Automated and Adaptive Threshold Setting: Enabling Technology for Autonomy and Self-Management

David Breitgand, Ealan Henis, Onn Shehory
IBM Haifa Research Lab
Haifa University Campus, Mount Carmel,
Haifa, Israel 31905
{davidbr,ealan,onn}@il.ibm.com

Abstract

Threshold violations reported for system components signal undesirable conditions in the system. In complex computer systems, characterized by dynamically changing workload patterns and evolving business goals, the pre-computed performance thresholds on the operational values of performance metrics of individual system components are not available. This paper focuses on a fundamental enabling technology for performance management: automatic computation and adaptation of statistically meaningful performance thresholds for system components. We formally define the problem of adaptive threshold setting with controllable accuracy of the thresholds and propose a novel algorithm for solving it. Given a set of Service Level Objectives (SLOs) of the applications executing in the system, our algorithm continually adapts the per-component performance thresholds to the observed SLO violations. The purpose of this continual threshold adaptation is to control the average amounts of false positive and false negative alarms to improve the efficacy of the threshold-based management.

We implemented the proposed algorithm and applied it to a relatively simple, albeit non-trivial, storage system. In our experiments we achieved a positive predictive value of 92% *and a negative predictive value of* 93% *for component level performance thresholds.*

1. Introduction

Autonomic computing requires computer systems to be capable of self-management [26]. To this end, the autonomous system needs to monitor its performance and differentiate between normal and abnormal behavior. Self healing requires that the system autonomously react to the undesirable behavior, providing some remediation. Undesirable behaviors can be roughly classified into two major classes: degraded functioning known as performance problems, and failures, *i.e.*, a total inability to carry out certain function. This study focuses on performance problems.

Modern computer systems are complex multi-component systems. Monitoring commonly takes place at the component level whereas problem identification and remediation are required at both the system and the component levels. Usually, detection of performance problems is done as follows. At the system level, the system's behavior is matched against the Service Level Objectives (SLOs) of the applications that consume the system's services. SLO violations are considered the system-level performance problems. At the component level the monitored operational parameters are compared against their preset thresholds. Threshold violations are considered problematic at the component level, and may also indicate performance problems at the system level.

Using pre-configured component-level thresholds for detecting performance problems in a managed system is a ubiquitous performance management approach. However, it is well known that correlating component-level threshold violations with the system-level SLO violations is challenging. In this work we address this problem.

The major weakness of threshold-based management schemes that employ non-adaptive, pre-configured component thresholds is the assumption that the thresholds on the values of operational parameters are known in advance. In practice, suitable thresholds are seldom known. Sometimes useful values of a few thresholds might be provided by component manufacturers. Typically, for a specific system these values are far from optimal. In general, even if initial adequate thresholds on the parameters are given, there is still a need to adapt them to changes in the system that come in two forms: 1) evolution in the workload patterns; 2) evolution of the business goals that the system has to support.

Another weakness usually attributed to threshold-based

Proceedings of the Second International Conference on Autonomic Computing (ICAC'05)
0-7695-2276-9/05 $ 20.00 **IEEE**

management is that component-level thresholds are not very useful in the absence of a semantic model that explicitly defines the relationships and dependencies among the applications and system components and among the system components themselves. Detecting a component-level threshold violation may indicate that there exists a problem at the system level or that such a problem is imminent. However, without knowing at least something about the system's structure it is very difficult to identify what this problem is and which parts of the system are affected.

Often, the structure of a system is difficult to capture. This situation is changing, however, as systems become increasingly service-centric and component-based. In such systems each component provides its services to other components through well-defined interfaces and performance guarantees of a component are explicitly defined in component's SLO (we discuss this in greater detail in Section 3). Storage Area Networks (SANs) [18] are an example of service-centric, component-based systems. In [7] we presented a threshold-based performance problem root cause analysis algorithm that explicitly constructs a semantic model of component relationships in the SAN. In [3], an explicit semantic model is constructed for electronic transactions. In some systems, components' interfaces and SLOs are not well defined or difficult to analyze explicitly, let alone building a satisfactory semantic model for their interrelationships. For such systems, machine learning or data mining techniques may be used to extract some information about the internal system organization [6, 11, 14, 19].

Current practices of system management do not automate threshold setting. Therefore, an administrator, if she wants to deploy a threshold-based management scheme, needs to resort to her intuition and experience with the system and set the per-component performance thresholds manually. This is an error-prone and tedious process. Moreover, this process has to be repeated each time workload patterns of the system or its SLOs change considerably. For modern computer systems which consist of hundreds and even thousands of diverse components, *e.g.*, SANs, the manual approach does not scale.

Despite the drawbacks pertaining to the threshold-based performance management we argue that this approach has a very prominent advantage. This advantage is simplicity and intuitive nature of thresholds. Even a novice administrator has no trouble understanding the concept. In system management thresholds serve as a basic building block for constructing sophisticated management functionality just as simple Boolean gateways serve as the basis for creating more complex logic. A bulk of research and practical work in, *e.g.*, Policy Based Management postulates availability of relevant threshold-based mechanisms for alarm generation [5, 7, 12, 21, 22, 24].

Automatic derivation, setting and adaptation of statistically meaningful thresholds on the values of operational parameters of system components is essential for the management of computer systems, and even more so for autonomic, self managed systems. The mechanisms proposed in this paper, perform on-line adjustment of component level thresholds with controllable levels of false positive and false negative alarms with respect to the performance SLOs of the system as a whole. The core idea of the proposed solution is modeling of SLO violations using logistic regression [4] and subsequent retrospective adjustment of the thresholds. The thresholds serve as independent explanatory variables in the regression equation. However, while usually there is no control over the explanatory variables in statistical models, component level thresholds are controlled by our algorithm that plays the role of an 'automated administrator' and adjusts them as needed.

We first present the basic generic algorithm and provide an empirical study of its performance in a relatively simple, albeit non-trivial, computer system. Then, we discuss the way in which the basic algorithm can be applied to more complex systems and which enhancements this may entail.

To the best of our knowledge, the problem of automated adaptive threshold setting with controllable error levels with respect to the application level SLOs was not solved to date. The more important research in this direction is reported in [3, 9, 16]. In general, however, while there is a considerable effort devoted to using thresholds once they are defined [15], until very recently there was almost no effort to solve the problem of their automatic and adaptive computation.

Some similarity to our study can be found in [10], where components' performance metrics behavior is correlated with SLO violations at the application level. In contrast to our study, that work neither computes, nor uses component-level performance thresholds. We discuss that paper in Section 8 in more detail.

Thresholds computed by our algorithms can serve as a basis for multiple performance management techniques. Our initial feasibility study (described in Section 5) examined a relatively simple storage system. Additional experiments are currently being performed to examine more intricate SAN architectures. However, the scope of the concepts presented in this paper have a far broader field of application beyond SAN performance management. We believe that automated threshold setting is an enabling technology for the vision of autonomic computing.

2. Model and Problem Statement

Let $C = \{c_i\}$ denote the managed system comprised of components $c_i \in C$. Let $A = \{a_j\}$ be a set of applications. Let $M = \{m_l\}$ be a set of performance metrics measured at $\{c_i\}$ by means of some monitoring tools. Let

$A \times C \rightarrow [0,1]$ and $C \times C \rightarrow [0,1]$ be explicit dependencies defined among the applications and system components and among the system components themselves. A value in $[0,1]$ denotes a *strength* of the dependency.

We assume that an *alarm target* exists – a processing entity to which alarms following detected threshold violations at the component level are sent. We assume a discrete time model, *i.e.*, $t \in \mathcal{N}$.

Let $SL = \{s_k\}$ be a set of SLOs such that for each $a_i \in A \; \exists \; s_k \in SL$. We focus on *binary* SLOs. Multilevel SLOs are captured by a straightforward generalization of our basic solution (discussed in Section 7.4). Let the binary SLO of an application be defined as a first order predicate $slo(r)$, where r is an application level performance metric. For example, r may be the average response time, and the SLO of the application may be defined as $r \leq R$, where R is a pre-specified value. At any time t, $slo(r)$ evaluates to either TRUE or FALSE. For this reason we refer to it as binary performance SLO. For brevity, wherever no ambiguity arises, we use the term SLO referring to binary SLOs.

	SLO violated	SLO not violated
Threshold violated	x	y
Threshold not violated	u	v

Table 1. Alarms and SLO violations distribution.

Let c be a system component. Let μ be a metric measured on this component. Let $\gamma_{c,\mu}$ be a threshold value defined on c, μ. We assume that, at regular time intervals w, component-level monitoring agents compare the value of each component metric against the threshold defined for the metric. In case of a threshold violation, an alarm is propagated to the alarm target.

Table 1 depicts the possible relations between component threshold violations and SLO violations. In the table, x counts threshold violations detected for $\gamma_{c,\mu}$ that coincide with the SLO violations and y counts threshold violations that do not coincide with SLO violations. Similarly, u counts non-violations of $\gamma_{c,\mu}$ that coincide with SLO violations, and v counts non-violations of $\gamma_{c,\mu}$ that coincide with non-violations of the SLO. These counters are used to define positive and negative predictive values of a threshold as follows.

Definition 1 *Positive Predictive Value (PPV) of a threshold*

$PPV_{\gamma_{c,\mu}} \stackrel{\text{def}}{=} \frac{x}{x+y}$ is the probability of violating $\gamma_{c,\mu}$, when there is indeed an the SLO violation of the application.

Definition 2 *Negative Predictive Value (NPV) of a threshold*

The $NPV\gamma_{c,\mu} \stackrel{\text{def}}{=} \frac{v}{u+v}$ is the probability of component level threshold not being violated, when there is, indeed, no SLO violation.

Given the above definitions, the level of TYPE1 errors for c, m, or *false positive alarms* ratio is $1 - PPV_{\gamma_{c,\mu}}$. The level of TYPE2 errors, or *false negative alarms* ratio is $1 - NPV_{\gamma_{c,\mu}}$.

Let $0 \leq \alpha_{c,\mu} \leq 1$ and $0 \leq \beta_{c,\mu} \leq 1$, $\alpha_{c,\mu} \neq \beta_{c,\mu}$, be the PPV and NPV desirable for the threshold on the component performance metric.

Problem 1 *On-Line Thresholds Setting Problem*

At time t, for each of the system components of interest c, and for each performance metric of interest μ compute threshold $\gamma_{c,\mu}(t+1)$, such that as $t \rightarrow \infty$, then $PPV_{\gamma_{c,\mu}} \rightarrow \alpha_{c,\mu}$, and $NPV_{\gamma_{c,\mu}} \rightarrow \beta_{c,\mu}$.

In other words, we need to compute performance threshold values on-line such that, on average, the PPV and NPV levels are maintained over the total operation period. This requires that the system operates for a sufficiently long period of time and the underlying stochastic process remains stationary for sufficiently long. However, it does not preclude time instances in which the threshold defined for the component may violate its desired PPV and NPV levels. From a practical perspective it is important that the threshold setting algorithm converge fast to the desired PPV and NPV levels when the underlying statistical process stabilizes. In our feasibility study, convergence takes only a few iterations of the algorithm (see Section 5 and Section 6).

Note that we do not assume that the system behaves as a *single* stationary stochastic process during its operation. On the contrary, we assume that the statistical nature of the system's behavior may change multiple times. We only assume that each time such a change occurs, it lasts sufficiently long. In other words, the system is not a chaotic one, as in this case it would be unmanageable in the strong sense – no amount of monitoring, analysis and remediation would help its performance. We assume that the system may operate in multiple normal and abnormal modes. We now formulate the necessary condition for solving Problem 1.

Definition 3 *Stochastic Monotonicity of a Metric (1st necessary condition)*

Let m_1, m_2 be values of performance metric μ at different time instances t_1 and t_2, and r_1, r_2 be the values of application performance metric ρ at the same time instances. Stochastic monotonicity requires that:
$Pr(r_1 \leq r_2 | m_1 \leq m_2) > Pr(r_1 > r_2 | m_1 \leq m_2)$.

The second necessary condition for solving Problem 1 is more difficult to formalize. Informally, it requires knowing the relationships among the components. This is needed for considering only SLO violations and component level performance threshold violations that are relevant to each other. In [7], explicit relationships between the system's components (SAN in that case) can be extracted from analyzing the system's configuration data. This is an example

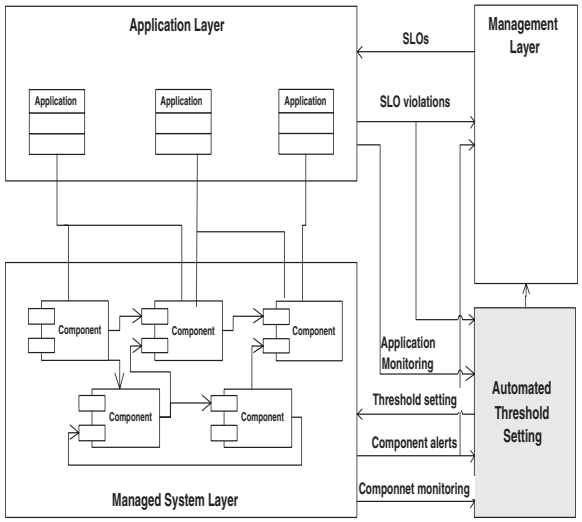

Figure 1. Autonomous Management Architecture

of a system in which the necessary condition of knowing the system's structure holds.

For simplicity, we first address the case of a single application having an SLO, a single component that can be monitored, and a single performance metric, *i.e.*, $|A| = 1$, $|C| = 1$, and $|M| = 1$. Extensions that address multiple applications, multiple components, and multiple metrics are discussed in Section 7.

Our example domain is storage-oriented, and in particular SAN-oriented. There are two reasons for this. First, the relationships among the SAN components are well defined. Second, our empirical studies show that in storage systems stochastic monotonicity condition holds. Note that our solution is applicable to any system that satisfies the stochastic monotonicity, and for which information about the relationships between system components is available.

3. Autonomic Threshold-Based Performance Management Architecture

Figure 1 depicts the envisioned generic architecture of autonomous, self-managing computer systems. It shows the role that our contribution – the automated and adaptive threshold setting mechanism (presented in grey) – serves within this architecture.

The autonomic computing approach proposes organizing computing systems as self-coordinating and self-managing ensembles of hardware and software components. These components communicate through well-defined standardized interfaces. Components provide and consume services from each other in complacency with the service level agreements (SLAs) defined among them. One crucial aspect of an autonomic component's behavior is the performance level that this component guarantees to other components that may consume its services. This part of an SLA is known as *service level objective (SLO)* and may include a variety of parameters, *e.g.*, reliability level, maximal average response time, minimal average throughput, *etc.*

Since the non-trivial services, such as data storage service, on-demand computation service, connectivity service, *etc.* are rarely provided by a single component, component-based systems are organized as follows. There is a structured set of the *system-level components* that together provide a service; there exist multiple *application-level components* that consume the service provided collectively by the system-level components; this service consumption is performed through the system *front-end* – a subset of the system-level components.

The gradual move to service-oriented, componentized (and eventually autonomic) systems, requires efficient performance management. An automated threshold setting mechanism is an essential building block of such management solutions, as explained in the introduction (Section 1). The current paper focuses on this building block.

4. Basic Algorithm

In this section, we present a generic on-line algorithm for automated and adaptive setting of thresholds on the operational values of component performance metrics. For simplicity of presentation, we consider a system consisting of one component of interest, having a single performance metric that can be measured, a single control application (*i.e.*, the application for which a binary performance SLO is defined), and, possibly, multiple other applications that inflict an additional load on the system. Although such a system may seem unnaturally simple, we show in Section 5 that real systems can be modeled this way. In Section 7, we discuss more complex system settings and the application of our basic algorithm in these settings.

Let $\Gamma(t) = \{\gamma(0), \gamma(1), ..., \gamma(t)\}$ denote the time series of the thresholds used from time 0 to time t. Let $A(t)$ denote a stochastic binary variable, such that $a(t) = 0$, if $slo(r) \equiv true$, *i.e.*, the SLO of the application is satisfied at time t, and $a(t) = 1$ otherwise. Let $\gamma_{c,\mu}(t)$ be the threshold value for performance metric μ of component c at time t. Since there is only one component and one metric, we omit the indices. Let $Y(t)$ denote a stochastic binary variable, such that $y(t) = 1$ if the operational value of μ on c violates $\gamma(t)$ at time t, and $y(t) = 0$ otherwise.

Let α and β be, respectively, the target PPV and NPV levels desirable for $\gamma(t)$, *s.t.*, $\alpha \neq \beta$. Let A_0^t denote a time series of the application SLO behavior ($a(t)$). Let Y_0^t denote a time series of the performance metric behavior ($y(t)$. As-

suming stochastic monotonicity (Definition 3), we need to solve Problem 1. That is, we need to compute $\gamma(t+1)$, the threshold value for the next time step, such that as $t \rightarrow \infty$, the $PPV \rightarrow \alpha$ and the $NPV \rightarrow \beta$.

We use a statistical approach to solve this problem. We treat $Y(t)$ (which encodes relationship between measurements of μ on c and $\gamma(t)$)) as a response variable. Using $\Gamma(t)$ and $A(t)$ as explanatory variables we construct a regression model for $Y(t)$. Then we solve the regression equation treating $\gamma(t+1)$ as an unknown variable to obtain the threshold value with the specified PPV and NPV for $t+1$. Thus Γ plays a dual role. It is an explanatory variable in the regression model that explains the past behavior of the system. It is also an unknown variable when the regression model is extrapolated to the next time step.

The reasons for using $\Gamma(t)$ and $A(t)$ variables as predictors are obvious. $\Gamma(t)$ directly controls the behavior of $Y(t)$: if the threshold is set to a too large value, no threshold violation events would follow, the value of $Y(t)$ will be 0 most of the time, and the percentage of false negative alarms (TYPE2 errors) will rise sharply. If, on the other hand, the threshold is set to a very low value, many threshold violation events will be generated, $Y(t)$ will equal 1 most of the time, and the percentage of false positive alarms (TYPE1 errors) will increase beyond the desired level[1]. $A(t)$ helps to explicitly connect the application behavior to the behavior of a performance metric in the system. Since it is assumed that the operational values of the performance metric are indicative of the application level problem, $A(t)$ is necessary in order to achieve the desired levels of PPV and NPV, as explained below.

Since $Y(t)$ is a binary variable, we cannot use the regular linear regression model of the form $y = c + \vec{b} \cdot \vec{x}$, where $\vec{x}$ is the vector of explanatory variables. One reason for this is that y exceeds 1 for sufficiently large values of $\vec{x}$. Another reason for the regular linear regression being inapplicable is that the significance testing of the regression coefficients is based on the assumption that the residual errors of prediction, $\epsilon = y' - y$, are normally distributed[2]. Because y only takes values 0 and 1, this assumption, obviously, does not hold. Yet another important reason is that we need a mixture of categorical, $A(t)$ and continuous, $\Gamma(t)$, predictor variables in our model. For these reasons we use a variant of Generalized Linear Model regression with the $logit$ link function called Logistic Regression. This is a commonly used statistical model for analyzing categorical data [4].

[1] In general, there exists a dependency between the PPV and NPV of a threshold. PPV is is monotonically non-increasing with the threshold value, while NPV is monotonically non-decreasing with the threshold value. If the threshold is set too high, its PPV will decrease while its NPV will increase. If, on the other hand, the threshold is set to a very low value, the PPV of the threshold will increase, while its NPV will decrease.

[2] y' are the values predicted by the regression model, and y are the observed ones.

The main idea behind using the logistic regression is to predict the probability of $Y(t+1)$ to assume 1 conditioned on the predictor variables values rather than predicting the value of $Y(t+1)$ directly.

Let $p(\vec{x}) = P(Y = 1|\vec{X} = \vec{x}) = 1 - P(Y = 0|\vec{X} = \vec{x})$. The Logit transformation of $p(\vec{x})$ is $logit(p(\vec{x})) = \ln \frac{p(\vec{x})}{1-p(\vec{x})}$. The logit value can assume any value between minus and plus infinity. Therefore, we may use the logit value as a response variable in the regular linear regression as follows:

$$logit(p(\vec{x})) = c + \vec{b} \cdot \vec{x}. \tag{1}$$

The regression coefficients c, $\vec{b}$ are derived using the Maximal Likelihood (ML) estimator [4]. Assuming that the logistic regression is fitted successfully (*i.e.*, the goodness of statistic fit of the model is sufficient for the desired confidence level), from Equation 1 we obtain:

$$p(\vec{x}) = \frac{1}{1 + \exp(-c - \vec{b} \cdot \vec{x})}. \tag{2}$$

Using our explanatory variables: $x_1 \equiv a(t)$ and $x_2 \equiv \gamma(t)$, we predict the probability of threshold violation, as follows:

$$p(t+1) = \frac{1}{1 + \exp(-c - b_1 \cdot a(t+1) - b_2 \cdot \gamma(t+1))}. \tag{3}$$

Since $a(t+1)$ may evaluate to either 0 or 1, we get, for the case of the SLO not being violated:

$$p(t+1) = \frac{1}{1 + \exp(-c - b_2 \cdot \gamma(t+1))}, \tag{4}$$

and for the case of the SLO being violated, we obtain:

$$p(t+1) = \frac{1}{1 + \exp(-c - b_1 - b_2 \cdot \gamma(t+1))}. \tag{5}$$

Equation 5 renders the probability of the true positive alarms, and, as $t \rightarrow \infty$, this probability represents PPV of a threshold. Similarly, for a large enough time series, $1 - p$, where p is given by Equation 4, approximates NPV of a threshold. Therefore, given pre-specified PPV α and NPV β, we compute the system component threshold for the next time instance from Equation 4 and Equation 5 as follows:

$$\gamma(t+1) = \frac{\ln \frac{\alpha}{1-\alpha} - \ln \frac{\beta}{1-\beta} - 2 \cdot c - b_1}{2 \cdot b_2}. \tag{6}$$

Equation 6 forms the core of the threshold setting algorithm. The algorithm starts with a random (or default) threshold and adjusts the threshold value at each iteration.

Recall that in the general case it is not possible to obtain exactly the same levels for PPV, α, and NPV, β for the threshold since there is a trade-off between PPV and NPV. Equal PPV and NPV levels are possible only if the system behaves in a very specific way. For example, if there are no dynamic changes in the system's workload and the system exhibits no performance problems at all, then equal PPV and NPV levels can easily be achieved.

The value of $\gamma(t+1)$ obtained in Equation 6 implicitly depends on values of $\gamma(0), \gamma(1), ..., \gamma(t)$, since the coefficients b_2 used in this equation, are obtained from the multivariate logistic regression model that fits the previous values of the performance metric thresholds. This introduces a bias to the regression model itself. Fortunately, we have full control over the explanatory variable Γ, the threshold. Let $\overline{\mu}$ be the sample mean of performance metric μ, and σ be the sample variance. In order to reduce the bias in this explanatory variable, we augment the time series of γ values that were actually used in the past as the thresholds, with the dummy γ' values that we toss from the discrete uniform distribution $U(max(\overline{\mu}-3\cdot\sigma, 0), \overline{\mu}+3\cdot\sigma)$ and apply retrospectively to the time series of the performance metric and application SLO, to obtain additional, virtual, points in the time series. This allows a more accurate model fit.

To illustrate the rationale behind the bias reduction procedure, assume that at time $t' < t$ the value of performance metric μ was m, *i.e.*, $\mu(t') = m$, and $\gamma(t') = g$ and, without loss of generality, let $g > m$. Without loss of generality, let's assume also that $slo(\cdot) = false$. As one can notice, this situation represents a TYPE2 error, *i.e.*, a false negative alarm, and, correspondingly, $a(t') = 1, y(t') = 0$. After we toss random threshold values $\gamma'_1(t'), \gamma'_2(t'), ..., \gamma'_k(t')$, and compare these threshold values against $\mu(t') = m$, we obtain more samples (virtual samples) for $y(t')$. Some of these new sample points still correspond to TYPE2 errors, yet others correspond to TYPE1 errors, yet still others correspond to the desirable behavior. The implementation of the algorithm as part of a performance management tool may take the form presented in Figure 2. Note that the thresholds are recomputed only when necessary to reduce computational overhead. If the current values of PPV and NPV meet their target values, no correction to the thresholds is required.

Before we proceed to describing our experimental results with the threshold setting algorithm presented in this section, we would like to point out that the algorithm, as presented, assigns the same importance to all observation points in $A(t)$ and $Y(t)$ time series. This may lead to degraded accuracy of prediction in the regression model, since too old historical observations may introduce the undesirable bias and high variance. At the same time, if the system exhibits complex behavior, this problem cannot be solved by a straightforward weighting of the observation points, *e.g.*, exponential weighting. It is known that real systems often behave in repetitive patterns that correspond, for example, to certain hours and week days [9]. Thus, the weighting scheme should take this behavior into account.

```
1.  ATS(α, β, σ) {
2.    //α and β are the PPV and NPV value;
3.    //may also be set to default values:
4.    // e.g., α = 0.95, β = 0.85
5.    //σ is the required confidence level
6.    initialize γ(0) //Set to a random value, unless known
7.    while (TRUE) { //main event loop
8.      given SLO observation at time t, do { //event arrived
9.      boolean v ← isViolated(SLO)
10.       if ((v ∧ (μ(t) < γ(t))) ∨ (¬v ∧ (μ(t) > γ(t))))
11.         if ((current PPV < α) ∨ (current NPV < β)) {
12.           fit logistic regression (see 5)
13.           if σ was achieved (use -2LL to test)
14.             compute γ(t + 1) using equation 6
15.         }
16.       else
17.         γ(t + 1) ← γ(t)
18.     } //end: do
19.   } //end: while (main loop)
20. } //end: ATS
```

Figure 2. Pseudo-code of the automated threshold setting algorithm

5. Experimental Evaluation

We implemented our algorithm using MATLAB [20] and evaluated its performance on the trace that we gathered from our system. Figure 3 depicts our experimental setting. The managed system consists of a Windows 2000 workstation and multiple I/O bound applications which consume I/O services through a Veritas Logical Volume Manager (LVM) on top of a RAID-5 disk array. The I/O bound applications were simulated using the standard I/O micro-benchmark software iozone [2] and iometer [1].

One of the I/O processes was designated as a 'test application'. Its performance SLO was defined in terms of maximal acceptable response time. We monitored the response time of the test application. Other I/O bound applications were not assigned SLOs and simply loaded the storage sub-

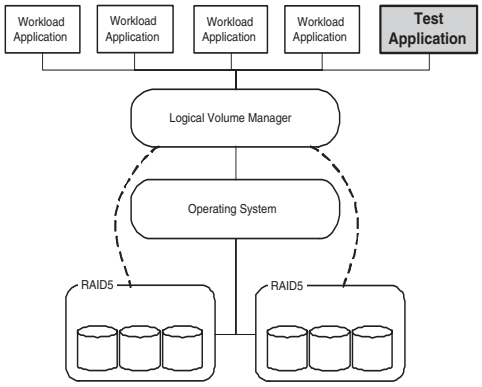

Figure 3. Experimental Testbed

system by repetitively executing benchmarks with random sleep periods in between.

We tagged each application sample point as either "SLO violation" ($a(t) = 1$), or "SLO conformance" ($a(t) = 0$). Our trace contains 375 observations for the application side, and over $190,000$ system-side measurements, each application observation corresponding to ~ 500 observations of the system component (*i.e.*, of the logical volume). The measurements of LVM were performed every second. At the system side we sampled the values of two performance metrics: *average requests arrival rate* (measured in I/O requests per second) and *average response time* of the logical volume (in seconds per *completed* request).

After obtaining the performance trace of the system we executed our algorithm in an on-line manner as follows. At each point in time in which the 'test application' produced a response time measurement, the latter was compared against the SLO definition. Correlating application's SLO status with threshold violation/non-violation status at this point in time, we identified "false positive", "true positive", "false negative", and "true negative" events (see Section 2). If either a false positive or a false negative event was detected and the target PPV or NPV level(s) were not satisfied at this point, the performance thresholds for the LVM performance metrics were recomputed. Only the information seen in the trace up to the current point in time was used for recomputing the thresholds (see Figure 2 for a detailed algorithm).

We explained earlier that in the general case achieving PPV and NPV target levels equal to one another is impossible. To check which maximal PPV (α) and NPV (α) levels can be attained simultaneously (see Equation 6) we performed a series of experiments in which one of the parameters was fixed and the other was varied. To limit the number of runs we excluded levels not acceptable for practical system management. Minimal acceptable values for PPV and NPV were set at 85%. For the raw values of the metrics the

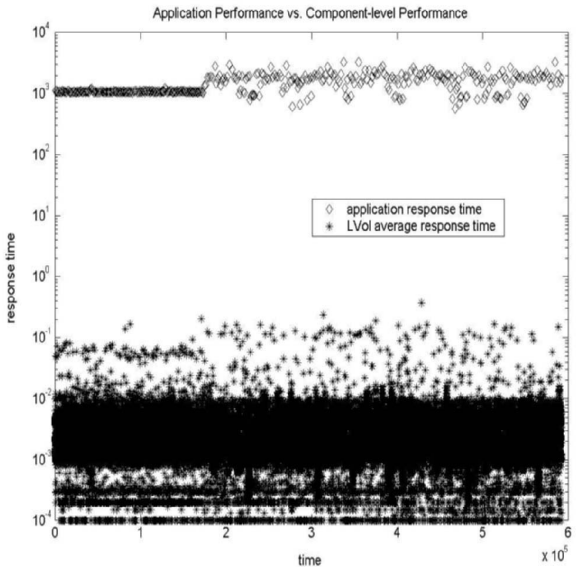

Figure 4. Application Response Time and Logical Volume Response Time, as functions of time

best combination attained was a PPV level of $\sim 89\%$ and an NPV level of $\sim 91\%$. When instead of using the momentary values we used the variance, the results were even better. We attained 92% of PPV (a 3% increase in PPV).

The algorithm exhibited fast convergence. Figure 7 depicts convergence of the threshold values for a specific SLO definition. There was no dramatic difference between the results attained by using either the arrival rate of I/O requests, or the response time of the logical volume as a predictor variable. In both cases the results were very similar, albeit using the response time resulted in marginally better results.

The basic algorithm assumes that at each iteration, the regression model successfully fits the observed history. The goodness of fit was measured using $-2LL$ (double log-likelihood) statistics, using Chi-Square test. We used a 0.95 confidence level for deciding whether a fit is successful. If the fit was not attained at this level (*e.g.*, due to the overly high variance), the thresholds were reset to random values as if there was no history. There were just a few such cases in applying the algorithm to this trace. We are currently investigating more sophisticated methods for *history selection* to guarantee a good model fit.

6. Discussion of Experimental Results

Figure 4 presents two scatter plots overlapped on the same graph: the response times of the application and of the logical volume as functions of time. Inspecting this graph

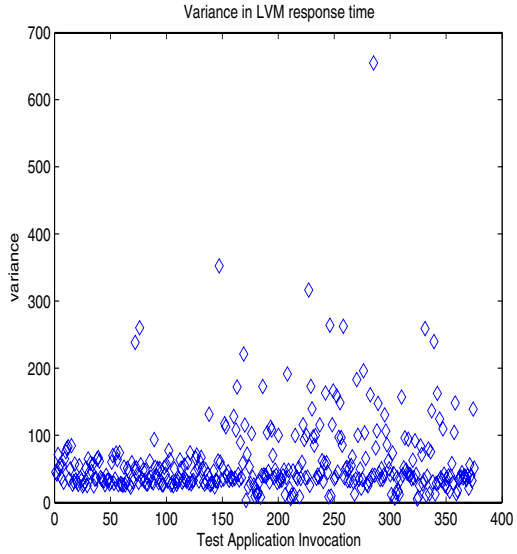

Figure 5. Variance in Logical Volume Response Time, as a function of time

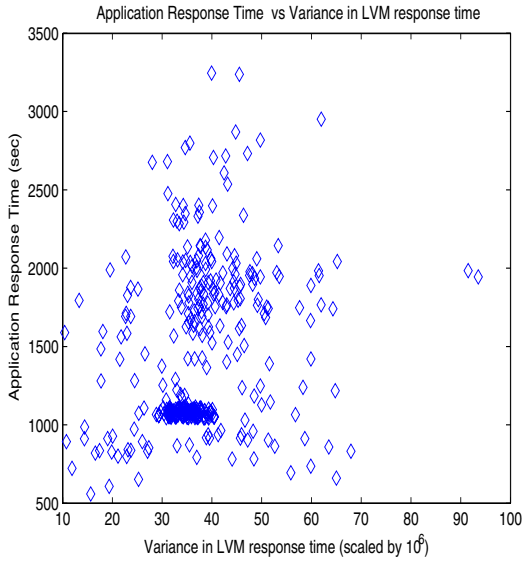

Figure 6. Application Response Time vs. Variance in Logical Volume Response Time

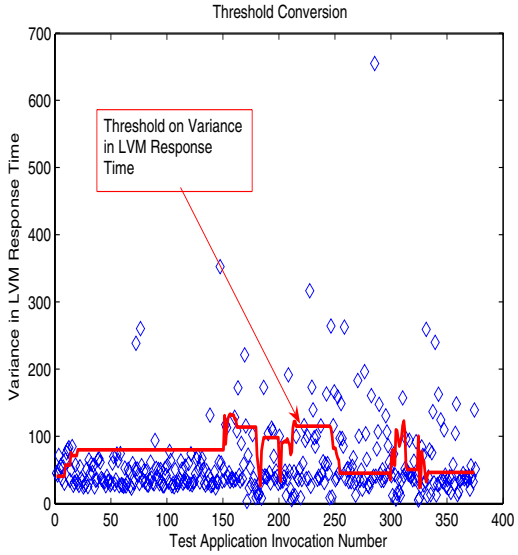

Figure 7. Threshold Convergence

shows that the metrics behavior is stratified. The strata correspond to different modes of system operation. In other words, there exist a number of operation modes that do not result in SLO violations. These modes significantly differ from one another in terms of the values of the performance metrics measured in each mode. Therefore, the adaptivity of the thresholds is indeed important. Another phenomenon to notice is that the higher values (outliers) of the application behavior roughly coincide with higher values of the logical volume metrics. This supports the conjecture of stochastic monotonicity (see Section 2) that justifies using thresholds as a meaningful management mechanism. Figure 5 depicts variance of the logical volume response time. As the absolute values of the variance are rather small (tens of milliseconds, see Figure 4), we performed 'scaling' by multiplying all variance values by 10^6. The scaling was needed to preclude numerical instability in the algorithm due to very small numbers.

One may notice that Figure 5 is considerably less noisy than Figure 4. Most values are being well clustered with relatively few outliers. At the same time the overall structure of the workload is preserved. This explains the improvement in accuracy when the variance was used as one of the predictor variables instead of the raw values in the regression model.

Figure 6 shows the application response time plotted as a function of the variance of the logical volume response time (scaled by 10^6, as explained above). There is no temporal structure in this graph. It merely shows the way in which the application behavior depends on the system component metric behavior. One can observe clusters of data points

that correspond to different operational conditions. The stochastic monotonicity assumption is well pronounced in this figure. However, a visual inspection suggests that a simple linear regression model will not be an accurate model for capturing the statistical structure of this data.

Figure 7 superposes two plots: the scaled variance in LVM response time and the threshold setting dynamics for SLO = 1700. Note that decreasing the SLO value, increases the number of SLO violations for a given trace. For the trace in our experiment SLO = 1700 yielded $\sim 40\%$ SLO violations. Every point on the abscissa corresponds to a measurement of the test application's response time taken upon its termination. As explained in Section 4, a threshold is recomputed at these points. If the threshold is sufficiently accurate, it remains unchanged for the next iteration (see line 16 of the pseudo-code shown in Figure 2).

As one can observe from Figure 7, in the first part of the trace, when the workload is less intense and there are fewer SLO violations, the threshold converges very fast and remains almost unchanged. This holds approximately until 'time' 150, when much more load is imposed on the system and the latter becomes noisier. In this part of the trace the threshold is updated more often and it takes more iterations to converge. However, on average it takes ~ 10 iterations. The algorithm is resistent against the occasional spikes in application response time that are not related to high variance values of the LVM response time. These spikes can be attributed to the operating system buffering and scheduling and not to poor LVM performance.

7. Extended Applications of the Algorithm

7.1. Multiple Metrics

Above we have solved the threshold setting problem for a single metric. A general solution should address multiple metrics. A naive approach is to execute the basic solution for each metric separately. Although this approach may work well for a small number of metrics, it has two disadvantages. Firstly, many performance metrics are mutually dependent, hence dependencies among the thresholds set on them may likely exist as well. Secondly, some metrics are of minor importance for system management. The major problem stemming from dependency among the metrics is the multi-collinearity. This implies that introducing additional predictor variables corresponding to more metrics into the regression model should be done with great caution.

Identifying metrics of minor importance early-on and ignoring them can significantly reduce the computational complexity of the proposed solution. Fortunately, means for doing that are known in the art (e.g., dimensionality reduction techniques). We have executed a preliminary feasibility study of using such techniques for SAN performance metrics. Initial indications are that the number of performance metrics for the SAN can be reduced significantly without adversely affecting the model's quality.

7.2. Multiple Components and Inseparability

Large systems consist of a multiple components that require performance management. One problem with using simple per-component performance thresholds is that in many cases the managed system may behave such that only certain combinations of components' behaviors lead to performance problems. Moreover, it is also possible that, when we inspect each component in isolation from others, performance thresholds are violated at neither of them, yet the combination of current values of performance metrics indicates a problem. When this happens, we say that the system is not separable. The number of combinations of components and metrics is exponential. Hence, it is infeasible to inspect them all. It is also not scalable to add too many predictor variables (at least one per component) into our basic regression model, as it increases the computational cost, while simultaneously reduces the overall quality of the model.

One approach to dealing with this problem is using coarser component granularity. This approach has inherent limitations. Another approach is using advanced statistical methods such as those employed for Data Mining, in order to identify only the most important combinations of metrics among different components and use their weighted linear combinations as derived metrics, on which the basic algorithm of Section 4 computes simple thresholds.

7.3. Multiple Applications

There may exist multiple applications which share system's resources and whose SLOs should be accommodated simultaneously[3]. There are various ways to deal with multiple applications. One natural extension of our basic solution is to define a single model for all applications using the binary variable $A(t)$ to model SLO violations in any application. Yet a more sophisticated approach is to classify the applications into classes and using $A(t)$ to model SLO violation in any application of a given class. Finally, the most expensive solution is to employ a separate regression model for an individual application behavior. This approach may increase the preciseness of the computed thresholds since they are crafted on a per-application basis, however it may not scale.

[3]Note that in our experimental study we already have multiple applications in the system, but we measured SLO violations only for one of them–the control application–for which the SLO was defined.

We suggest using either the first or the second approach to construct the regression model. We supplement these with the *case study* [4] statistical method. Namely, we treat a threshold violation by a component as an *effect*. When a threshold is violated, we compute the proportion and types of applications for which SLO violations are observed in conjunction. Over time, we develop an empirical probability for a given threshold violation to affect a given application. Consequently, if we observe that the required PPV and NPV levels are not satisfied for some application, we may consider crafting its individual threshold setting regression model.

7.4. Multi-level SLOs

The basic threshold computation method presented earlier is suitable for "binary" SLOs. More sophisticated SLOs may exist. For example, one may consider an SLO that has multiple levels and for each level there is an associated price. We can easily extend our method to accommodate multi-level SLOs by using multinomial logistic regression [4].

8. Related Work

Using thresholds is a common practice in performance management [15, 24]. Typically, performance thresholds are not adaptive and do not take into account application SLOs. Despite the obvious need for having such thresholds as part of the management arsenal, relatively few advances were reported on this problem so far.

A recent paper [10] studies correlations between the behavior of component performance metrics and application SLO violations. These correlations are used to build a performance anomaly detector for three-tier client/server systems. In seeking performance anomaly detection, the work in [10] is similar to ours. However, our study differs from that work in several ways. First, [10] suggests using Tree-Augmented Bayesian Networks (TANs) as a means for anomaly detection. In that approach performance thresholds at the component level are neither computed nor used. In contrast, we take a more incremental approach, as our solution suggests to improve the existing threshold-based performance management paradigm by automating and optimizing threshold computation. Therefore, our approach can be adopted seamlessly since in most managed systems performance thresholds already exist. Second, our solution allows for the user to set a desired level of TYPE1 and TYPE2 errors, and the thresholds are computed to meet, on average, these target levels. The solution in [10] measures the level(s) of false positives (negatives), but does not allow one to set desired levels of these quantities. Finally, our approach is better suited for a distributed implementation. Monitoring agents do not have to communicate with the management station at each step. Communication is necessary only when a local threshold violation is detected.

One of the first attempts to solve the problem of dynamic threshold setting for management purposes was presented in [17]. In that work a seasonal model was proposed for analyzing the time series of variables (*e.g.*, memory usage, disk space, number of users, *etc.*, indicating the workload of a host) pertaining to hosts, forming a network. The seasonal model was used to predict the workload of each host in the next time instance. When the difference between the predicted and observed workload values of a given host was perceived as too large, the host was declared problematic. As one may notice, the approach used in [17] did not really solve the problem that we are attacking. It rather shifted the focus from defining accurate adaptive thresholds on the values of the management variables themselves to defining the accurate threshold on the model's error. This task was still left to the human administrator.

This direction was developed and improved in [8, 13, 23, 25]. Common to these studies is definition of a statistical model based on the historical performance time series and then using it either for 1) predicting the future performance values and comparing the discrepancy between the predicted and observed values against a pre-set threshold value; or 2) continually recomputing the model and detecting considerable changes in the model parameters and subsequently searching for change points in the time series corresponding to the model parameter changes. However, in both cases the problem of finding direct thresholds on the operational values is not solved, but rather substituted by the problem of defining a suitable threshold value for statistical model's discrepancy.

In [15, 24] a seasonal model of metrics variance is used to predict threshold violations (for proactive management purposes). It is assumed that the thresholds themselves are preset. The problem of computing and adapting them is not addressed.

In [9] it is recognized that using very long time series is both computationally and space inefficient. To remedy these drawbacks, a two-dimensional time approach is introduced to parameterize the time series. This work assumes that the system behavior is strongly periodic (*e.g.*, due to the social cycles behind the usage of the system). To capture the periodic behavior of the system, the linear time is transformed into an elliptic one: $t = P \cdot n + \tau$ with the period P, which is discovered empirically. A statistical analysis (mean and variance computation) is applied to the data belonging to the same period and among the similar periods (*e.g.*, the same day of the week, the same week *etc.*) to define the dimensionless regions in terms of variance and mean which differentiate between the normal and abnormal behavior of the system. An important advantage of the two-

dimensional time-series approach introduced in [9], is that it is storage and computationally efficient. The main disadvantage is that it depends on the actual managed system behaving periodically as expected. In contrast to our work, [9] does not guarantee specific PPV and NPV levels of the thresholds and is best suited to be a long term anomalous tendency detection mechanism.

In [3] a simple on-line performance threshold scheme, based on exponential averaging is introduced for electronic transaction performance management. Performance thresholds computed using this scheme do not guarantee specific levels of PPV and NPV. Tuning of the threshold setting algorithm parameters, such as choosing the sample size and the weighting coefficient α, relies on empirical experience with the system and is left to the administrator.

9. Future Work

The methods proposed in this study require additional work to show their applicability to complex systems and SLOs. In current work we already address some of these requirements and future work will address others. In particular, we are currently deploying a complex SAN; on this SAN we will perform experiments with multiple components, multiple metrics, and multiple test applications. Utilizing the extensions of our basic algorithm (see Section 7), we will analyze the results of the experiments to compute thresholds for such complex settings. Further, as suggested in section 7.4, we will use the multinomial logistic regression to extend our solution to complex SLOs. Via these future studies we intend to provide a comprehensive solution to the automated on-line threshold setting problem.

10. Conclusions

Thresholds are fundamental to the management of computer systems, and in particular to the management of autonomic systems. Setting appropriate thresholds on the operational values of performance metrics is an intricate problem. To date, the threshold setting problem was not sufficiently addressed in the art.

In this study we present an innovative method for automated on-line threshold setting. Our method is founded on rigorous statistical techniques, supported by an analytical justification for the choices made in devising our method. We examine the proposed method via a set of experiments. The experiments show that our method arrives at very high levels of Positive Predictive Values (PPV) and Negative Predictive Values (NPV). This implies that a system in which our threshold setting method is implemented should have a low level of false positive and false negative error messages. Our basic algorithm is exemplified for relatively simple settings.

We believe that the method introduced in this paper, exemplified here for storage networks, should serve as a fundamental building block for the management of a variety of autonomic computer systems.

11. Acknowledgements

We thank Shmuel Gal (University of Haifa; IBM) for proof reading of the paper and suggesting valuable ideas. We thank Alain Azaguri (IBM), Hillel Kolodner (IBM), Danny Raz (Technion) for their comments on the problem. Special thanks go to Shai Fine (IBM) and Elad Yom-Tov (IBM) for their help and suggestions in performing the initial feasibility study of reducing the number of performance metrics in the SAN environment. We also thank the reviewers of the paper, who provided invaluable comments and helped us to improve the paper much.

References

[1] Iometer: an I/O system measurement tool. `www.iometer.org`.

[2] Iozone: a file system benchmarking tool. `www.iozone.org`.

[3] M. K. Agarwal, K. Appleby, M. Gupta, G. Kar, A. Neogi, and A. Sailer. Problem Determination Using Dependency Graphs and Run-Time Behavior Models. In *15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'04*, pages 171–182, New York, NY, USA, November 2004.

[4] Agresti, Alan. *Categorical Data Analysis*. John Wiley and Sons, Wiley Series in Probability and Statistics, Hoboken, New Jersey, USA, 2nd edition, 2002.

[5] B. Aiken, J. Strassner, B. Carpenter, I. Foster, C. Lynch, J. Mambretti, R. Moore, and B. Teitelbaum. RFC2768: Network Policy and Services: A Report of a Workshop on Middleware. `http://www.ietf.org/rfc/rfc2768.txt?number=2768`.

[6] U. Blumenthal, G. Kar, and A. Keller. Classification and Computation of Dependencies for Distributed Management. In *IEEE Symposium on Computers and Communication (ISCC 2000)*, July 2000.

[7] D. Breitgand, E. Henis, E. Ladan-Mozes, and E. Yerushalmi. Root-Cause Analysis od SAN Performance Problems: an I/O Path Affine Search Approach. In *IFIP/IEEE 9th International Symposium on Integrated Network Management, IM'05 (to appear)*, Nice, France, May 2005.

[8] J. D. Brutlag. Aberrant Behavior Detection in Time Series for Network Monitoring. In *USENIX LISA XIV*, pages 139–146, New-Orleans, 2000.

[9] M. Burgess. Two dimensional time-series for anomaly detection and regulation in adaptive systems. In *DSOM*, pages 169–180, 2002.

[10] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase. Correlating instrumentation data to system states: a building block for automated diagnosis and control. In *USENIX*

OSDI 2004, pages 231–244, San Francisco, CA, December 2004.

[11] Y. Diao, F. Eskesen, S. Froehlich, J. L. Hellerstein, A. Keller, L. Spainhower, and M. Surendra. Generic On-Line Discovery of Quantitative Models for Service Level Management. In *8th IFIP/IEEE International Symposium om Integrated Network Management*, pages 157–170, Colorado Springs, Colorado, USA, March 2003.

[12] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. RFC2748: The COPS (Common Open Policy Service) Protocol. `http://www.ietf.org/rfc/rfc2748.txt?number=2748`.

[13] H. Hajji, B. H. Far, and J. Cheng. Detection of network faults and performance problems. In *Internet Conference, IC'01*, pages 159–168, Osaka, Japan, november 2001.

[14] J. Hellerstein and S. Ma. Mining Event Data for Actionable Patterns. In *26th Computer Measurement Group Conference*, pages 307–318, Orlando, FL, USA, 2000.

[15] J. Hellerstein, F. Zhang, and P. Shahabuddin. A Statistical Approach to Predictive Detection. *Computer Networks*, 35(1):77–95, 2001.

[16] J. L. Hellerstein. A general-purpose algorithm for quantitative diagnosis of performance problems. *J. Networks Syst. Manage.*, 11(2), 2003.

[17] P. Hoogenboom and J. Lepreau. Computer System Performance Problem Detection Using Time Series Model. In *USENIX Summer*, pages 15–32, 1993.

[18] J. Tate and A. Bernasconi and P. Mescher and F. Scholten. *Introduction to Storage Area Networks*. Number SG24-5470-01 in IBM Red Books. IBM, April 2003.

[19] A. Kochut and G. Kar. Managing Virtual Storage Systems: An Approach using Dependency Analysis. In *IFIP/IEEE 8th International Symposium on Integrated Network Management, IM'03*, pages 593–604, Colorado Springs, Colorado, USA, March 2003.

[20] MathWorks. MATLAB. `http://www.mathworks.com/`.

[21] D. Rawlins, A. Kulkarni, M. Bokaemper, and K. Chan. IETF RFC3483, Framework for Policy Usage Feedback for Common Open Policy Service with Policy Provisioning (COPS-PR). `http://www.ietf.org/rfc/rfc3483.txt?number=3483`.

[22] H. Simitci. *Storage Network Performance Analysis*. John Wiley and Sons, USA, 2003.

[23] M. Throttan and C. Ji. Adaptive thresholding for proactive network problem detection. In *Third IEEE International Workshop on Systems Management*, pages 108–116, Newport, Rhode Island, April 1998.

[24] R. Vialta, C. Apte, J. Hellerstein, S. Ma, and S. Weiss. Predictive Algorithms in the Management of Computer Systems. *IBM Systems Journal*, 41(3), 2002.

[25] A. Ward, P. Glynn, and K. Richardson. Internet service performance failure detection. In *Internet Server Performance Workshop (in conjunction with SIGMETRICS'98)*, 1998.

[26] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart. An Architectural Approach to Autonomic Computing. In *IEEE 1st International Conference on Autonomic Computing (ICAC'04)*, pages 2–9, New York, NY, USA, May 2004.