

Online Tracking of Component Interactions for Failure Detection and Localization in Distributed Systems

Haifeng Chen, Guofei Jiang, Cristian Ungureanu, and Kenji Yoshihira

Abstract—This paper proposes a novel failure-detection approach that can handle high-dimensional observation and frequent system changes. We extract two statistics from the subspace decomposition of observations, and use the mixture of Gaussians to model their probability density. Instead of monitoring the original data, the density model of extracted statistics is adaptively updated and examined regularly to detect failures. We also present a localization method to identify the faulty components once the failure happens. Applying our technique to monitor the component interactions in an e-commerce application shows satisfactory results in detecting a variety of injected failures.

Index Terms—Distributed computing, failure detection, online tracking, subspace decomposition.

I. INTRODUCTION

WITH the increasing acceptance of the Internet, the number of large-scale services such as Google and Yahoo has grown significantly in recent years. Such systems are integrated with thousands of machines, and it is hard for them to be running $24 \times 7 \times 365$ excluding failures. A potential solution to increase availability, as presented in this paper, is to quickly detect and localize failure as soon as it occurs.

It is a challenging task to detect failures in large dynamic systems because failure events appear rarely and may not have fixed behavior. The high dimensionality of observation data due to the complexity of systems, together with the frequent changes of system normal behavior resulting from software upgrading, web content, and user behavior changes, makes detection even more difficult. The main contribution of this paper is to propose an *online dynamic tracking approach for high-dimensional data*, and apply it to monitor the component interactions and, hence, detect system failures. Since the tracking strategy does not require human intervention to adjust to normal system changes, our algorithm can be applied to the constantly changing Internet services.

Recently, component-based architectures (J2EE, .Net) have become prevalent in developing large-scale web applications. A component is an independent unit of software deployment with well-defined narrow interfaces that allow it to communicate with other components. To accomplish a specific type of user request, a sequence of component interactions is always activated, i.e., a

component dynamically calls or returns to another component to fulfil the service request. Such interactions usually follow certain *implicit* patterns that are determined by the business logic and functionality of system. When the system service suffers from failures, we believe that the component-interaction patterns will also get significantly affected. Therefore, we collect the frequencies of interactions between components as a feature and track such information over time to detect system failures.

One main obstacle in applying component interaction tracking is the high dimensionality of observations. For a system with l components, the dimension of interaction data is l^2 . However, we observe that in most real systems, each component only interacts with a small number of other components. This allows us to decompose the original high-dimensional space into signal and noise subspaces. Two important statistics, the *Hotelling T^2 score* and the *squared prediction error (SPE)* [1], are computed to represent the characteristics of data distribution in two subspaces. An online tracking algorithm, called *sequentially discounting expectation maximization (SDEM)* algorithm [2] is employed to learn the probability density of these two statistics. The anomaly of a new sample is then determined by comparing the density model before and after that sample is learned. Meanwhile, the signal and noise subspaces are also updated as time passes. In both the SDEM algorithm and subspace updating, an *exponentially weighted moving average (EWMA)* filter is employed to enhance the adaptivity of our algorithm to system changes. If an anomaly is detected, we identify the likely faulty components by statistically comparing the abnormal observation with normal ones.

The performance of our proposed detector has been tested on some synthetic data and a real web application based on the J2EE architecture. The reason of using synthetic data is that we can arbitrarily manipulate the dynamic nature of data and, thus, test the performance of our approach under scenarios that are difficult to simulate with real applications. The real interaction data is collected from *pet store*, an open source e-commerce application. We have used an instrumented application server to collect observations and also to inject a variety of failures. The detection and localization results show that our data reduction and tracking strategy gives good results for failure detection in distributed systems.

II. RELATED WORK

Traditional failure detections can be divided into two main categories: low-level and high-level detection techniques. The

Manuscript received August 8, 2005; revised November 22, 2005. This paper was recommended by Editor E. Trucco.

The authors are with NEC Laboratories America, Inc., Princeton, NJ 08540 USA (e-mail: haifeng@nec-labs.com; gfy@nec-labs.com; cristian@nec-labs.com; kenji@nec-labs.com).

Digital Object Identifier 10.1109/TSMCC.2007.897496

low-level detection techniques such as ping, heartbeat [3], and HTTP error code monitors [4] are easy to deploy, since they are application generic. But they cannot detect application level failures such as blank pages, wrong links, loops, and so on. These application-level service failures can be detected by some high-level detection techniques such as end-to-end tests of service functionality. However, those high-level techniques must be custom-built for each application and updated as the application evolves. An ideal detector would combine the ease of deployability found in low-level techniques with the more sophisticated detection capabilities found in high-level tools.

To achieve this goal, statistical learning approaches have been drawing a surge of interest recently due to their capabilities in mining large quantities of observations for interesting patterns that can be directly related to system high-level behavior. For instance, Sahoo *et al.* [5] collected various event logs from a large-scale cluster system, and applied temporal data mining and time series analysis to predict failure events of the system. Ide and Kasima [6] treated the web-based system as a weighted graph and applied graph mining techniques to monitor the graph sequences for failure detection. In the Magpie project [7], Barham and his colleagues used the stochastic context free grammar to model the request's control flow across multiple machines for the purpose of detecting component failures as well as localizing performance bottlenecks. The Pinpoint project [8], a close relative to Magpie, proposed two features for system-failure detection: request path shapes and component interactions. For the former, the set of seen traces was modeled with a probabilistic context free grammar (PCFG). The latter feature was used in building a profile for each component's interaction and using the χ^2 test for comparing the current distribution with the profile. In the same context of request shape analysis as in [8], [9] put forward a multiresolution abnormal trace detection algorithm using variable-length N-grams and automata. Bodik *et al.* [10] made use of the user access behavior as an evidence of the system's health, and applied several statistical approaches such as the Naive Bayes to mining such information for detecting application-level failures.

Most of the current approaches for failure diagnosis use event correlation [11], [12]. There are also many commercial tools that aid problem determination such as HP's OpenView [13] and IBM's Tivoli [14]. These methods typically rely on either human-generated rules or some known dependency models about the system. However, since the distributed systems are always complex and frequently changing, it is hard to directly obtain a meaningful model for diagnosis. To address these issues, Cohen *et al.* [15] presented a simplified Bayesian network to model the dependencies of 124 system variables and, hence, achieved automatic correlation between variables and the service-level agreement (SLA) violations. Aguilera *et al.* [16] and Brown *et al.* [17] relied on dynamic observations to adaptively build the dependency knowledge for diagnosis. An intelligent probing technology is proposed in [18] to actively localize system failures with the help of a Bayesian-based inference approach.

Our paper is inspired by the Berkley/Stanford ROC group's success in using traces for failure detection [8]. However, while

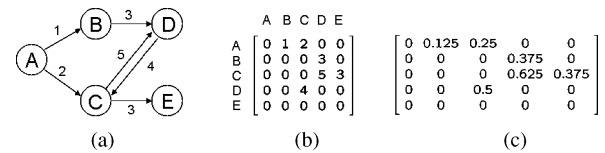


Fig. 1. Representation of component interactions. (a) Interaction of five components. (b) Matrix formalized by interaction frequencies. (c) Frequency matrix normalized by Frobenius norm.

they focus on developing the whole concept of recovery-oriented computing, in this paper we are interested in developing specific machine learning technique to explore the component interactions extensively. Compared with their way of building a profile for each component's interaction, our approach is more sophisticated and allows us to discover the dependencies between multiple interactions. The detailed comparison between our approach and the one in [8] is presented in Section V-B. Furthermore, we also believe that our method is applicable to observations other than component interaction (e.g., number of hits to individual web pages) and, in general, to applications where quantitative information is represented by high-dimensional data.

III. APPROACH TO FAILURE DETECTION

There are many techniques that enable us to collect the frequencies of component interactions from request traces [8], [16]. Fig. 1 presents an example of component interaction representation. There are five components in Fig. 1(a), and the number on the edge reveals the amount of interaction between two components within a time interval. For example, we can see that component B called component D three times. Such interaction frequencies can be formalized into a matrix as shown in Fig. 1(b), in which each row (or column) is related to the calling frequencies from a specific caller (or to a specific callee). This frequency matrix is normalized by its Frobenius norm (the square root of the sum of squares of each element), as shown in Fig. 1(c), in order to reduce the false positives introduced by workload changes. Also, for the ease of calculation, we transform the normalized matrix into a vector $\mathbf{x} \in R^l$ where l is the number of components in the system.

Our strategy is to track the interaction frequency \mathbf{x} over time, and regard the sudden change of new observation as a signal for system failure. To achieve this, a description of normality has to be learnt from system normal observations; failures are then identified by testing for new observation against this description. Since the dimension of data \mathbf{x} is usually high, it is hard to use common properties, such as density, to model its normal description. However, we observe that in many systems, each component only interacts with a limited number of other components to exchange information. That means that the observation \mathbf{x} is actually located in a low-dimensional subspace of the original space. Our idea is to find such low-dimensional subspace, and use the geometry features of data distribution in these subspaces to define the boundary between normal data and anomalies. Along this line, two statistics, the *Hotelling T^2 score* and *SPE*, are extracted from each sample to characterize the original data.

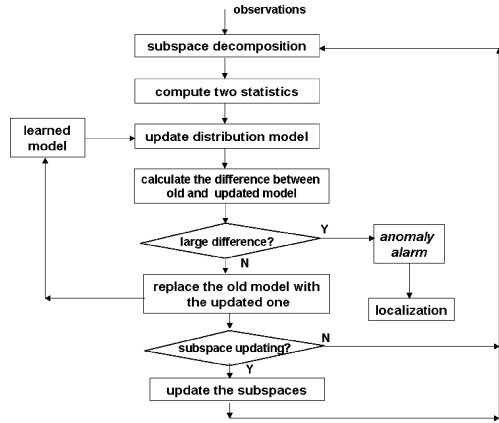


Fig. 2. Building blocks of our online detector.

We use the Gaussian mixture model to build the probabilistic density of extracted statistics. In the monitoring process, such density model is updated every time a new observation comes in. The failure is detected once the difference between the updated model and the previous one is larger than a predefined value. Since the system is frequently changing, an adaptive algorithm called SDEM algorithm is applied in updating the density model. We also update the signal and noise subspaces for every new observation or a batch of observations in order to adapt to system changes. The whole workflow of our failure detector is shown in Fig. 2. The detailed descriptions of its building blocks are presented in the following sections.

A. Statistics Used in Subspace Decomposition

Given a set of observations $\{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1}\}$, with $\mathbf{x}_i \in R^p$ observed at time t_0, t_1, \dots, t_{n-1} , the signal subspace S_s and noise subspace S_n spanned by those observations can be obtained either by *singular value decomposition* (SVD) of the data matrix X or by *eigen decomposition* of the data correlation matrix $C = \frac{1}{n}XX^T$.

The SVD of data matrix $X = [\mathbf{x}_0 \dots \mathbf{x}_{n-1}]$ is expressed as $X = U\Sigma V^T$, where $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_r, \sigma_{r+1}, \dots, \sigma_m) \in R^{p \times n}$, and $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_m$, with $m = \min\{p, n\}$. The two orthogonal matrices U and V are called the left and right eigen matrices of X . Based on the singular value decomposition, the subspace decomposition of X is expressed as

$$X = X_s + X_n = U_s \Sigma_s V_s^T + U_n \Sigma_n V_n^T \quad (1)$$

where the diagonals of Σ_s are large singular values $\{\sigma_1, \dots, \sigma_r\}$, and $\{\sigma_{r+1}, \dots, \sigma_n\}$ belong to the diagonals of Σ_n with $\sigma_r \gg \sigma_{r+1}$. The set of orthonormal vectors $U_s = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_r]$ forms the bases of signal space S_s . The projection matrix P_s onto the signal space would be given by $P_s = U_s U_s^T$. Since the noise subspace is the orthogonal complement of signal subspace, $S_n = S_s^\perp$, the projection onto noise subspace S_n can be written as $P_n = I - P_s$. Any vector $\mathbf{x} \in R^p$ can be represented by a summation of two projection vectors

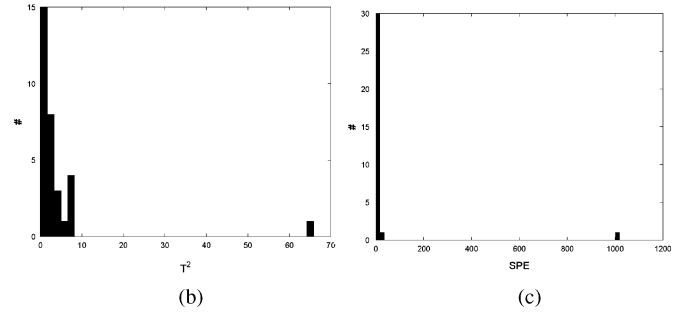
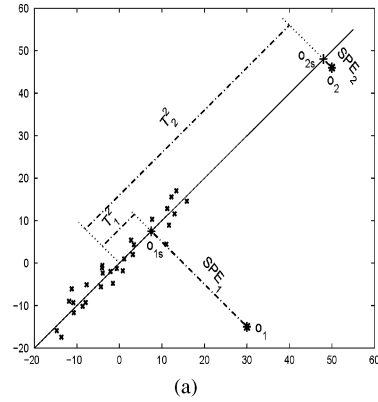


Fig. 3. Role of extracted statistics for failure detection. (a) SPE value of the abnormal sample o_1 is much larger than those of normal samples. The Hotelling T^2 score of the abnormal sample o_2 is much larger than those of normal samples. (b) Histogram of Hotelling T^2 score for all the samples. (c) Histogram of SPE value for all the samples.

from two subspaces S_s and S_n

$$\begin{aligned} \mathbf{x} &= \mathbf{x}_s + \mathbf{x}_n \\ &= P_s \mathbf{x} + (I - P_s) \mathbf{x}. \end{aligned} \quad (2)$$

The subspace decomposition can also be accomplished by eigen analysis of the correlation matrix C , which is expressed as

$$C = \frac{1}{n}XX^T = \frac{1}{n}U\Sigma^2U^T \quad (3)$$

where the columns of U are actually the eigenvectors of C , and the eigenvalues of C are related to the diagonals of matrix Σ .

Once the observations have been decomposed into signal and noise subspaces, we extract some statistics to describe the data distributions in two subspaces. One is the *Hotelling T^2 score* [1], which measures the variation of each sample in signal subspace. For a new sample vector \mathbf{x} , it is expressed as

$$T^2 = \mathbf{x}^T U_s \Sigma_s^{-1} U_s^T \mathbf{x}. \quad (4)$$

Another statistic SPE [1] indicates how well each sample conforms to the model, measured by the projection of sample vector on the noise subspace

$$SPE = \|P_n \mathbf{x}\|^2 = \|(I - P_s) \mathbf{x}\|^2. \quad (5)$$

The intention of using these two statistics for failure detection is illustrated in Fig. 3. In this figure, a set of 2-D normal samples marked as x are generated from a line (1-D subspace)

with certain noises. Through the process of subspace decomposition, the direction of the line (the signal subspace) is obtained. We then project each sample \mathbf{x} onto that line to get its noisy-free estimates \mathbf{x}_s . In this case, the value of Hotelling T^2 score actually represents the Mahalanobis distance from \mathbf{x}_s to the origin (0, 0). The value of SPE is the squared distance between \mathbf{x} and \mathbf{x}_s . Two abnormal samples, marked as * are also shown in Fig. 3(a). Since the sample o_1 is far away from the line, its SPE value is much larger than those of other points. While the sample o_2 has a reasonable SPE value, its Hotelling T^2 score is very large since its projection on the line o_{2s} is far away from the cluster of projected normal samples. We plot the histograms of the Hotelling T^2 score and SPE value for all the samples in Fig. 3(b) and (c), respectively. From these histograms, we conclude that by defining suitable boundaries for normal samples in the extracted statistics, we can find abnormal observations and, hence, detect failures.

In real situations, the distributions of Hotelling T^2 score and SPE are arbitrary and unknown. It is not reliable to set *fixed* thresholds for these statistics to distinguish normal and abnormal observations. Instead, we use the following strategy to model the two statistics and detect failures.

B. Online Detector

We denote the extracted two statistics as \mathbf{z} , and use a Gaussian mixture model to represent their probability density, $p(\mathbf{z}|\boldsymbol{\theta}) = \sum_{i=1}^k c_i p(\mathbf{z}|\boldsymbol{\mu}_i, \Lambda_i)$ where k is a positive integer, $c_i \geq 0$, $\sum_{i=1}^k c_i = 1$, and each $p(\mathbf{z}|\boldsymbol{\mu}_i, \Lambda_i)$ is a d -dimensional Gaussian distribution with density specified by mean $\boldsymbol{\mu}_i$, and covariance matrix Λ_i

$$p(\mathbf{z}|\boldsymbol{\mu}_i, \Lambda_i) = \frac{1}{(2\pi)^{d/2} |\Lambda_i|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{z} - \boldsymbol{\mu}_i)^\top \Lambda_i^{-1} (\mathbf{z} - \boldsymbol{\mu}_i)\right)$$

where $i = 1, \dots, k$ and $d = 2$. We set the parameter vector

$$\boldsymbol{\theta} = (c_1, \boldsymbol{\mu}_1, \Lambda_1, \dots, c_k, \boldsymbol{\mu}_k, \Lambda_k). \quad (6)$$

In order to adapt to the frequent system changes, we use the SDEM algorithm [2] to update the density model every time a new observation comes in. An important feature of SDEM is its utilization of an EWMA filter to adjust to nonstationary data sources, e.g., when drifting sources of time series are tackled. For instance, given a set of observations $\{x_1, x_2, \dots, x_n, \dots\}$, an online EWMA filter of the mean μ is expressed as

$$\mu_{n+1} = (1 - \rho)\mu_n + \rho x_{n+1} \quad (7)$$

where the forgetting parameter ρ dictates the degree of discounting previous examples. Intuitively, the larger the ρ is, the faster the algorithm can “age out” past examples.

The whole SDEM algorithm is presented in Fig. 4. Usually the initial parameters $c_i^{(0)} = 1/k$ and $\boldsymbol{\mu}^{(0)}$ are set so that they are uniformly distributed over the data space. For every sample \mathbf{z}_n , the $\boldsymbol{\theta}$ of its density model is updated by SDEM. It is to be noted that there is another parameter α , which is set between [1.0, 2.0], in the estimation of γ_i in order to improve the stability of solution. The computation time at each round is $O(d^3k)$ where d is the dimension of \mathbf{z} and k is the number of Gaussian

```

Step 1 Set  $c_i^{(0)}, \boldsymbol{\mu}_i^{(0)}, \bar{\boldsymbol{\mu}}_i^{(0)}, \Lambda_i^{(0)}, \bar{\Lambda}_i^{(0)}$  ( $i = 1, \dots, k$ ).
 $n := 1$ 
Step 2 /*Parameter Updating*/
while  $n \leq T$  ( $T$ : sample size)
  Read  $\mathbf{z}_n$ 
  for  $i = 1, 2, \dots, k$ 
     $\gamma_i^{(n)} := (1 - \alpha\rho) \frac{c_i^{(n-1)} p(\mathbf{z}_n|\boldsymbol{\mu}_i^{(n-1)}, \Lambda_i^{(n-1)})}{\sum_{i=1}^k c_i^{(n-1)} p(\mathbf{z}_n|\boldsymbol{\mu}_i^{(n-1)}, \Lambda_i^{(n-1)})} + \frac{\alpha\rho}{k}$ 
     $c_i^{(n)} := (1 - \rho)c_i^{(n-1)} + \rho\gamma_i^{(n)}$ 
     $\bar{\boldsymbol{\mu}}_i^{(n)} := (1 - \rho)\bar{\boldsymbol{\mu}}_i^{(n-1)} + \rho\gamma_i^{(n)} \mathbf{z}_n$ 
     $\boldsymbol{\mu}_i^{(n)} := \bar{\boldsymbol{\mu}}_i^{(n)} / c_i^{(n)}$ 
     $\bar{\Lambda}_i^{(n)} := (1 - \rho)\bar{\Lambda}_i^{(n-1)} + \rho\gamma_i^{(n)} \mathbf{z}_n \mathbf{z}_n^\top$ 
     $\Lambda_i^{(n)} := \bar{\Lambda}_i^{(n)} / c_i^{(n)} - \boldsymbol{\mu}_i^{(n)} \boldsymbol{\mu}_i^{(n)\top}$ 
   $n = n + 1$ 

```

Fig. 4. SDEM algorithm (with ρ, α, k given).

distributions. We suggest that the value of integer k should be chosen between 2 and 5 in usual cases.

In the online monitoring process, every time a new observation \mathbf{x}_n comes in, its statistics \mathbf{z}_n are calculated and the density of \mathbf{z} is updated. The anomaly is determined based on the statistical deviation of density distribution before and after \mathbf{z}_n is obtained. If we denote the two distributions as $p_{(z)}^{(n-1)}$ and $p_{(z)}^{(n)}$, our metric called *Hellinger score* is defined by

$$s_H(\mathbf{z}_n) = \int (\sqrt{p_{(z)}^{(n)}} - \sqrt{p_{(z)}^{(n-1)}})^2 dz. \quad (8)$$

Intuitively, this score measures how much the probability density function $p_{(z)}^{(n)}$ has moved from $p_{(z)}^{(n-1)}$ after learning \mathbf{z}_n . A higher score indicates that \mathbf{z}_n is an outlier with high probability. For the efficient computation of Hellinger score, refer to [2].

C. Subspace Updating

It does not bring enough benefits to update \mathbf{z} by keeping the subspaces fixed, since the two subspaces may also change. Therefore, we update the subspaces for every new observation or every batch of observations. Both the sample-wise and block-wise subspace updating algorithms are based on the eigen decomposition of data correlation matrix as shown in (3). If we choose to update the subspaces after k data samples are obtained, $B = [\mathbf{x}_{n+1}, \mathbf{x}_{n+2}, \dots, \mathbf{x}_{n+k}]$, the new correlation matrix is estimated by using the EWMA filter

$$C_{n+k} = (1 - \rho)C_n + \rho B B^\top \quad (9)$$

where the forgetting parameter ρ is introduced to discount the previous samples.

The new subspace is obtained by finding the modified eigen system $C_{n+k} = \tilde{U} \tilde{\Sigma} \tilde{U}^\top$, where $\tilde{\Sigma} = \text{diag}(\tilde{\lambda}_1, \dots, \tilde{\lambda}_p)$, and first r column vectors of \tilde{U} form the bases of the new signal subspace. Suppose $\tilde{\lambda}$ and $\tilde{\mathbf{u}}$ are the eigen pairs of C_{n+k} , we have

$$(C_{n+k} - \tilde{\lambda} I_p) \tilde{\mathbf{u}} = 0. \quad (10)$$

Common methods of solving (10) such as QR iteration would take computation complexity with magnitude $O(p^3)$, where p

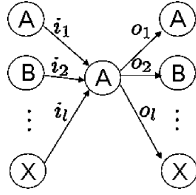


Fig. 5. Interaction behavior model for component A.

is the dimension of data. However, since we already have the eigen decomposition of C_n , such information can be utilized to speed up the computation. Substituting (9) into (10) gives

$$(C_n - \tilde{\lambda}I_p)\tilde{\mathbf{u}} + \tilde{B}\tilde{B}^\top\tilde{\mathbf{u}} = 0 \quad (11)$$

where $\tilde{B} = \sqrt{\frac{\rho}{1-\rho}}B$. If we introduce a vector $\mathbf{y} = \tilde{B}^\top\tilde{\mathbf{u}} \in R^k$, the following system of equations is induced:

$$(C_n - \tilde{\lambda}I_p)\tilde{\mathbf{u}} + \tilde{B}\mathbf{y} = 0 \quad (12)$$

$$\tilde{B}^\top\tilde{\mathbf{u}} - \mathbf{y} = 0. \quad (13)$$

Solving $\tilde{\mathbf{u}}$ in terms of \mathbf{y} in (12) and substituting $\tilde{\mathbf{u}}$ into (13) gives

$$F(\tilde{\lambda})\mathbf{y} = 0 \quad (14)$$

where

$$F(\tilde{\lambda}) = \mathbf{I}_k + \tilde{B}^\top(C_n - \tilde{\lambda}I_p)^{-1}\tilde{B}. \quad (15)$$

In this paper, we utilize the matrix $F(\tilde{\lambda})$ to find the eigen pairs of C_{n+k} .

For the sample-wise subspace updating ($k = 1$), $F(\tilde{\lambda})$ is a scalar. The eigenvalues $\tilde{\lambda}_i$ of C_{n+k} can be obtained by finding the roots of $F(\tilde{\lambda}) = 0$. Various iterative search algorithms such as the bisection method can be applied. Once the eigenvalues are known, the corresponding eigenvectors $\{\tilde{\mathbf{u}}_i\}$ are calculated from (11).

For the block-wise subspace updating ($k > 1$), the eigenvalues of C_{n+k} can be located by means of the inertia [19] of matrix $F(\tilde{\lambda})$, which are the numbers of negative, zero, and positive eigenvalues of $F(\tilde{\lambda})$. The eigenvectors of C_{n+k} are evaluated efficiently in two steps. First, we solve the intermediate vector \mathbf{y} from (14). The eigenvector $\tilde{\mathbf{u}}$ is then computed explicitly using (12). Applying the sample-wise or block-wise subspace updating algorithm reduces the computation complexity from $O(p^3)$ to $O(p^2)$.

IV. FAILURE LOCALIZATION

After the failure is detected, accurate localization makes it possible to initiate repair actions quickly, increasing the availability of the system. In order to localize the most suspicious components, we first transform the component interaction vector into a set of vectors, one for each component. A score is then evaluated for each component, based on the deviation of failure observation from its normal model. The components with the highest scores are the most suspicious. For instance, as shown in Fig. 5, the behavior of component A is represented as a set of links by which the paths enter A from other components or leave

A to other components. We use a vector $\mathbf{v}_A \in R^{2l}$ to represent the behavior $\mathbf{v}_A = [v_{A_{i1}}, \dots, v_{A_{il}}, v_{A_{o1}}, \dots, v_{A_{ol}}]^\top$, where l is the number of components in the system. Under the assumption that all the variables in \mathbf{v}_A are mutually independent, given a set of normal observations, we obtain the mean and standard deviation of each variable in \mathbf{v}_A

$$\mathbf{m}_A = [m_{A_{i1}}, \dots, m_{A_{il}}, m_{A_{o1}}, \dots, m_{A_{ol}}]^\top \quad (16)$$

$$\boldsymbol{\sigma}_A = [\sigma_{A_{i1}}, \dots, \sigma_{A_{il}}, \sigma_{A_{o1}}, \dots, \sigma_{A_{ol}}]^\top. \quad (17)$$

Given the failure observation, the anomaly score of component A is calculated as the sum of weighted deviation of each link frequency with respect to its normal mean

$$s_A = \sum_{s=1}^l \frac{(v_{A_{is}} - m_{A_{is}})^2}{\sigma_{A_{is}}^2} + \sum_{s=1}^l \frac{(v_{A_{os}} - m_{A_{os}})^2}{\sigma_{A_{os}}^2}. \quad (18)$$

In the same way, we calculate the scores of all other components. In fact, the score s_A satisfies χ^2 distribution with degree of freedom $2l$. By choosing a certain significance level, we obtain a threshold to separate faulty and normal components. It is to be noted that here we only use one failure observation \mathbf{x}_f for localization. If more failure observations from the same accident are available, we can use some voting strategies [20] to increase the confidence of localization.

V. EXPERIMENTAL RESULTS

In this section, we first use some synthetic data to compare the performances of tracking algorithms with and without subspace updating. Then, we apply our failure detector to a real system developed on the J2EE platform. The detection and diagnosis results on a variety of injected failures show the effectiveness of our approach.

A. Synthetic Data

The advantage of using synthetic data is that we can arbitrarily manipulate the dynamic nature of the data that are used to compare the performances of online detectors with and without subspace updating. We call the detection technique without subspace updating as “fixed subspace tracking,” and detection with subspace updating as “dynamic subspace tracking.”

Each normal observation is obtained by sampling a harmonic sinusoid curve with random shift u_0

$$x(u) = A \sin \frac{2\pi f(u + u_0)}{T_0} + \epsilon \quad (19)$$

where A is the maximum amplitude, f is the frequency, T_0 is the period of the signal, u_0 is a uniformly generated random number between $[0, T_0)$, and ϵ is the noise. In the experiment, we choose $A = 10$, $T_0 = 100$, and the curve is sampled at position $u = 1, 2, \dots, T_0$. Then, each observation is with 100 dimensions. It is easy to see that the signal space of those observations is spanned by samples from two curves $\left\{ \sin \frac{2\pi fu}{T_0}, \cos \frac{2\pi fu}{T_0} \right\}$. The abnormal data are also generated from sinusoid functions, but the function is truncated when its absolute amplitude is larger than a threshold, which is randomly generated between $[3, A]$.

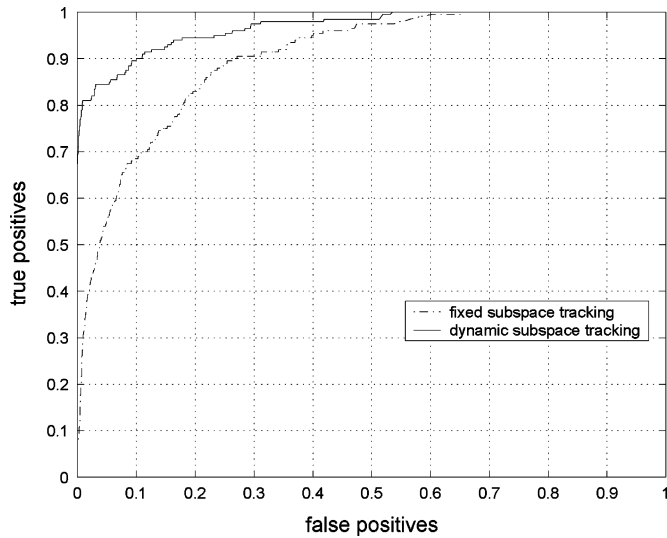


Fig. 6. ROC curves of online detectors with and without subspace updating.

We added Gaussian noise with standard deviation $\sigma = 0.5$ on both normal and abnormal signals.

We generate 1000 normal observations as the training data and 1100 test samples. To truly simulate the situation of dynamic systems, we create nonstationary test data by sampling the curves with a continuously varying frequency f , whereas the training data are from the curves with fixed frequency $f = 1$. Twenty abnormal observations are produced in the test data, which appear every 50 observations between index 50 and 1000. We then run the online tracking algorithms with and without subspace updating separately. The performance of each algorithm is represented by its *receiver operating characteristic* (ROC) curve, which is a plot of the true positive rate against the false positive rate for the different possible cut points of the final tracking scores. We repeat such random data generation and performance comparison 100 times. Fig. 6 plots the average ROC curve of the 100 trials for each algorithm. It illustrates that the dynamic subspace tracking performs much better than fixed subspace tracking in case of nonstationary observations.

B. Real Application

Our approach is also tested on a real e-commerce application to detect and localize actual failures. We use the same mechanism as proposed in [8] to modify the JBoss application server and collect the component interaction activity. JBoss is an open-source implementation of J2EE, which is a widely adopted middleware standard for constructing enterprise applications from reusable Java modules called Enterprise Java Beans (EJBs). Pet Store 1.3.1 is deployed as our testbed application. Its functionality consists of store front, shopping cart, purchase tracking, and so on. There are 47 components in Pet Store, including EJBs and Servlets. We built a client emulator to generate workloads similar to that created by typical user behavior. The emulator generates a varying number of concurrent client connections. Each client simulates a session, which consists of a series of requests such as creating new accounts, searching,

browsing for item details, updating user profiles, placing the order, and checking out. The component interactions are monitored under these simulated workloads.

Two types of component faults are simulated in the experiment, *expected exception* and *null call* [8]. The first failure happens when a method declaring exceptions (which appear in the method's signature) is invoked. In this situation, an exception is thrown without the method's code being executed. Even though applications are expected to mask such exceptions from the end user, it is still possible in real situations that they are not handled well and, consequently, turn into run-time failures. The *Null call* failure causes all methods in the affected component to return a null value, again without executing the methods. These bugs can easily happen in practice due to incomplete or incorrect handling of rare conditions. We injected each type of faults into 15 EJB components of Pet Store to simulate a variety of failures. For example, injecting a null call fault into component "AccountEJB" would prevent a customer from seeing his account information on the related web page. Similarly, injecting an expected exception fault into the component "SignOnEJB" would prevent a client from creating a new account.

To test the effectiveness of our failure-detection algorithm, we collect 1000 observations under system's normal operation as the training data. Each observation reflects the number of component interactions in PetStore under a simulated workload, which obeys the Gaussian distribution with mean 30 and standard deviation 6, $N(30, 6)$. Since the standard deviation of this distribution is relatively large, the workloads for the failure cases are totally different with the one that generates normal samples even though they share the same distribution. Each time after a simulated failure is injected, we collect one failure observation. It is to be noted that the system is restarted before every failure injection in order to remove the impacts of the previously injected failures. We then collected 30 failure observations. Each observation corresponds to a specific failure. We also collect 770 normal observations to form 800 test samples. The failure samples appear every 20 observations in the test data between index 110 and 690.

We first apply subspace decomposition of the training data, and calculate the Hotelling T^2 score and SPE value for each training sample. Based on the values of these two statistics, we build their density model with the number of Gaussians $k = 3$. In the online monitoring process, every time a new observation comes in, its Hotelling T^2 score and SPE value are computed. Fig. 7(a) and (b) show the values of these two statistics for all the test samples. We also update the density of statistics for each test sample by using the SDEM algorithm. Each time the density is updated, we calculate the tracking score (8) that represents the difference of the density model before and after the update. As shown in Fig. 7(c), the online tracking scores provide a more reliable indication about failure than those directly relying on the two statistics. Actually, if we choose the threshold 50, we can detect 28 failures with only one false positive. In online monitoring, the datum is sampled in 5-s interval, and our detection algorithm meets the real-time requirement of practical applications. It is also important to realize that our injected failures are hard to be detected by traditional techniques, since

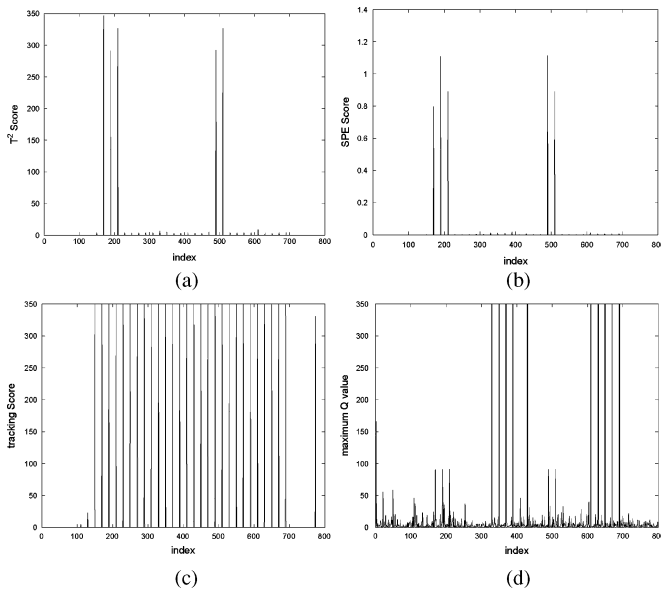


Fig. 7. Experimental results in a real Internet service. (a) T^2 scores. (b) SPE scores. (c) Online tracking scores. (d) Maximum χ^2 values among all components.

they produce very subtle outcomes and do not cause exceptions to be printed on the operator's console, thus, avoiding crashing the application software.

The performance of our algorithm is also compared with the χ^2 scores proposed in [8], in which the detection is based on the statistical test for each component. Chen *et al.* [8] built a template of normal behavior for every component, which is expressed as the expected interactions of all its links. We use the average of training samples to build the template of each component. For every test observation, the Q values [8] of all components are calculated. We extract the maximum Q value for every test sample and plot them in Fig. 7(d). It can be seen from Fig. 7(c) and (d) that our technique got a much higher detection rate with fewer false positives. There are several reasons to account for this. First, while in [8] all the interaction links are regarded as independently distributed, our approach captures the dependencies between them. For instance, if there exists a rule that *whenever component A calls component B, B always calls component C and no other components call B and C*, then the frequencies of the interaction "A calls B" and "B calls C" should be the same. Such information is taken into account in the subspace decomposition and implicitly embedded in the two statistics, whereas it is lost when separate profiles are built. Another advantage is that our algorithm can dynamically adapt to the changes in the simulated workload.

After each failure has been detected, every component will get an anomaly score for that failure. Fig. 8 shows the localization scores of the failure injected components for all the failure cases. According to (18), the localization score satisfies χ^2 distribution with a degree of freedom $2l$. If we choose a level of significance 0.005 to determine the threshold, we can localize the failure injected components in 22 of the 30 simulated failure cases. However, we also notice that for some simulated

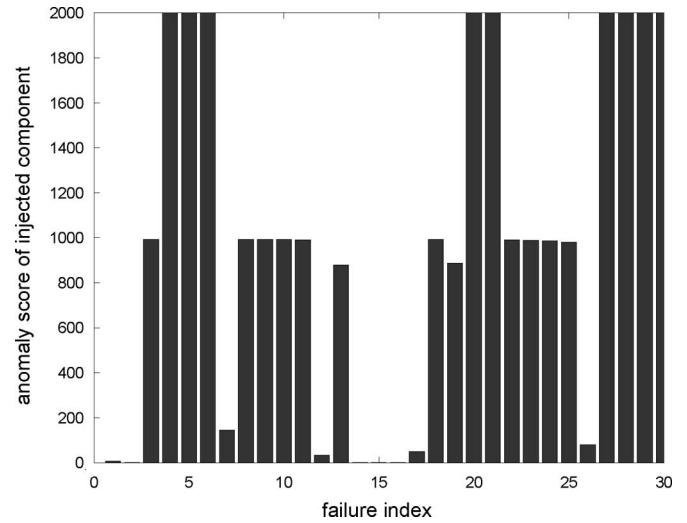


Fig. 8. Localization score of the failure injected component in each failure case.

failures, the injected component is not the component with the highest anomaly score. This is because of the cascading effects of failures. That is, when a component is failure injected, not only itself but also other components, e.g., its neighbors, will change the interaction behaviors significantly. Our strategy can only localize a cluster of suspicious components. In order to further localize the faulty components, other information about the system has to be utilized. For instance, in the Pet Store testbed system, when both the "SignOnEJB" and "Servlet" components have highest localization scores, we found that usually the "SignOnEJB" component will be analyzed first because all of the business logic of Pet Store is implemented in the EJB components.

VI. CONCLUSION

We have presented a failure-detection technique that can reduce the dimensionality of observation data and adapt to the system changes. It has been applied to monitor the interactions among different components in distributed systems. Experiments show that our method can detect a variety of service failures in a J2EE-based web application. With these results, we expect to apply this approach to more complex systems to get further confidence in our technique. In addition, we plan to investigate the tradeoffs involved in extracting more statistics (instead of two as in this paper) from highdimensional data.

ACKNOWLEDGMENT

The authors would like to thank the Berkley/Stanford ROC group for providing Pinpoint [8], which was used to collect observations from Pet Store. The authors also would like to thank K. Yamanishi for the software to implement the SDEM algorithm. Finally, the authors would like to thank P. Ashar, H. P. Graf, K. Yamanishi, and the anonymous reviewers for their comments and suggestions for the improvement of this paper.

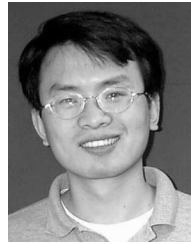
REFERENCES

- [1] I. T. Jolliffe, *Principal Component Analysis*. New York: Springer-Verlag, 1986.
- [2] K. Yamanishi, J. Takeuchi, G. Williams, and P. Milne, "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discovery Data Min.*, 2000, pp. 320–324.
- [3] M. K. Aguilera, W. Chen, and S. Toueg, "Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks," *Theoret. Comput. Sci.*, vol. 220, pp. 3–30, 1999.
- [4] E. Marcus and H. Stern, *Blueprints for High Availability*. New York, NY: Wiley, 2000.
- [5] R. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam, "Critical event prediction for proactive management in large-scale computer clusters," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Min.*, Washington, DC, 2003, pp. 426–435.
- [6] T. Idé and H. Kashima, "Eigenspace-based anomaly detection in computer systems," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Min.*, Seattle, WA, Aug. 2004, pp. 440–449.
- [7] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, "Magpie: Real-time modelling and performance-aware systems," presented at the 9th Workshop Hot Topics Oper. Syst., Lihue, HI, May 2003.
- [8] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic systems," presented at the Int. Perform. Dependability Symp., Washington, DC, Jun. 2002.
- [9] G. Jiang, H. Chen, C. Ungureanu, and K. Yoshihira, "Multi-resolution abnormal trace detection using varied-length n-grams and automata," in *Proc. 2nd Int. Conf. Autonom. Comput.*, Seattle, WA, Jun. 2005, pp. 111–122.
- [10] P. Bodik, G. Friedman, and L. Biewald *et al.*, "Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization," in *Proc. 2nd Int. Conf. Autonom. Comput.*, Seattle, WA, Jun. 2005, pp. 89–100.
- [11] A. Bouloutas, S. Calo, and A. Finkel, "Alarm correlation and fault identification in communication networks," *IEEE Trans. Commun.*, vol. 42, no. 2, pp. 523–533, Feb. 1994.
- [12] I. Rouvellou and G. W. Hart, "Automatic alarm correlation for fault identification," in *Proc. INFOCOM*, 1995, pp. 553–561.
- [13] Hewlett-Packard Corporation H.P. openview, [Online]. Available: <http://www.openview.hp.com/>
- [14] IBM, Tivoli business system manager, [Online]. Available: <http://www.tivoli.com/>
- [15] I. Cohen, S. Jeffrey, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," presented at the 6th Symp. Oper. Syst. Des. Implement., San Francisco, CA, Dec. 2004.
- [16] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthicharoen, "Performance debugging for distributed systems of black boxes," in *Proc. 19th ACM Symp. Oper. Syst. Principles*, Bolton Landing, NY, 2003, pp. 74–89.
- [17] A. Brown, G. Kar, and A. Keller, "An active approach to characterizing dynamic dependencies for problem determination in a distributed application environment," in *Proc. 7th Int. IFIP/IEEE Symp. Integr. Manag.*, Seattle, WA, 2001, pp. 377–390.
- [18] M. Brodie, I. Rish, and S. Ma, "Intelligent probing: A cost-efficient approach to fault diagnosis in computer networks," *IBM Syst. J.*, vol. 41, pp. 372–385, 2002.
- [19] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: The John Hopkins Univ. Press, 1996.
- [20] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Mach. Learning*, vol. 36, no. 1, pp. 105–139, 1999.



Haifeng Chen received the B.Eng. and M.Eng. degrees in automation from Southeast University, Nanjing, China, in 1994 and 1997, respectively, and the Ph.D. degree in computer engineering from Rutgers University, Newark, NJ, in 2004.

He was a Researcher in the Chinese National Research Institute of Power Automation, Nanjing. He is currently a Research Staff Member with NEC Laboratories America, Inc., Princeton, NJ. His research interests include data mining, autonomic computing, pattern recognition, and robust statistics.



Guofei Jiang received the B.S. and Ph.D. degrees in electrical and computer engineering from the Beijing Institute of Technology, Beijing, China, in 1993 and 1998, respectively.

From 1998 to 2000, he was a Postdoctoral Fellow in computer engineering at Dartmouth College, Hanover, NH. He is currently a Research Staff Member with the Robust and Secure Systems Group at NEC Laboratories America, Inc., Princeton, NJ. His current research interests include distributed systems, dependable and secure computing, and system and information theory. He is the author of nearly 50 publications.

Dr. Jiang is an Associate Editor of IEEE SECURITY AND PRIVACY MAGAZINE. He has participated in the program committees of many prestigious conferences.



Cristian Ungureanu received the Diploma in computer science from Bucharest Polytechnic Institute, Bucharest, Romania, in 1987 and the Ph.D. in computer science from New York University, New York, in 1998.

He is currently a Senior Research Staff Member at NEC Laboratories America, Inc., Princeton, NJ. His research interests include robust and secure distributed storage and automatic failure detection in distributed systems.



Kenji Yoshihira received the B.E. degree in electrical engineering from the University of Tokyo, Tokyo, Japan, in 1996 and the M.S. degree in computer science from New York University, New York, in 2004.

He was with Hitachi Ltd., Tokyo, for five years, where he designed processor chips. He then joined Investoria, Inc., Tokyo, Japan, to develop an Internet service system for financial information distribution until 2002. He is currently a Research Staff Member with the Robust and Secure Systems Group at NEC Laboratories America, Inc., Princeton, NJ. His current research interests include distributed system and autonomic computing.

His current research interests include distributed system and autonomic computing.