutes?

The questions above differ in two main aspects: time horizon and object of prediction. The former characterizes our ability to perform short-term or long-term predictions and has a direct bearing on the kind of corrective actions one can apply. Any action requiring human intervention requires at least several hours, but if actions are automated, minutes or even seconds may suffice. The latter relates to the outcome of a prediction and can be either a numeric variable (e.g., amount of disk utilization) or a categorical event (e.g., router failure).

Both time horizon and object of prediction are important factors in deciding which predictive algorithm to use. In this paper, we present three major predictive algorithms addressing the following problems: (1) long-term prediction of performance variables (e.g., disk utilization), (2) short-term prediction of abnormal behavior (e.g., threshold violations), and (3) short-term prediction of system events (e.g., router failure). The first problem is solved using a regression-based approach. A salient characteristic of a regression algorithm is the ability to form a piecewise model of the time series that can capture patterns occurring at different points in time. The second problem employs time-series analysis to predict abnormal behavior (e.g., threshold violations); prediction is achieved through a form of hypothesis testing. The third problem predicts critical events by using data-mining techniques to search for patterns frequently occurring before these events.

Our goal in this paper is to provide some criteria in the selection of predictive algorithms. We proceed by matching problem characteristics (e.g., time horizon and object of prediction) with the right predictive algorithm. We use our selection criteria in three case studies corresponding to the problems described above.

Extensive work has been conducted in the past trying to predict computer performance. For example, work is reported in the prediction of network performance to support dynamic scheduling,[1] in the prediction of traffic network,[2] and in the production of a branch predictor to improve the performance of a deep pipelined micro-architecture.[3] Other studies reported in the literature[4–7] focus on predicting at the instruction level, whereas we focus on predicting at the system and event level (e.g., response time, CPU utilization, network node down, etc.). A common approach to performance prediction proceeds analytically, by relying on specific performance models; one example is in the study of prediction models at the source code level, which plays an important role for compiler optimization, programming environments, and debugging tools.[8] Our view of the prediction problem is mainly driven by historical data (i.e., is data-based). Many studies have tried to bridge the gap between a model-based approach versus a data-based approach.[9]

The rest of the paper is organized as follows. First we provide a general view of prediction algorithms and describe our approach to selecting an algorithm for the problem at hand. In the following section we discuss algorithms for long-term prediction of computer performance. Next we discuss an algorithm for detecting threshold violations of workload demands, and then we describe our approach to the prediction of system events. We list our conclusions in the last section.

Prediction in computer networks

We begin by giving a general view of the prediction problem. We then provide some criteria for selecting a predictive algorithm to use, based on the characteristics of the problem at hand.

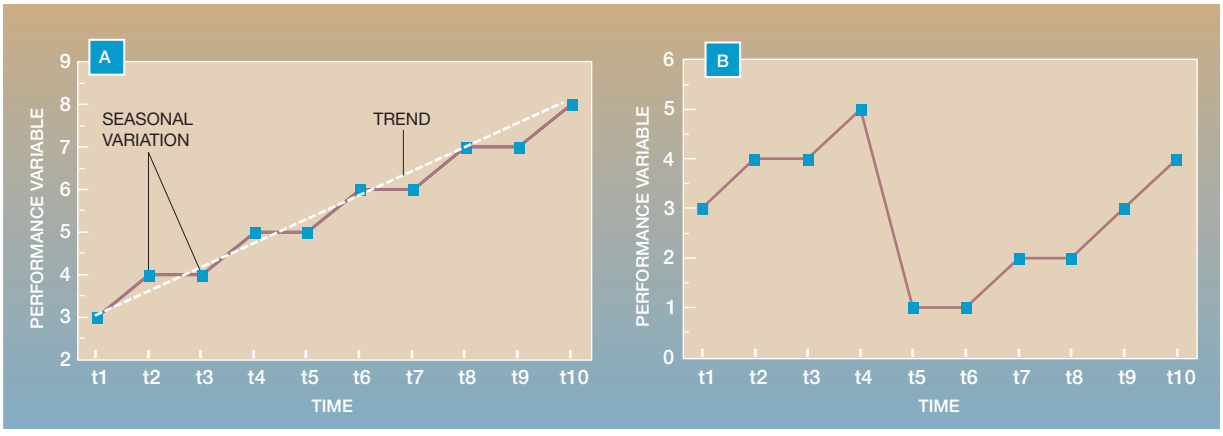
A formulation of the prediction problem. To make predictions, one needs access to historical information. We define historical information as an ordered collection of data, S_i, that starts at a point in time t_1, covering events up to a final time t_i. Specifically, $S_i = \{s_j\}$, $1 \leq j \leq i$, where the jth element is a pair $s_j = (v_j, t_j)$. The first element of the pair, v_j, indicates the value of one or more variables of interest, whereas the second element of the pair, t_j, indicates its occurrence time. The elements of S_i are ordered, that is, $t_j \leq t_k$ for any $j < k$.

As an example, assume monitoring systems capture the disk utilization on a server at five-minute intervals during an experiment of one hour. In this case, the historical information is the collection of pairs $\{(v_j, t_j)\}_{j=1}^{12}$, where v_j is the disk utilization at time t_j, and time increases in five-minute steps. In some cases we want to capture the values associated with several variables at time t_j, where the first element of each event pair is a vector $\tilde{v}_j$. For example, $\tilde{v}_j = (v_{j1}, v_{j2}, v_{j3})$, where the values on the right represent the disk utilization, the memory utilization, and the CPU utilization at time t_j, respectively. We collect data up to a point in time t_i, and the goal is to predict the disk utilization at a time t_{i+k} (i.e., k steps in the future).

A prediction is an estimation of the value of a variable v_{i+k} occurring at time t_{i+k} in the future conditioned on historical information S_i. Hence, a prediction is the output of a generic function conditioned on S_i, $v_{i+k} = g(S_i) + \epsilon_i$, in which $g(\cdot)$ is a function capturing the predictable component and ϵ_i models the possible noise.

Normally, the further out our prediction, the less accurate the result. Hence, a predicted value is ideally accompanied by a probability term that reflects our degree of confidence. This confidence can be mea-

Figure 1 A homogeneous (A) and a nonhomogeneous (B) time series



sured by a conditional probability, $P(v_{i+k}|S_i)$. Although determining the conditional probability $P(v_{i+k}|S_i)$ is always desirable, it is not always possible.

Data characteristics. Variables of interest whose values we might want to predict include the memory utilization or disk utilization on a host or group of hosts, the number of HyperText Transfer Protocol (HTTP) operations per second in a Web server, and the status of a network node (up or down) at a given time. In all these cases we rely on historical information to anticipate future behavior. We wish to emphasize the temporal component of this information: knowing when an event occurred in the past is as important as its nature.

A first step in prediction is looking for a technique matching the characteristics of the problem. Important factors are the discrete or continuous nature of the data, whether observations are taken at equal time intervals or not, and whether the data are aggregated over time intervals or correspond to instantaneous values. For example, most techniques based on time series analysis deal with discrete observations taken at equal time intervals.

We characterize historical information along two dimensions: data type and sampling frequency. Data types can be either numeric (e.g., memory utilization is 80 percent) or categorical (e.g., event type is "router failure"). The sampling mechanism depends on the data collecting method and is either periodic sampling (i.e., equal time intervals) or triggered sampling (i.e., data collected when a predefined condition is satisfied). Data collected by periodic sampling include performance measurements such as utilization and end-to-end response time to a probing agent (e.g., "ping" and mail server probe).

Prediction techniques. Once the problem is well characterized, there are often a wide variety of prediction techniques available. In some cases we rely on classical time-series analysis, whereas in other cases we employ data-mining techniques. An important factor that differentiates among techniques is whether or not the model is homogeneous through time. A homogeneous model captures key characteristics of the time series such as the general trend, seasonal variation, and variation in the stationary residuals.[10] Figure 1A shows the general trend and seasonal variation on a time series. The general trend could correspond to a constant rate of increase of the CPU utilization on a server over months, whereas the seasonal variation could reflect some, say, monthly activity particular to the customer. If one were to remove these variations from the data, the result would be a stationary time series (as explained later in this paper). When these variations are present, the general assumption is that they persist throughout the entire time interval.

We also consider the case where key characteristics on the time series vary significantly depending on the time and the state of the system being modeled. Figure 1B shows a scenario where both trend and seasonal effects are not constant through time. It is under these conditions that using new techniques can

Table 1 Selecting a predictive algorithm based on domain properties

	Short-Term Predictions	Long-Term Predictions
Numeric Data	Stationary models for time series	Trend and seasonal analysis
Boolean Data	Data mining	Periodicity analysis failure model

add flexibility to the prediction process, and as we show later, this flexibility often results in improved accuracy.

Selection criteria. Selecting the right predictive algorithm depends on at least two factors: the time of the prediction and the type of data. The first factor can be divided into short-term prediction and long-term prediction. A difficulty inherent to this differentiation is to ascribe a precise meaning to both terms. In the context of computer systems, it is reasonable to assume short-term prediction in the range of minutes or hours, and long-term prediction in the range of days, weeks, or months. The second factor can be either numeric or Boolean. In some cases it is also important to note if the observations were made at equal time intervals or not.

Table 1 presents our selection guidelines for a prediction technique based on the factors above. Long-term predictions of numeric data need to consider the general trend and seasonal variations. The general trend measures the long-term change in the mean level, whereas seasonal variations are normally the result of long-term fluctuations. Trend and seasonal variations normally account for most of the long-term behavior of a time series. We exemplify this case in the next section.

Short-term predictions of numeric variables are attained by applying classical time-series analysis over stationary data. The data obtained by removing the general trend (or mean) and the seasonal variations are usually stationary (no systematic change is detected). For equally spaced sampling of data, one can use models such as autoregressive processes and moving averages.[10] We exemplify this case in the section "Predicting threshold violations."

On the other hand, short-term predictions of categorical variables (not necessarily from equally spaced data) are attainable by relying on data-mining techniques. Recent years have seen an explosion in the study of data-mining techniques looking for different forms of temporal patterns.[11–14] A common technique is to find frequent subsequences of events in the data. An additional step, however, is needed to integrate these patterns into a model for prediction.[15–18] We exemplify this case in the section "Predicting target events in production networks."

Our last scenario deals with long-term predictions of Boolean data, for which there are two different methods available: use of periodic patterns and use of failure models. Periodic patterns are represented as sets of events occurring at regular intervals of time.[19] In computer networks, for example, a periodic pattern may correspond to high CPU utilization on several servers at regular time intervals due to scheduled maintenance. Periodic patterns may be removed if they reflect normal behavior, or used for prediction if signaling uncommon situations. Failure models have been used to model device life times.[20] Modeling involves a mathematical equation (e.g., Poisson lift span, independent failures) that must capture the true data distribution.

The next three sections describe real applications that exemplify our choice of predictive algorithms. We focus on the first three cases described above in the context of computer systems.

Predicting computer performance

Our first study deals with the problem of long-term predictions on numeric data (Table 1). We wish to predict performance parameters, such as response time or disk utilization, for capacity planning.[21] Estimating the future value of a performance parameter is helpful in assessing the need to acquire additional devices (e.g., extra memory or disk space) to ensure continuity in all network services. Here we focus on the nonstationary components of the time series. In particular, we look at the general trend. We overview two different approaches: the traditional k-step extrapolation, and learning to map k-steps ahead. We look at each approach in turn.

Extrapolation. A familiar approach to the prediction of the general trend is to fit a model to the data and then to extrapolate k steps ahead in time. For example, a simple technique is to assume the data follow a linear trend plus noise of the form

$$v_j = \beta_1 + \beta_2 t_j + \epsilon_i, \qquad 1 \le j \le i \tag{1}$$

where β_1 and β_2 are constants and ϵ_j is a random variable with zero mean. Prediction is simply a matter of projecting Equation 1 to find the value of v_{i+k}. For example, applying a linear least-squares regression over disk utilization vs time on a computer server where data are aggregated on a monthly basis can indicate a critical threshold will be reached in approximately five months. In some cases we may find we can do a better job by fitting the data using polynomial curves.

Learning to predict *k* steps ahead. A different perspective to the prediction problem uses concepts from machine learning. Instead of fitting a model to our historical data we could try to *learn* a mapping between the state of a computer at time t_j and the state of the computer at time t_{j+k}. The mapping can be described by the following equation:

$$v_{j+k} = g(v_j) + \epsilon_j \tag{2}$$

In other words, we try to learn how to estimate the value of the performance variable k steps in the future by creating a set of pairs $\{(v_j, v_{j+k})\}$, using our historical data (i.e., match each measurement with the measurement k steps ahead). The problem is now transformed into that of function approximation: we want to approximate the function that generated these points. Learning this mapping gives a direct model for prediction, which we can use to estimate v_{i+k}, where v_i is the last observation in S_i. The nature of g can take on different forms: it can be represented as a linear function of v_j, as a decision tree, a neural network, etc. We do not restrict the nature of function g to a specific form, but simply indicate its functionality: to map computer states from time t_j to time t_{j+k}.

Approximating function g obviates any form of extrapolation. The difference with extrapolation is that in this formulation we need to approximate a different function g for each value of k. The advantage lies on the flexibility imbued in the model: it enables us to deal with time series where key characteristics may vary significantly through time (see the subsection "Prediction techniques," earlier).

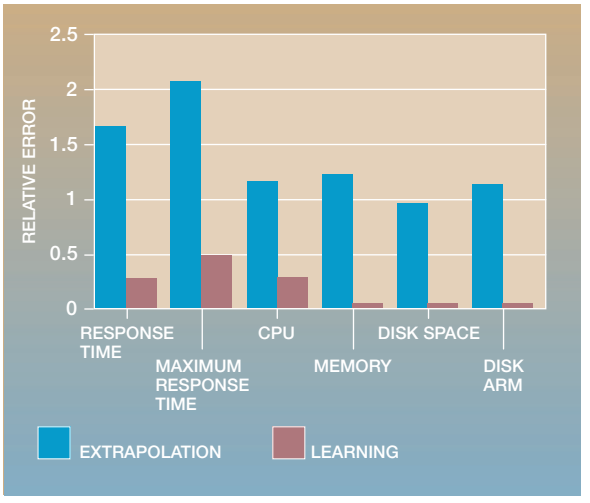
The general approach to learn to perform predictions is to transform the original database to reflect the mapping between a state at time t_j and a state at time t_{j+k}. The idea is to cast the prediction problem into a learning problem. In a learning problem, the input data are normally represented in tabular form, where each record represents an example characterized by features, and the last column is the target class to which the example belongs. A numeric class calls for regression methods (as in our case), whereas a categorical or nominal class calls for classification methods. The goal is to learn how to map feature values into class values in order to *predict* the class of new examples.[22,23]

Returning to the problem of predicting computer performance, remember our historical data are an ordered collection of events. Each event can play the role of an example characterized by one or multiple performance variables at time t_j. The target class of the example corresponds to the value of the predictive variable at time t_{j+k} (in order to learn to predict the value of the performance variable k steps in the future).

Empirical findings. Different algorithms can be used to learn the mapping mentioned above, including linear and nonlinear regression methods and decision trees for regression. Our experiments using these techniques reveal two interesting findings. First, casting the prediction problem into a learning problem yields significant gains in accuracy compared to traditional techniques. Our conclusions come from experiments on a central database that contains information on the performance of thousands of IBM AS/400* computers. Each record in the database reports the values of tens of performance parameters for a particular machine and month of the year. We form predictions for six important parameters: response time, maximum response time, CPU utilization, memory utilization, disk utilization, and disk arm utilization. Figure 2 compares a multivariate linear regression model using the learning approach vs the extrapolation approach. We measure relative error defined as the ratio between the error of the multivariate linear model and the error of a simple baseline model that takes the mean of all past values to predict future values. For all six performance variables under study, applying the model on the learning approach yields significant gains in accuracy (Figure 2).

A second interesting finding is that learning from data extracted from multiple computers of similar architecture yields better accuracy than learning from data extracted from a single computer. It is reasonable to suppose that computers having similar architecture will experience similar performance if the overall utilization is the same. Hence, looking for patterns across computers increases the evidential

Figure 2 A comparison of multivariate linear regression models using the learning approach vs the extrapolation approach



support for correlations between performance variables and target variables. As an example, assume CPU utilization is a function of the memory size. Data from a single computer may show evidence of a positive correlation, but with high variance due to the limited number of points available. In contrast, data from multiple computers enables us to increase our confidence in the quality of the model and therefore in our predictions.

Predicting threshold violations

Our second study deals with the problem of short-term predictions on numeric data (Table 1). We assume the existence of Internet-attached servers that have time-varying workloads. We describe a systematic, statistical approach to the characterization of normal operation for time-varying workloads, and we apply this approach to problem detection for a Web server by predicting threshold violations.[24]

We show how our method can be used to construct a predictive model for the purposes of workload forecasting. We begin by developing a statistical model of the time-varying behavior of the data and then remove the nonstationary components. This problem differs significantly from the study in our previous section. Here we assume a performance line centered around a constant value (μ). Our goal is not to predict the general trend, but to detect when a deviation from the trend is extreme. We do this by first removing mean and seasonal effects (i.e., the nonstationary components). We then look to the residuals in search of abnormal behavior.

Removing mean and seasonal effects. The data we consider span a time interval of eight months (June 1996 through January 1997) from a production Web server at a large corporation. Data are aggregated over five-minute intervals. The variable of interest is HTTP operations per second (httpops), since this is an overall indicator of the demands placed on the Web server.

We begin by considering the effect of time of day. Let v_{jd} be the value of httpops for the j th five-minute interval (time-of-day value) and the d th day in the data collected. Figure 3A plots v_{jd} for a work week (Monday through Friday) in June of 1996 and a work week in November of 1996. The x-axis is time, and the y-axis is httpops.

We partition v_{jd} into three components: the grand mean, the deviation from the mean due to the j th time-of-day value (e.g., 9:05 A.M.), and a random error term that captures daily variability. The grand mean is denoted by μ. The j th time-of-day deviation from the grand mean is denoted by α_j (note that $\Sigma_j\ \alpha_j = 0$). The error term is denoted by ϵ_{jd}.

The model is:

$$v_{jd} = \mu + \alpha_j + \epsilon_{jd} \tag{3}$$

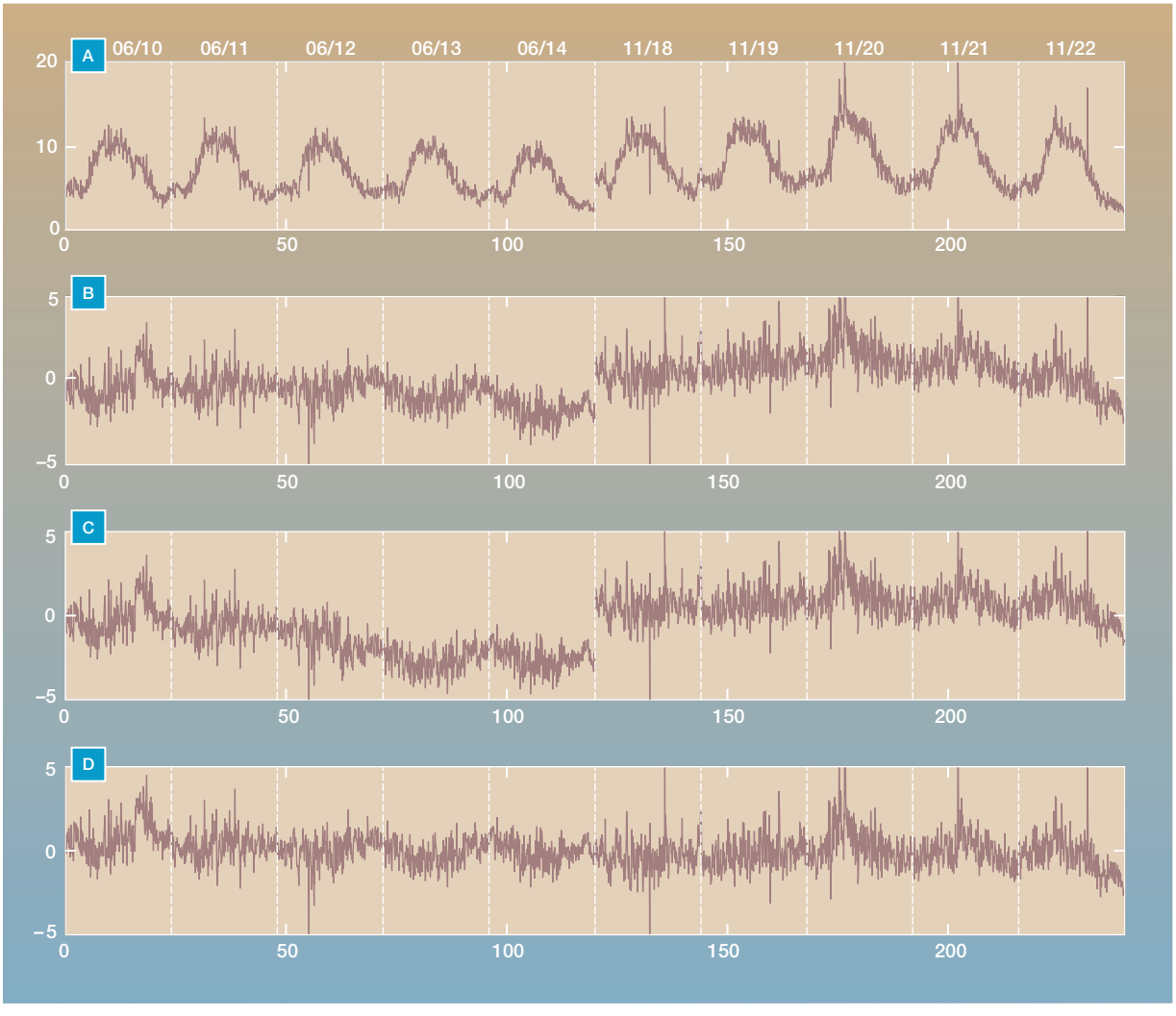
We use the residuals to look for more patterns in the data after time-of-day effects have been removed. Figure 3B plots the residuals for Equation 3. Observe that much of the rise in the middle of the day (as evidenced in Figure 3A) has been removed.

A further examination of Figure 3B indicates that there is a weekly pattern. Let β_w denote the effect of the w th day of the work week. As with α, this is a deviation from the grand mean (μ). Thus, $\Sigma_w\ \beta_w = 0$. Our extended model is:

$$v_{jdw} = \mu + \alpha_j + \beta_w + \epsilon_{jdw} \tag{4}$$

Note that since we include another parameter (day of week), another subscript is required for both v and ϵ. The residuals of this model are plotted in Figure 3C.

Figure 3 Rate of HTTP transactions vs time for a Web server



Reprinted from *Computer Networks,* Vol. 35, No. 1, J. L. Hellerstein, F. Zhang, and P. Shahabuddin, "A Statistical Approach to Predictive Detection," pp. 77-95, Figure 2, (c) 2001, with permission from Elsevier Science.

Looking further, we observe that another pattern remains: httpops is larger in November than it is in June. To eliminate this, we extend our model to consider the month. Let γ_m denote the effect of the mth month. As with α and β, $\Sigma_m\ \gamma_m = 0$. The model here is:

$$v_{jdwm} = \mu + \alpha_j + \beta_w + \gamma_m + \epsilon_{jdwm} \qquad (5)$$

Once again, another subscript is added to both v and ϵ.

An autoregressive model. Until now we have been able to account for the mean, daily, weekly, and monthly effects. Figure 3D still has time serial dependencies. To remove these dependencies, we extend the characterization in Equation 5. We assume that the time index time t can be expressed as a function of (j, d, w, m). Then, we consider the following model:

$$\epsilon_t = \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + u_t \qquad (6)$$

This is a second-order autoregressive model (AR(2)). Here, θ_1 and θ_2 are parameters of the model (which are estimated from the data), and the u_t are independent and identically distributed random variables. The model parameters are estimated using standard techniques.[25]

Now consider the prediction of threshold violations. Current practice for problem detection is to establish threshold values for measurement values. If the observed value violates its threshold, an alarm is raised. Threshold values are obtained from historical data, such as the 95th quantile.[26]

Unfortunately, there is a significant difficulty with this approach in practice: normal load fluctuations are so great that a single threshold is inadequate. That is, a single threshold either results in an excessive number of false alarms, or the threshold fails to raise an alarm when a problem occurs. Some performance management products attempt to overcome this difficulty by allowing installations to specify different thresholds at different times of the day, day of the week, etc. However, requiring that installations supply additional thresholds greatly adds to the burden of managing these installations.

Prediction using change-point detection. We propose here an approach in which we use the characterization model to remove all known patterns in the measurement data, including the time-serial dependencies. For httpops in the Web server data, this means using Equation 5 to remove "low frequency" behavior, and then applying Equation 6 to the residuals of this equation so as to remove time-serial dependencies. The residuals of Equation 6 constitute filtered data for which all patterns in the characterization have been removed. Last, a change-point detection algorithm is applied to these filtered data to detect anomalies, such as an increase in the mean or the variance.

There are many algorithms for change-point detection.[27] Herein, we use the GLR (Generalized Likelihood Ratio) algorithm. This is an on-line technique that examines observations in sequence rather than in mass. When a change has been detected, an alarm is raised.

First, we introduce some terminology. Recall that u_t is the tth residual obtained by filtering the raw data using a characterization such as Equations 5 and 6. We consider two time windows, that is, a set of time indexes at which data are obtained. The first is the *reference window*; values in this window are used to estimate parameters of the "null hypothesis" in the test for a change point. The reference window starts with the time at which the last change point was detected; it continues through the current time (t). Within the reference window, u_t has variance σ_u^2. The second time window is the *test window*. Values in this window are used to estimate parameters of the "alternative hypothesis" that a change point has occurred. The test window spans $t - L$ through t. L should be large enough to get a stable estimate of $\sigma_{u'}^2$ (the variance of u_t in the test window), but small enough so that change points are readily detected.

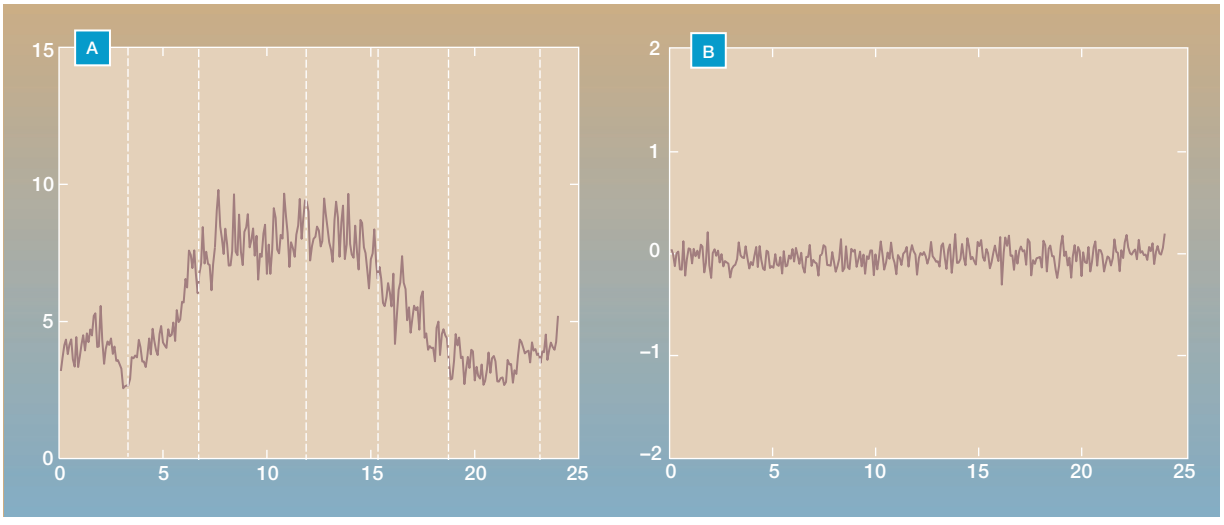
Empirical findings. We apply the foregoing approach to the Web server data collected on July 15, 1996, a day for which no anomaly is apparent. Figure 4A displays httpops for this day. The vertical lines indicate where change points are detected using the GLR algorithm. Note that not taking into account normal load fluctuations, as is often done in practice, would have resulted in six alarms even though no problem is apparent. Figure 4B plots the residuals after using Equation 5 to filter the raw data and Equation 6 to filter the residuals produced by it. Observe that the GLR algorithm does not detect any change point. Hence, taking normal behavior into account enables our algorithm to reduce the number of false alarms to only those cases where abnormal deviations from the mean are authentic.

Predicting target events in production networks

Our third study deals with the problem of short-term predictions on categorical data (Table 1). The prediction targets are computer-network events, corresponding either to single hosts (e.g., CPU utilization is above a critical threshold), or to a network (e.g., communication link is down). Monitoring systems capture thousands of events in short time periods; data analysis techniques may reveal useful patterns characterizing network problems.[28]

The task of predicting target events across a computer network exhibits characteristics different than the problems discussed in the last two sections (predicting performance variables or predicting threshold violations). A prediction here is an estimation of a categorical or nominal value in the near future (e.g., communication link will be down within five minutes).

Figure 4 Change-points in data: (A) raw data, and (B) data after filtering

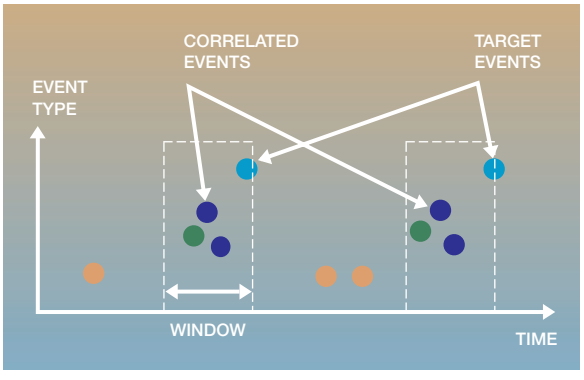


Target events and correlated events. We let the user specify what events are of interest. For example, a system administrator may wish to understand what causes a printer error, or why a particular server is not responding within a specified time threshold. We refer to these events, such as all occurrences of a printer error, as the set of *target events*.[29,30] We assume the proportion of target events with respect to all events is low; target events do not represent a *global* property, such as periodicity or constant trend, but rather a *local* property, such as a computer attack on a host network.

Figure 5 shows the idea behind our predictive algorithm. We look at those events occurring within a time window of size W (user-defined) before a target event. We are interested in finding sets of event types, referred to from now on as *eventsets*, frequently occurring before a target event. A solution to the problem above is important to many real applications. Understanding the conditions preceding a system failure may pinpoint its cause. On the other hand, anticipating a system failure enables us to apply corrective actions before the failure actually occurs. For example, an attack on a computer network may be characterized by an infrequent but highly correlated subsequence of events preceding the attack.

Technical approach. The problem of finding meaningful eventsets preceding the occurrence of target events, which we then use to build a model for prediction, can be divided in three steps: (1) use associations to find frequent eventsets within the time windows preceding target events; (2) validate those eventsets against events outside the time windows considered in step 1; (3) build a rule-based model for prediction. We explain each step next.

Figure 5 Target events and correlated events



Finding frequent eventsets. Our first step makes use of mining of frequent itemsets as follows. Consider a single target event e^*. The conditions preceding e^* can be characterized by simply recording the event types within a window of size W. For example, if each target event is preceded by four different events

within a window of size W, then each window can be represented as an event transaction T made of four event types (e.g., $T = \{e_1, e_2, e_3, e_4\}$). Note that it is admissible for consecutive target events to generate time windows that overlap.

The procedure above can be applied over all occurrences of target events to generate a set of event transactions D. More specifically, our algorithm makes one pass through the sequence of events in D, which we assume to be in increasing order along time. With each new event, the current time is updated; the algorithm keeps in memory only those events within a time window of size W from the current time. If the current event is a target event, the set of event types contained in the most recent time window become a new transaction in D. Finally, we use association-rule mining[31] to find large eventsets, that is, eventsets with frequency above minimum support (e.g., *a priori* algorithm). Our work is in some sense related to the area of sequential mining,[11–14] in which traditional association mining is extended to search for frequent subsequences.

Note that the ordering of events and the interarrival time between events within each time window is not relevant. This is useful when an eventset occurs under different permutations, and when interarrival times exhibit high variation (i.e., signals are noisy). These characteristics are present in many domains, including the real production network used for our experiments. For example, we observed that a printer-network problem may generate a set of events under different permutations and with interarrival-time variation in the order of seconds. Our approach to overcome these uncertainties is to collect all event types falling inside the time windows preceding target events, which can then be simply treated as database transactions.

Validating eventsets or patterns. For a target event such as "host A is down," an example of an eventset Z frequently occurring before the target event is "low response time and high CPU utilization." We refer to Z as a pattern. We may associate a pattern Z with the occurrence of the target event if Z does not occur frequently outside the time windows preceding target events. Otherwise Z would appear as the result of background noise, or of some global property of the whole event sequence. For example, if "low response time" is constant through time, it cannot be used for prediction.

We start by computing the confidence of each eventset or pattern, filtering out those below a minimum degree of confidence. Confidence is an estimation of the conditional probability of Z belonging to a time window that precedes a target event, given that Z matches the event types in that same time window. Specifically, if D is the database capturing all eventsets preceding target events, then let D' be defined as the complement database capturing all eventsets occurring within time windows of size W not preceding target events. Let x_1 and x_2 be the number of transactions in D and D', respectively, matched by eventset Z. We eliminate all Z below a minimum confidence level, where confidence is defined as follows:

$$\text{confidence}(Z, B, B') = x_1/(x_1 + x_2) \tag{7}$$

In addition, our filtering mechanism performs one more test to validate an eventset. The reason is that confidence alone is not sufficient to guarantee that the probability of finding an eventset Z within database D is significantly higher than the corresponding probability in D'; confidence does not check for negative correlations.[32] Thus, we add a validation step described as follows.

Let $P(Z|D)$ denote the probability of Z occurring within database D, and $P(Z|D')$ the corresponding probability within D'. Eventset Z is validated if we can reject the null hypothesis

$$H_0: P(Z|D) = P(Z|D') \tag{8}$$

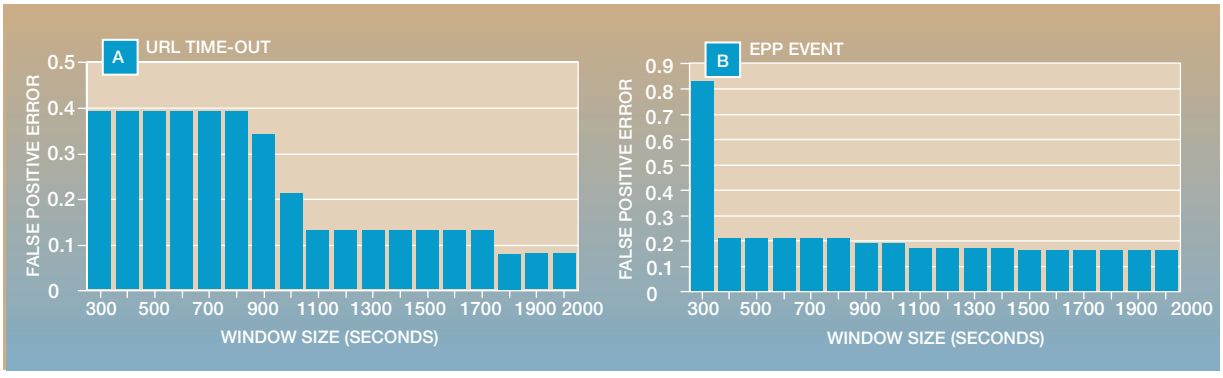
with high confidence. If the number of events is large, one can assume a Gaussian distribution and reject the null hypothesis in favor of the alternative hypothesis

$$H_1: P(Z|D) > P(Z|D') \tag{9}$$

if for a given confidence level α the difference between the two probabilities (normalized to obtain a standard normal variate) is greater by z_α standard deviations. In such case we reject H_0. The probability of this happening when H_0 is actually true is α. By choosing a small α we can be almost certain that Z is related to the occurrence of target events.

In summary, our validation phase ensures that the probability of an eventset Z appearing before a target event is significantly larger than the probability of Z not appearing before target events. The vali-

Figure 6 Error of rule-based model vs size of window preceding target events



dation phase discards any negative correlation between Z and the occurrence of target events. In addition, this phase serves as a filtering step to reduce the number of candidate patterns used to build a rule-based model for prediction.

Rule-based system for prediction. Our last step uses the set of validated patterns to build a rule-based system for prediction. Previous work exists combining associations with classification.[15–18] This work differs in the temporal nature of the data, and in the nature of the rule-based system.

The rationale behind our rule-based system is to find the most accurate and specific rules first.[23] Our assumption of having a large number of available eventsets and few target events obviates ensuring each example is covered by a rule. Specifically, our algorithm sorts all eventsets according to confidence (ties are resolved by larger frequency and larger size). In general, other metrics can be used to replace confidence,[33] such as information gain, gini, or χ^2. Starting with the highest-confidence eventset Z_i, we eliminate all other eventsets more general than Z_i. Eventset Z_i is said to be more general than eventset Z_j, if $Z_i \subset Z_j$. For example, eventset $\{a, b\}$ is more general than eventset $\{a, b, c\}$. This step eliminates eventsets that refer to the same pattern as Z_i but are more general. The resulting rule is of the form $Z_i \rightarrow$ targetevent. The search then continues with the next highest-confidence eventset, until no more eventsets are left.

The final rule-based system $\mathcal{R}$ can be used for prediction by checking for the occurrence of any of the eventsets in $\mathcal{R}$ along the event sequence set apart for testing. The model predicts finding a target event within a time window of size W after any such eventset is detected.

Empirical findings. We report results obtained from a production computer network. Data were obtained by monitoring systems active during one month on a network having 750 hosts. One month of continuous monitoring generated over 26000 events, with 165 different types of events. All events serve as input to the system. Our analysis concentrates on two types of target events labeled as *critical* by domain experts. The first type, URL (uniform resource locator) Time-Out, indicates a Web site is unaccessible. The second type, EPP (end-to-end probing platform) Event, indicates that end-to-end response time to a host generated by a probing mechanism is above a critical threshold.

The first 50 percent of events serve for training and the other 50 percent serve for testing. Error is computed on the testing set only as follows. Starting at the beginning of the sequence, nonoverlapping time windows of size W that do not intersect the set of time windows preceding target events are considered negative examples; all time windows preceding target events are considered positive examples. Error is defined as the fraction of examples incorrectly classified by the rule-based model.

We investigate the effect of varying the time window preceding target events on the error of the rule-based model. In all our experiments, the error corresponding to false positives is small (<0.1) and does not vary significantly while increasing the time windows. We focus on the false negative error rate defined as

the proportion of times the rule-based model fails to predict a true target event. Figure 6A shows our results when the target event corresponds to URL Time-Out on a particular host. With a time window of 300 seconds, the error is 0.39 (9/23). But as the time window increases, the error decreases significantly. Evidently, larger time windows enable us to capture more information preceding target events. Figure 6B shows our results with a different target event: EPP Event on a particular host. With a time window of 300 seconds the error is as high as 0.83 (9/62). Increasing the window to 2000 seconds brings the error rate down to 0.16. Our results highlight the importance of the size of the time window preceding target events in order to capture relevant patterns.

We also investigate the effect of having a *warning window* before each target event in case the rule-based model were used in a real-time scenario with a need for corrective actions to take place. In this case, the algorithm does not capture any events within the warning window while characterizing the conditions preceding target events. Our results show a degradation of performance when the safe window is incorporated, albeit to a small degree. On the EPP Event, for example, a time window of 300 seconds and a safe window of 60 seconds produces the same amount of error as when the safe window is omitted.

Conclusions

In this study of predictive algorithms, we establish a distinction between short- and long-term predictions and between numeric and categorical data. We describe three case studies corresponding to the following scenarios: (1) long-term prediction of performance variables, (2) short-term prediction of abnormal behavior, and (3) short-term prediction of system events. Empirical results show how predictive algorithms can be successfully employed in the estimation of performance variables and critical events.

Future work will look at possible ways to unify the mechanism behind predictive algorithms to enrich our understanding of their applicability. For example, we note that problems characterized by numerical data can be converted into categorical data and vice versa. Aggregating events over fixed time intervals converts categorical data into numerical data. For example, workload in a server is computed by aggregating the number of site requests over fixed time units. Conversely, thresholding can be used to transform numbers into categories. For example, a measure of the end-to-end response time of a service request such as ping or mail probe, is often categorized as normal or abnormal, depending on whether the end-to-end response time exceeds a predefined threshold.

The transformation above can extend the applicability of predictive algorithms. For example, aggregating the number of times a host is down over fixed time intervals enables us to analyze the trend and seasonal variation of the host-down frequency. Since there may be occasions in which bringing a host down is part of scheduled maintenance, the same transformation can be used to detect if host-down frequency falls into abnormal behavior. In the example above, all three predictive algorithms described in previous sections can play an important role. Our goal is to develop tools to transform the input data so as to enable the use of different predictive algorithms. The result would increase the amount of information necessary to determine the root cause of a problem and the amount of evidence to perform accurate predictions.

Acknowledgments

We are grateful to IBM Rochester, Minnesota, IBM IGS, and in particular to Teiji Kimbal and Herb Lee, for kindly providing the data for our case studies. We are equally grateful to the anonymous reviewers for their excellent suggestions. This work was supported by the IBM Thomas J. Watson Research Center.

Cited references

1. R. Wolski, "Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service," *Proceedings of the Sixth IEEE International Symposium on High Performance Distributed Computing*, IEEE, New York (1997).
2. K. C. Claffy, H.-W. Braun, and G. C. Polyzos, "Tracking Long-term Growth of the NSFNET," *Communications of the ACM* **40**, No. 11, 34–45 (1994).
3. T.-Y. Yeh and Y. N. Patt, "Alternative Implementation of Two-Level Adaptive Branch Predictions," *Proceedings of the 19th Annual International Symposium on Computer Architecture*, Gold Coast, Australia, 1992, ACM, New York (1992), pp. 124–134.
4. B. Calder, D. Grunwald, and J. Emer, "A System Level Perspective on Branch Architecture Performance," *Proceedings of the 28th Annual IEEE/ACM International Symposium on Microarchitecture*, Ann Arbor, MI, 1995, IEEE, New York (1995), pp. 199–206.

5. B. Calder and D. Grunwald, "Next Cache Line and Set Prediction," *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, ACM, New York (1995), pp. 287–296.
6. N. P. Jouppi and P. Ranganathan, "The Relative Importance of Memory Latency, Bandwidth, and Branch Limits to Performance," *Workshop on Mixing Logic and DRAM: Chips that Compute and Remember* (1997).
7. Z. Xu, X. Zhang, and L. Sun, *Semi-Empirical Multiprocessor Performance Predictions*, TR-96-05-01, University of Texas, San Antonio, High Performance Computation and Software Lab (1996).
8. C.-H. Hsu and U. Kremer, *A Framework for Qualitative Performance Prediction*, Technical Report LCSR-TR98-363, Department of Computer Science, Rutgers University (1998).
9. M. E. Crovella and T. J. LeBlanc, "Parallel Performance Prediction Using Lost Cycles Analysis," *Proceedings of Supercomputing 94*, IEEE, New York (1994), pp. 600–609.
10. C. Chatfield, *The Analysis of Time Series: An Introduction*, Chapman & Hall/CRC Press (1975).
11. R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proceedings of the 11th International Conference on Data Engineering*, IEEE, New York (1995), pp. 3–14.
12. H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering Frequent Episodes in Sequences," *Proceedings of the First International Conference on Knowledge Discovery and Data Mining* (KDD'95), Montreal, Canada, 1995, AAAI Press, Menlo Park, CA (1995), pp. 210–215.
13. R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," *Proceedings of the 5th International Conference on Extending Database Technology*, Springer-Verlag (1996), pp. 3–17.
14. M. J. Zaki, "Sequence Mining in Categorical Domains: Algorithms and Applications," *Sequence Learning: Paradigms, Algorithms, and Applications*, R. Sun and G. L. Giles, Editors, *Lecture Notes in Artificial Intelligence*, Vol. 1828, Springer-Verlag (2001), pp. 163–187.
15. K. Ali, S. Manganaris, and R. Srikant, "Partial Classification Using Association Rules," *Proceedings of the 3rd International Conference on Knowledge Discovery in Databases and Data Mining*, AAAI Press, Menlo Park, CA (1997), pp. 115–118.
16. R. J. Bayardo, "Brute-Force Mining of High Confidence Classification Rules," *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining* (KDD'97), AAAI Press, Menlo Park, CA (1997), pp. 123–126.
17. D. Meretakis and B. Wuthrich, "Classification as Mining and Use of Labeled Itemsets," *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery* (DMKD), Philadelphia, 1999, ACM, New York (1999).
18. W. Pijls and R. Potharst, "Classification and Target Group Selection Based upon Frequent Patterns," *Proceedings of the Twelfth Belgium-Netherlands Artificial Intelligence Conference* (BNAIC'00), (2000), pp. 125–132.
19. S. Ma and J. Hellerstein, "Mining Partially Periodic Event Patterns," *Proceedings of the International Conference on Data Engineering* (ICDE), IEEE, New York (2001), pp. 205–214.
20. R. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing*, Holt Rinehart and Winston, New York (1975).
21. R. Vilalta, C. Apte, and S. Weiss, "Operational Data Analysis: Improved Predictions Using Multi-Computer Pattern Detection," *Proceedings of the 11th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Austin, TX, 2000, Springer-Verlag, *Lecture Notes in Computer Science*, Vol. 1960 (2000), pp. 37–46.
22. S. M. Weiss and C. A. Kulikowski, *Computer Systems That Learn*, Morgan Kaufmann Publishers, San Mateo, CA (1991).
23. R. S. Michalski, "A Theory and Methodology of Inductive Learning," *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. Carbonell, and T. Mitchell, Editors, TIOGA Publishing Co., Palo Alto, CA (1983), pp. 83–134.
24. J. Hellerstein, F. Zhang, and P. Shahabuddin, "A Statistical Approach to Predictive Detection," *Computer Networks* **35**, No. 1, 77–95 (2001).
25. G. E. P. Box and G. M. Jenkins, *Time Series Analysis Forecasting and Control*, Prentice Hall, Englewood Cliffs, NJ (1976).
26. P. Hoogenboom and J. Lepreau, "Computer System Performance Problem Detection Using Time Series Model," *Proceedings of INFOCOM*, Kobe, Japan, IEEE, New York (1997).
27. M. Basseville and I. Nikiforov, *Detection of Abrupt Changes: Theory and Applications*, Prentice Hall, Englewood Cliffs, NJ (1993).
28. R. Vilalta and S. Ma, *Predicting Rare Events in Temporal Domains Using Associative Classification Rules*, Technical Report, IBM Research, T. J. Watson Research Center, Yorktown Heights, NY (2002).
29. G. Weiss and H. Hirsh, "Learning to Predict Rare Events in Event Sequences," *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, CA (1998), pp. 359–363.
30. R. Vilalta, S. Ma, and J. Hellerstein, "Rule Induction of Computer Events," *Proceedings of the 12th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Springer-Verlag, *Lecture Notes in Computer Science* (2001).
31. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast Discovery of Association Rules," *Advances in Knowledge Discovery and Data Mining*, U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Editors, AAAI Press, Menlo Park, CA (1996), pp. 307–328.
32. S. Brin, R. Motwani, and C. Silverstein, "Beyond Market Baskets: Generalizing Association Rules to Correlations," *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining* (KDD'97), AAAI Press, Menlo Park, CA (1997), pp. 265–276.
33. A. P. White and W. Z. Liu, "Bias in Information-Based Measures in Decision Tree Induction," *Machine Learning* **15**, No. 3, 321–329 (1994).

Accepted for publication March 28, 2002.

Ricardo Vilalta *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York, 10598 (electronic mail: vilalta@us.ibm.com).* Dr. Vilalta received his Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1998. His interests lie in machine learning, pattern recognition, neural nets, data mining, and artificial intelligence. His research is centered on applying meta-knowledge to improve the performance of learning algorithms. His current work at IBM involves the development of data analysis techniques for computer-problem determination.

Chidanand V. Apte *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: apte@us.ibm.com).* Dr. Apte manages the Data Abstraction Research group at the Thomas J. Watson Research Center. He received his Ph.D. degree in computer science from Rutgers University in 1984. His research interests include knowl-

edge discovery and data mining, applied machine learning and statistical modeling, and business intelligence automation.

Joseph L. Hellerstein *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: hellers@us.ibm.com).* Dr. Hellerstein manages the Systems Management Research Department with projects such as event mining, event prediction, intelligent probing, automated diagnosis, and generic adaptive control. He received his Ph.D. degree in computer science from the University of California at Los Angeles in 1984. He has taught at Columbia University and has published approximately 70 papers.

Sheng Ma *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: shengma@us.ibm.com).* Dr. Ma received his B.S. degree in electrical engineering from Tsinghua University, China, in 1992. He received M.S. and Ph.D. (with honors) degrees in electrical engineering from Rensselaer Polytechnic Institute in 1995 and 1998, respectively. He joined the Thomas J. Watson Research Center as a research staff member in 1998, where he is now manager of the Machine Learning for Systems Department. His current research interests include network and computer system management, machine learning, data mining, and network traffic modeling and control.

Sholom M. Weiss *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: sholom@us.ibm.com).* Dr. Weiss is a research staff member at the Thomas J. Watson Research Center. Prior to joining IBM, he was a professor of computer science at Rutgers University. He is an author or coauthor of numerous papers on artificial intelligence and machine learning, including *Predictive Data Mining: A Practical Guide*, published by Morgan Kaufmann Publishers in 1997. His current research focuses on innovative methods in data mining. He is a Fellow of the American Association for Artificial Intelligence and serves on numerous editorial boards.