Toward Automated Anomaly Identification in Large-Scale Systems

Zhiling Lan, *Member*, *IEEE Computer Society*, Ziming Zheng, *Student Member*, *IEEE*, and Yawei Li, *Member*, *IEEE*

Abstract—When a system fails to function properly, health-related data are collected for troubleshooting. However, it is challenging to effectively identify anomalies from the voluminous amount of noisy, high-dimensional data. The traditional manual approach is time-consuming, error-prone, and even worse, not scalable. In this paper, we present an automated mechanism for node-level anomaly identification in large-scale systems. A set of techniques is presented to automatically analyze collected data: data transformation to construct a uniform data format for data analysis, feature extraction to reduce data size, and unsupervised learning to detect the nodes acting differently from others. Moreover, we compare two techniques, principal component analysis (PCA) and independent component analysis (ICA), for feature extraction. We evaluate our prototype implementation by injecting a variety of faults into a production system at NCSA. The results show that our mechanism, in particular, the one using ICA-based feature extraction, can effectively identify faulty nodes with high accuracy and low computation overhead.

Index Terms—Anomaly identification, large-scale systems, independent component analysis, principal component analysis, outlier detection.

1 Introduction

1.1 Motivation

It has been widely accepted that failures are ongoing facts of life to be dealt with in large-scale systems. Studies have shown that in production systems, failure rates are as high as more than 1,000 per year, and depending on root cause of the problem, the average failure repair time ranges from a couple of hours to nearly 100 hours [14], [27]. Every hour that a system is unavailable can cause undesirable loss of processing cycles, as well as substantial maintenance cost.

When a system fails to function properly, health-related data are collected across the system for troubleshooting. Unfortunately, how to effectively find anomalies and their causes in the data has never been as straightforward as one would expect. Traditionally, human operators are responsible of examining the data with their experience and expertise. Such manual processing is time-consuming, error-prone, and even worse, not scalable. As the size and complexity of computer systems continue to grow, so does the need for automated anomaly identification.

To address the problem, in this paper, we present an automated mechanism for *node-level anomaly identification*. Different from fine-grained root cause analysis that aims to identify the root causes of problems or faults, ours is a coarse-grained problem localization mechanism focusing on detecting culprit node(s) by automatically examining the health-related data collected across the system. By finding the abnormal nodes, system managers are able to know where to fix the problem and application users can take relevant actions to avoid or mitigate fault impact on their applications. Following the terminology used in the dependability literature[6], a fault—like a hardware defect or a software flaw—can cause system node to transit from a normal state to an error state, and the use of the node in the error state can lead to node failure. Hence, we seek to discover the nodes in error or failed states, which are also called *abnormal states* in the paper; we regard these nodes as *anomalies* that require further investigation.

1.2 Technical Challenges

Finding anomalies is a daunting problem, especially in systems composed of vast amount of nodes. We classify the key challenges into four categories as follows:

- *Data diversity.* Depending on the monitoring tools used, the data collected often have different formats and semantics, thereby making it hard to process them in a uniform way.
- *Data volume.* Due to the size of modern systems, data collected for analysis are characterized by their huge volume, e.g., in order of gigabytes per day [14]. Finding anomalies from such potentially overwhelming amount of data is like finding needles in a haystack.
- *Data dependency.* Most measured data are mixtures of independent signals and they often contain noises. A naive method that directly compares the measured data for anomalies is generally inaccurate by producing substantial amount of false alarms.
- *Anomaly characteristics.* In large-scale systems, anomaly types are many and complex. Moreover, node behaviors are dynamically changing during operation

- *Z. Lan and Z. Zheng are with the Department of Computer Science, Illinois Institute of Technology, 10 W. 31st Street, Chicago, IL 60616. E-mail: {lan, zzheng11}@iit.edu.*
- *Y. Li is with Google, Inc., 1600 Amphitheatre Pkwy, Mountain View, CA 94043. E-mail: liyawei@iit.edu.*

Manuscript received 19 Aug. 2008; revised 8 Feb. 2009; accepted 13 Mar. 2009; published online 23 Mar. 2009.
Recommended for acceptance by M. Yamashita.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-08-0314.
Digital Object Identifier no. 10.1109/TPDS.2009.52.

as system nodes are dynamically assigned to different tasks. Thus, it is extremely difficult to precisely define the normal behaviors of system nodes.

1.3 Paper Contributions

The primary contribution of this paper lies in a collection of techniques to find abnormal nodes (i.e., anomalies) via automated analysis of system data. Specifically, *data transformation* is first employed to manage data diversity by reducing the problem of anomaly identification with different data types to the problem of finding outliers in a new space of a single data type (Section 3.1). Second, to address the sheer data volume and inherent data dependency, *feature extraction* is introduced to convert the multi-dimensional data into a space of lower dimensions for quick and better analysis (Section 3.2). In particular, we present and compare two techniques, namely principal component analysis (PCA) and independent component analysis (ICA), for feature extraction. In addition to the main objective of this paper—to design an automated anomaly identification mechanism for large-scale systems, another purpose of this study is to compare PCA and ICA-based feature extractions and further to point out their strengths and limitations. Finally, unlike the traditional methods that require samples of both normal and abnormal data for training, *outlier detection* automatically extracts the expected normal behavior from the data and flags significant deviations from the expected as anomalies (Section 3.3). Together, these techniques form an unsupervised learning approach, which can address the unknown and dynamic characteristics of anomalies.

We evaluate our prototype system to identify system faults that are manually injected into a production system at the National Center for Supercomputing Applications (NCSA). Under a variety of faults injected, the automated mechanism using ICA-based feature extraction is capable of detecting every anomalous node, with a false alarm rate less than 8 percent. Our framework incurs very low runtime overhead. In our experiments, it takes less than 0.25 second to identify up to 20 anomalous nodes out of 256 nodes. We believe the high accuracy and time efficiency attribute to the novel use of feature extraction, which can significantly reduce data size by 96 percent. Our results also suggest that the ICA-based mechanism is a promising approach to automate anomaly identification in large-scale systems.

While the proposed mechanism mainly targets on failure diagnosis, it can also be integrated with tools like that of [44], [45] to aid failure prediction by pinpointing the potential locations of system failures.

1.4 Paper Outline

The rest of the paper is organized as follows: Section 2 gives an overview of our methodology, followed by a detailed description in Section 3. In Section 4, we describe our experiments and list experimental results. Section 5 discusses the benefits and limitations of the proposed mechanism. A brief discussion of related works is presented in Section 6. Finally, we conclude the paper in Section 7.

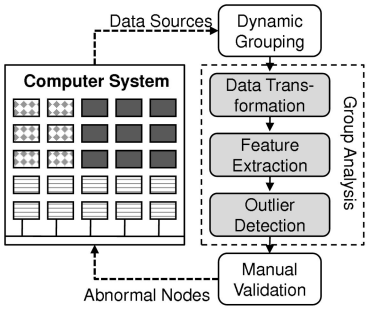


Fig. 1. Overview of anomaly identification. Node grouping is needed to dynamically divide system resources into groups and the nodes in the same group are expected to exhibit similar behaviors. Three group analysis techniques, namely data transformation, feature extraction, and outlier detection, are applied per group to find abnormal nodes (i.e., anomalies). These anomalies can manually be validated by system administrators. Note that the shaded boxes indicate the major contributions of this study (Sections 3.1-3.3).

2 Methodology Overview

"Computers are incredibly fast, accurate and stupid; humans are incredibly slow, inaccurate and brilliant; together they are powerful beyond imagination."—Albert Einstein

Drawing on the wisdom in the above quotation by Albert Einstein, we propose an anomaly identification framework as shown in Fig. 1 in which computers are utilized to process massive quantity of data by leveraging pattern recognition technologies and human can focus on final validation. Health-related data are collected across the system and sent for automated data analysis that includes dynamic grouping and group analysis. The automated mechanism can be triggered either periodically with a predefined frequency or by a system monitoring tool in case of unusual events. In this paper, we focus on detecting anomalies in homogeneous collection of nodes (also called "groups"), and heterogeneous collections will be addressed by grouping in the future. The resulting list of anomalies will be sent to system administrators for final validation. As we will see later, by combining the fast processing capability of computers with human expertise, the proposed mechanism can quickly discover anomalies with a very high accuracy.

Our group analysis is based on two key observations. First, the nodes performing comparable activities generally exhibit similar behaviors [5], [24], [33]. Second, faults are rare events, so the majority of system nodes are functioning normally. For each group, three tightly coupled techniques are applied to find the nodes that exhibit different behaviors from the majority as follows:

1. *Data transformation.* It is responsible for collecting relevant data across the system and assembling them into a uniform format called *feature matrix* (generally in high dimensionality). Here, a *feature* is defined as any individually measurable variable of the node being observed, such as CPU utilization, available memory size, I/O, network traffic, etc.
2. *Feature extraction.* A feature extraction technique, such as PCA or ICA, is applied on the feature matrix to generate a matrix with much lower dimensionality,

while keeping the most relevant information in the data. This not only accelerates data analysis by reducing data dimensionality but also improves data analysis by removing inherent data dependency.

3. *Outlier detection.* It determines the nodes that are "far away" from the majority as potential anomalies. By analyzing the low-dimensional matrix produced by feature extraction, a cell-based algorithm is used to quickly identify the outliers.

To truly make the above mechanism useful in practice, we intend to provide two guarantees in the design of our anomaly identification system. First is *high accuracy*, in terms of low false alarm rate and extremely low missed rate (i.e., close to zero). Second is *time efficiency*, meaning to quickly identify faulty nodes from the data collected, no matter how large are the data. The first is essential for its acceptance by the community, while the second is needed for quick detection and timely response.

3 Description

In this section, we present the details of our group analysis mechanism.

3.1 Data Transformation

The goal of this step is to gather system data for representing node behaviors and then transform the data into a uniform format for data analysis. A fault typically induces changes in multiple subsystems of a node, such as CPU, memory, I/O and network. For example, a memory leaking may affect the amount of free memory and the CPU utilization rate; an operation to a malfunctioning disk may lead to huge IO time and long CPU idle time. Hence, in order to cover a broad fault space, it is necessary to collect and store feature data from all the subsystems per node. Furthermore, it is beneficial to track and store the tendencies of these features by collecting multiple samples. Note that a *feature* is defined as any individually measurable variable of the node being observed.

Modern computer systems are deployed with various health monitoring tools. On the hardware side, hardware sensors are widely deployed to monitor device attributes, such as CPU temperature and fan speed [18], [19], [22]. On the software side, most operating systems provide various event logs and/or system calls to track memory, processes, network traffic, paging, and block I/O information. In addition, a number of third party software tools have been developed for system monitoring, such as Ganglia, Supermon, Inca, splunk, and innumerous others [20], [39], [40], [41]. From the application level, there also exist tools to collect application performance data through source-based profiling or dynamic instrumentation [24]. The data collected by these tools can be used by our framework.

As data collected from different tools are in arbitrary formats, data preprocessing is needed. Possible preprocessing includes converting variable-spaced time series to constant-spaced ones, filling in missing samplings, generating real-value samples from system logs, and removing period spikes or noises [33], [35], [38].

Fig. 2 describes data preprocessing and transformation whose goal is to construct feature matrix for data analysis. Let

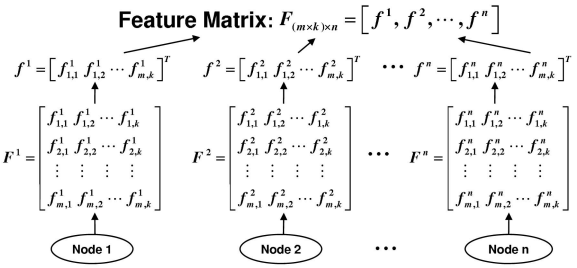


Fig. 2. Data transformation. Here, n is the number of nodes, m is the number of features, and k is the number of samples per feature. F^i represents the feature matrix of the ith node and F denotes the feature matrix of the group. It enables us to generate a uniform data type called feature matrix from diverse data sources for data analysis.

m be the number of features collected from n nodes. In order to capture the tendencies of these features, k snapshots are sampled per node. As a result, there are n matrices $F^i (i = 1, 2, \ldots, n)$, each representing the feature matrix collected from the ith node. In each matrix F^i, the element $f^i_{h,j}$ denotes the value of feature h collected at the jth snapshot, where $1 \leq j \leq k$ and $1 \leq h \leq m$. The elements in the matrices are supposed to be preprocessed as mentioned above.

To facilitate data analysis, we reorganize each matrix F^i into a long $(m \times k)$ column vector $f^i = [f^i_{1,1}\ f^i_{1,2} \cdots f^i_{m,k}]^T$. We then construct a single large matrix as the feature matrix for the group:

$$F = [f^1, f^2, \ldots, f^n]. \quad (1)$$

The transformation from a multiway matrix to a single large matrix makes it easy to diagnose anomalies across different nodes.

3.2 Feature Extraction

After constructing feature matrix, a simple method is to directly compare the data for anomalies. However, this simple comparison can lead to high miss rate and false alarm rate, as we will see in Section 4. There are several reasons for the ineffectiveness. First, useful information may be hidden between features and it is hard for direct comparison to discover the hidden information. Second, some features have large fluctuation even when they are collected from normal nodes, while other features may be close to each other even when they are collected from the nodes in different states (i.e., normal versus abnormal states). Third, noise often exists in the data, thereby distorting the meaning of the data. Even worse, the fluctuation and noise may propagate across multiple samples, thereby impacting the accuracy of anomaly localization. Finally, analyzing high-dimensional matrices is time-consuming. To address these issues, *feature selection* and *feature extraction* are two well-known techniques.

Feature selection is a technique to select an optimal subset of the original features based on some criteria [16]. However, feature selection is a supervised learning, requiring a priori knowledge of both normal and abnormal behaviors. Moreover, an optimal selection requires an exhaustive search of all the possible subsets of features, thereby being time-consuming. In general, feature selection works well with irrelevant features. As mentioned before, a fault typically induces changes in multiple subsystems, and

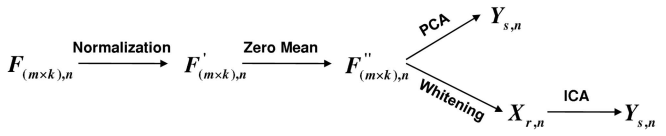


Fig. 3. Feature extraction. Here, F is the feature matrix obtained via data transformation (see Fig. 2). Both normalization and zero-mean procedures are needed before applying PCA and ICA.

hence, is reflected in multiple features. In other words, these features are not irrelevant, thereby greatly limiting the effectiveness of feature selection.

Unlike feature selection, *feature extraction* transforms the original feature space into a space of fewer dimensions in which the information most relevant to the problem is preserved [16]. The data presented in a low-dimensional subspace are easier to separate into different classes. Further, as we will see in Section 4, by transforming the data into a new space that keeps the most relevant information, feature extraction can also improve analysis accuracy.

In this paper, we investigate two feature extraction techniques: 1) PCA and 2) ICA. Both are well-known pattern recognition techniques. Both belong to unsupervised learning, meaning that we do not need to know the data label, e.g., whether it is normal or abnormal. Therefore, they work well for unknown anomaly patterns. Another advantage of these techniques is that in the new space, the difference between the classes of data is amplified, while the difference within the same class becomes less. This can greatly help to separate anomalies from normal nodes. While PCA has been explored for problem diagnosis and application classification [25], [26], ICA has been mainly used in the field of signal processing or face recognition [30], [32]. As mentioned before, one goal of this study is to compare PCA- and ICA-based anomaly identification mechanisms.

Fig. 3 presents PCA- and ICA-based feature extractions on feature matrix F obtained from data transformation. Before using PCA or ICA, a *normalization* process is first applied on feature matrix F. We consider that all the features are equally important. Note that the collected data may have different scales. For example, memory size is generally a large number, while CPU utilization is usually less than 1.0. To transform the data into a uniform scale, the matrix F is first normalized to F' such that feature values are controlled in the range between 0.0 and 1.0. Next, it is necessary to adjust F' to F'' so that its columns have *zero mean*. This ensures that PCA and ICA dimensions capture true variance, thereby avoiding skewing results due to differences in mean [25]. For data matrix F'', each column represents a node and each row gives the values of a particular feature.

3.2.1 PCA-Based Feature Extraction

PCA is a well-known dimensionality reduction technique [16], [25]. Basically, it performs linear transformation to map a given set of data points onto new axes (called *principal components*). When working with zero-mean data, each principal component has the property that it points in the direction of maximum variance remaining in the data, given the variance already accounted for in the preceding components.

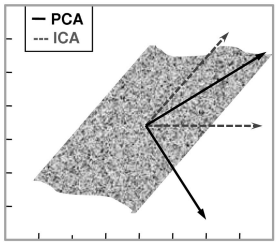


Fig. 4. PCA versus ICA. In this 2D plane, the principal components in PCA are orthogonal and in order while the independent components in ICA are not. PCA maximizes the variance while ICA aims at getting independent projection.

We apply PCA on the matrix F'', treating each column of F'' as a data point in $\mathbb{R}^{m\times k}$. It first calculates the covariance matrix of F'':

$$C = \frac{1}{n} F'' F''^T. \tag{2}$$

It then calculates the Eigenvalues of C and sorts them in a descending order: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{m\times k}$. The first s eigenvalues that satisfy the following requirement are chosen:

$$\frac{\sum_{i=1}^{s} \lambda_i}{\sum_{i=1}^{m\times k} \lambda_i} \geq TH, \tag{3}$$

where $TH < 1$ is a predefined threshold.

A projection matrix $A = [a_1, a_2, \ldots, a_s]$ is constructed, where a_i (the ith principal component) is the Eigenvector corresponding to λ_i. Each data point $f_i'' \in \mathbb{R}^{m\times k}$ is then projected into a data point $y_i \in \mathbb{R}^s$:

$$y_i = A^T f_i'', \quad i = 1, \ldots, n. \tag{4}$$

The complexity of computing all the eigenvectors and eigenvalues of C is not trivial. However, we only need to calculate the first s eigenvectors and eigenvalues (typically, $s \leq 3$). A neural network-based method can be applied for the calculation, with a linear complexity and fast convergence [36].

3.2.2 ICA-Based Feature Extraction

ICA is a blind source separation technique [30]. Similar to PCA, ICA also finds a new set of basis vectors for vector space transform. However, PCA finds the directions of *maximal variance* by using the second order statistics (i.e., covariance matrix), while ICA finds the directions of *maximal independence* by using higher order statistics. Unlike PCA, the basis vectors in ICA (called *independent components*) are not necessarily orthogonal and in order. Fig. 4 pictorially shows the major differences between PCA and ICA.

There are a number of algorithms for performing ICA. For the purpose of fast convergence, we choose the FastICA algorithm developed by Hyvärinen and Oja [30]. *Whitening* is a typical preprocessing step used to simplify and reduce the complexity of ICA algorithms. It ensures that all the dimensions are treated equally before the algorithm is run. For a vector v, its whitening means that its covariance matrix is equal to the identity matrix, that is, $\frac{1}{n} vv^T = I$. To whiten the matrix F'', we first calculate its covariance matrix C (2), and then calculate nonzero Eigenvalues of C in a descent order:

$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r$. Let $V = diag(\lambda_1, \lambda_2, \ldots, \lambda_r)$ and $E = [e_1, e_2, \ldots, e_r]$, where e_i is the Eigenvector corresponding to λ_i. The whitened data of F'' are defined as

$$X = V^{-1/2} E^T F'', \tag{5}$$

where X is a $r \times n$ matrix and $r \leq m \times k$.

After whitening, ICA projects the data point $x_i \in \mathbb{R}^r$ into a data point $y_i \in \mathbb{R}^s$ as below:

$$y_i = W^T x_i. \tag{6}$$

Here, to compare ICA and PCA under identical conditions, the dimensionality of y_i is also set to s (see (3)). A drawback is that this may discard some useful information. One of our future work is to exploit other techniques to estimate the dimensionality of y_i [32].

The goal is to find an optimal projection matrix W such that y_i are maximally independent, measured by some function [30]. In general, an iterative process is employed to search for W:

1. Choose a random initial matrix $W = [w_1, w_2, \ldots, w_s]$, where $\|w_i\| = 1$.
2. For $i = 1$ to s

$$w_i^+ = \frac{1}{n} \sum_{i=1}^{n} x_i (w_j^T x_i)^3 - 3 w_j,$$
$$w_i = w_i^+ / \|w_i^+\|.$$

   Here, the first calculation is to ensure that the iteration approaches the maximal independence and the second one ensures that $\|w_i\| = 1$.
3. $W = W(W^T W)^{-1/2}$, which is to ensure that $w_i \neq w_j$ when $i \neq j$.
4. If $(1 - |w_i^T w_i|) < \delta$, where δ is a small constant, the algorithm converges; Otherwise, go to step 2.

The convergence of FastICA is good [30]. In our experiments, generally only a few iterations are needed, and the total calculation time is less than 0.1 second.

3.3 Outlier Detection

The final step is to identify a subset of nodes that are significantly dissimilar from the majority. In the field of data mining, these nodes are called *outliers*. Simply put, an outlier is a data point that is quite different from other data according to some criteria [17]. In this paper, we measure the dissimilarity between two data points $y_a \in \mathbb{R}^s$ and $y_b \in \mathbb{R}^s$ by using euclidean distance:

$$d(y_a, y_b) = \sqrt{\sum_{i=1}^{s} (y_{a,i} - y_{b,i})^2}. \tag{7}$$

An object o is called a $DB(p, d)$ *outlier* if at least p fraction of the objects in the data set lies greater than d distance away from o.

A straightforward algorithm to detect a $DB(p, d)$ outlier is to calculate the number of neighbors within the distance of d from each object to determine whether it is an outlier or not. This naive object-based algorithm has a complexity of $O(sn^2)$, where s is the dimensionality and n is the number of objects in the data set. Obviously, this is not efficient, especially when there are a large number of nodes (e.g., tens to hundreds of thousands) in the system. We adopt a *cell-based algorithm* for outlier detection [17]. It has a complexity of $O(c^s + n)$, where c is a small constant. Note that after feature extraction, s is quite small (e.g., in our experiments, it is 3). Hence, it is more time efficient, as compared to a naive outlier detection algorithm.

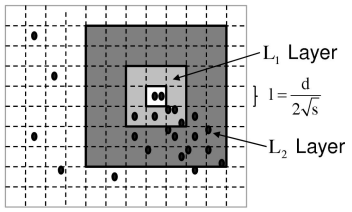


Fig. 5. Cell-based outlier detection. The data space is partitioned into cells of length $l = \frac{d}{2\sqrt{s}}$. Each cell is surrounded by two layers: L_1 (in the light-gray area) and L_2 (in the dark-gray area).

The cell-based algorithm works as follows. We first partition the data space that holds $Y = \{y_1, y_2, \ldots, y_n\}$ into cells of length $l = \frac{d}{2\sqrt{s}}$ (see Fig. 5). Each cell is surrounded by two layers: 1) the first layer L_1 includes those immediate neighbors and 2) the second layer L_2 includes those additional cells within three cells of distance. For simplicity of discussion, let M be the maximum number of objects within the d-neighborhood of an outlier (i.e., within a distance of d). According to the outlier definition, the fraction p is the minimum fraction of objects in the data set that must be *outside* the d-neighborhood of an outlier. Hence, $M = n(1 - p)$. The cell-based algorithm aims to quickly identify a large number of outliers and nonoutliers according to three properties, in the order as listed below:

1. If there are $>M$ objects in one cell, none of the objects in this cell is an outlier.
2. If there are $>M$ objects in one cell plus the L_1 layer, none of the objects in this cell is an outlier.
3. If there are $\leq M$ objects in one cell plus the L_1 layer and the L_2 layer, every object in this cell is an outlier.

These properties are used in the order to determine outliers and nonoutliers on a cell by cell basis rather than on an object by object basis. For cells not satisfying any of the properties, we have to resort to object by object processing. As analyzed in [17], this optimized algorithm can significantly reduce execution time to a complexity that is linear with respect to n.

The above detection algorithm will separate the data set Y into two subsets: normal data set Y_n and abnormal data set Y_a. For each y_i, we calculate its anomaly score:

$$\eta_i = \begin{cases} 0, & y_i \in Y_n, \\ d(y_i, \mu), & y_i \in Y_a, \end{cases} \tag{8}$$

where μ is the nearest point belonging to the normal data set Y_n. Anomaly score indicates the severity of anomaly. The abnormal subset, along with anomaly scores, will be sent to system administrators for final validation.

4 Experiments

We evaluate our prototype implementation on a production system by manually injecting a variety of system faults. While experimental evaluation with real faults would be

better, a major problem with this approach is that we do not often have the luxury of allowing systems to run for many days to see their behaviors. Further, it is not guaranteed that the system will experience a variety of faults during the experimental time. The generally accepted solution to this problem is to inject the effects of faults in a system and observe the behavior of the system under the injected faults [54]. In our experiments, we are able to test the proposed anomaly identification mechanism on dozens of fault effects via fault injection.

We compare three types of identification mechanisms, namely the one using ICA-based feature extraction, the one using PCA-based feature extraction, and the one without using any feature extraction. In the rest of the paper, we simply use the terms *ICA-based*, *PCA-based*, and *No-Extraction* to denote these mechanisms.

4.1 Testbed

We use the IA64 Linux cluster called *Mecury* at NCSA as our testbed, which is part of the TeraGrid infrastructure [1]. It consists of 887 IBM nodes: 256 nodes with dual 1.3 GHz Intel Itanium 2 processors and 631 nodes with dual 1.5 GHz Intel Itanium 2 processors, all connected by Myrinet interconnect network. The cluster is running SuSE Linux.

Our experiments are conducted on 256 1.5 GHz nodes. A parameter sweep application is submitted on these nodes, where application processes solve dense linear equations by using Gaussian Elimination method, thereby performing comparable computation tasks [42].

4.2 Fault Injection

We randomly inject faults into the testbed by generating faulty threads in the background—separating from the application threads, and test whether different mechanisms can effectively identify the faulty nodes. Three random factors are considered in our fault injection. First is to decide how many nodes to inject faults, second is to determine which nodes to inject faults, and the last is to decide the type(s) of fault(s) to inject. In our experiments, 0-20 nodes are randomly selected for fault injection. The inclusion of the zero cases is to test our anomaly identification mechanism under a fault-free environment. Totally, five types of faults are tested as follows:

- *Memory leaking*: On randomly selected nodes, besides the normal computation threads, we introduce threads to generate memory leaking on the nodes, meaning that these threads continue consuming memory without releasing it periodically.
- *Unterminated CPU-intensive threads*: On randomly selected nodes, the injected threads compete for the CPU resource with the normal computation threads on the nodes.
- *Frequent I/O operations*: On randomly selected nodes, we introduce extra I/O intensive threads, which keep reading and writing a large number of bytes from local disks.
- *Network volume overflow*: On randomly selected nodes, additional threads are introduced to keep transferring a large number of packets among them.
- *Deadlock*: On randomly selected nodes, the injected threads block application processes from system resources.

We select these faults based on the literature and our own experience on system log analysis [44], [45], [46]. For example, we have found that some job hang failures are triggered by deadlock, some network-related failures like packet loss are caused by heavy traffic volume, and some node map file failures are due to frequent IO operations. In our experiments, these faults are supposed to originate from the system rather than from the application. They can affect multiple subsystems in a node, including memory, CPU, I/O, and network, and may eventually lead to system failure and/or performance degradation.

TABLE 1
Feature List

Features	Description
1. CPU_System_Proc1 2. CPU_System_Proc2	Percentage of CPU utilization at the system level
3. CPU_User_Proc1 4. CPU_User_Proc2	Percentage of CPU utilization at the user level
5. CPU_IO_Proc1 6. CPU_IO_Proc2	Percentage of time CPU blocked for I/O
7. Memory_Free	Amount of free memory (KB)
8. Memory_Swapped	Amount of virtual memory used (KB)
9. Page_In	Paged in from swap (KB/s)
10. Page_Out	Paged out to swap (KB/s)
11. IO_Write	No. of blocks written per second
12. IO_Read	No. of blocks read per second
13. Context_Switch	No. of context switches per second
14. Packet_In	No. of packets received per second
15. Packet_Out	No. of packets transmitted per second
16. Comp_Proc1 17. Comp_Proc2	Computation time
18. Comm_Proc1 19. Comm_Proc2	Communication time

4.3 Collected Features

Totally, we collect 19 features per node. At the operating system layer, we use four system commands, namely *vmstat*, *mpstat*, *iostat*, and *netstat*, to collect node-level features from CPU, memory, I/O, and network. We also use *mpstat* to collect CPU-level features to distinguish dual CPUs per node. Further, we collect two application-level features by using MPI profiling interface (PMPI). PMPI allows us to obtain application-level features without modifying user codes [5]. These features are summarized in Table 1.

4.4 Evaluation Metrics

The goal of anomaly identification is to separate the nodes into two classes: those that contain faults (i.e., anomalies) and those that do not. *Sensitivity* and *specificity* are two widely used metrics to measure a binary classification test. *Sensitivity* measures the proportion of actual *positives*, which are correctly identified, and *specificity* measures the proportion of *negatives*, which are correctly identified. More specifically, they are defined as follows:

1. *Sensitivity* is the proportion of correct faulty classifications to the number of actual faulty nodes

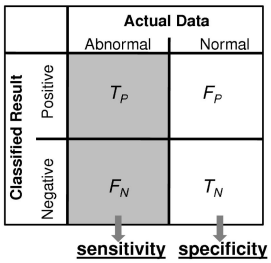


Fig. 6. Evaluation metrics. *Sensitivity* is the proportion of correct faulty classifications to the number of actual faults, i.e., opposite to missed rate; *specificity* is the proportion of correct nonfaulty classifications to the number of actual normal nodes, i.e., opposite of false alarm rate.

$$sensitivity = \frac{T_P}{T_P + F_N}. \quad (9)$$

2. *Specificity* is the proportion of correct nonfaulty classifications to the number of actual normal nodes

$$specificity = \frac{T_N}{F_P + T_N}. \quad (10)$$

Here, T_P, F_P, F_N, and T_N denote the number of true positives, false positives, false negatives, and true negatives, respectively. Fig. 6 illustrates the relationships among these terms.

A good mechanism should provide a high value (close to 1.0) for both metrics. A *sensitivity* of 1.0 means that the mechanism recognizes all the faulty nodes and a *specificity* of 1.0 means that the mechanism identifies all the nonfaulty nodes. High *sensitivity* is expected to avoid the cases, where humans have to reprocess all the data to find out those missed anomalies. In essence, it determines whether the proposed automated mechanism is useful in practice. Low *specificity* should be avoided as it leads to nontrivial human effort to remove false alarms via manual processing. Our goal is to obtain extremely high *sensitivity* (very close to 1.0) and high *specificity* (better above 0.9).

4.5 Parameter Setting

Choosing optimal values for parameters is difficult in practice, and often experimental determination might be the only viable option.

In our experiments, we collect features every 40 seconds and 5 samples per node. As a result, the feature matrix F presented in Fig. 3 is of the size $(19 \times 5) \times 256$. In general, sampling frequency is dependent on monitoring tools and an understanding of fault modes. For example, the default monitoring frequency used by Ganglia is 300 seconds, whereas Supermon is capable of providing up to 6,000 samples per second. A high sampling frequency usually improves analysis accuracy at the expense of introducing more computation overhead. We choose these parameters based on a study of several commonly used monitoring tools and our understanding of the injected fault effects. We have tried different sampling frequencies, from a low value (e.g., every 300 seconds) to a high value (e.g., every 10 seconds). In the cases where the frequency is lower, we could end up with insufficient sampling data for automated analysis. On the other hand, in the cases where the frequency is higher like every 10 seconds, the analysis accuracy remains similar, while the computation overhead is increased by 38 percent.

TABLE 2
Accuracy Results In Case of Single-Fault Tests There are Two Numbers in Each Cell: The First is *sensitivity* and the Second is *specificity*

Fault	ICA-based	PCA-based	No-Extraction
memory	1.00/0.97	1.00/0.93	0.97/0.80
CPU	1.00/0.94	1.00/0.93	1.00/0.82
I/O	1.00/0.93	1.00/0.92	1.00/0.77
network	1.00/0.93	0.89/0.86	0.92/0.74
deadlock	1.00/0.97	1.00/0.94	1.00/0.85
Average	1.000/0.948	0.978/0.916	0.978/0.796

A value of $1.00/1.00$ indicates a perfect result. The results indicate that both ICA and PCA-based mechanisms are good candidates for automated anomaly identification, in case of single-fault scenarios. The results also demonstrate that no-extraction is not effective due to its low specificity.

Then, the threshold TH used in (3) is set to 0.80, which results in three principal components. The number of independent components is also set to 3 for the purpose of fair comparison. As a result, data reduction achieved by both PCA and ICA is over 96 percent. We have also tested with different values for TH and the resulting number of principal components does not change dramatically.

Finally, for outlier detection, the parameters p and d are set to 0.9 and variance of Y (i.e., the newly constructed data space after applying feature extraction), respectively. According to the definition of p described in Section 3.3, p should be set to the maximal fraction of possible faulty nodes in a system. Based on our experiences on system logs [44], [45], [46], we have found that the percentage of faulty nodes is generally lower than 10 percent. As a result, we set p to 0.9. A higher value of p typically results in better *sensitivity*, whereas *specificity* could be lower.

4.6 Results

We conduct two sets of experiments: 1) *single-fault tests*, where one type of faults are injected into the system and 2) *multifault tests*, where multiple types of faults are injected into the system. For each experiment, we conduct multiple runs (i.e., 4-10) and the results shown here are the average of multiple runs.

4.6.1 Single-Fault Tests

Table 2 lists accuracy results for the first set of experiments, i.e., single-fault tests. In the experiment, we inject one type of faults onto 0-20 randomly selected nodes and assess whether different mechanisms can correctly identify these nodes. Fig. 7 pictorially shows the results achieved by PCA- and ICA-based mechanisms. On each plot, the X, Y, and Z-axis represent the 1st, 2nd, and 3rd principal or independent components, respectively. The points in the ellipses are true outliers, those in the rectangles are false outliers, and those in the triangle are missed outliers.

First, we can see that without feature extraction, the results, especially *specificity*, are much lower than the cases using feature extraction. This indicates that in addition to reducing data size, feature extraction can also

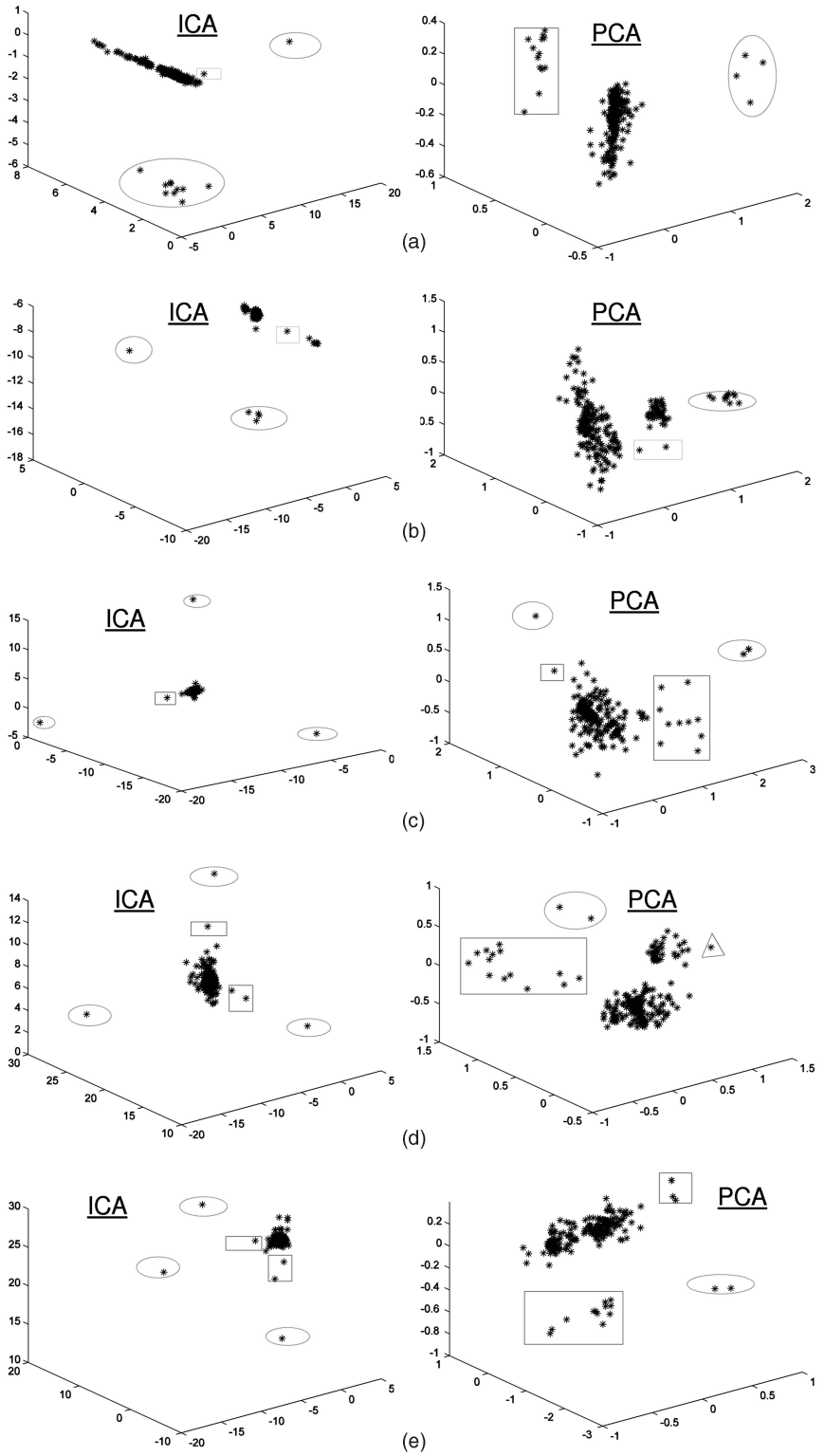


Fig. 7. Single-fault test results by using ICA and PCA-based mechanisms. The ones in the ellipses are true outliers, those in the rectangles are false outliers, and those in the triangle are missed outliers. While both mechanisms are good for anomaly identification in case of a single type of faults, ICA-defined subspaces can better separate abnormal nodes from normal nodes as compared to PCA-defined subspaces. (a) Memory, (b) CPU, (c) IO, (d) network, (d) deadlock.

improve accuracy. As mentioned before, we collect node features during regular system operation. Thus, noise and fluctuation are unavoidable in the collected features, which can lead to false identifications. Feature extraction can reduce these negative effects by keeping the most useful characteristics in the features. Further, in the new data space, feature extraction also amplifies the difference between normal and abnormal behaviors, thereby decreasing the possibility of identifying normal nodes as anomalies.

Next, it is clear that ICA-based mechanism provides the best results in terms of both *sensitivity* and *specificity*. By examining the key difference between ICA and PCA, we believe that the benefits of ICA come from two aspects. First, much of the information that perceptually characterizes node behaviors is contained in the higher order statistics of the data. Distinguished from PCA that only uses the second order statistics for feature extraction, ICA explores higher order statistics of the data, thereby outperforming PCA. Second, as we can observe in Fig. 7, ICA uses a different subspace than PCA. Since ICA allows the basis vectors to be nonorthogonal, the angles and distances between data points differ in the subspaces generated by ICA and PCA. Clearly, ICA can better separate abnormal groups from normal groups by grasping the normal points much closer to each other than PCA. This implies that ICA-defined subspaces encode more information about node behaviors than PCA-defined subspaces.

Last but not least, we also observe that detecting network-related faults has the lowest accuracy, followed by IO-related faults. For example, when using ICA-based mechanism, the values of *specificity* are 0.93, lower than the average value of 0.948; when using PCA-based mechanism, the accuracy results are $(0.89, 0.86)$ and $(1.00, 0.92)$ for network-related and IO-related faults, respectively; when using no-extraction mechanism, the values of *specificity* are lower than 0.80. We attribute this to the fact that network- and IO-related faults can easily propagate throughout the system [7], [28]. Due to fault propagation, neighboring nodes may get infected, and as a result, the outlier list generated by anomaly identification mechanisms may be larger than the actual anomaly list. One solution to address this problem is to involve human in the loop as shown in Fig. 1. Further, we notice that ICA is better at capturing these faults than PCA (see Figs. 7c and 7d). Unlike PCA, ICA is capable of substantially separating the normal nodes away from the real faulty nodes and infected nodes, thereby resulting in a perfect *sensitivity*.

Since our goal is to obtain extremely high *sensitivity* and high *specificity*, we conclude that, in case of single-fault scenarios, both ICA and PCA-based mechanisms are good candidates for automated anomaly identification, whereas no-extraction mechanism can introduce nontrivial false alarms as the average *specificity* is only about 0.796.

ICA and PCA apply different ways to transform the original data space, both with the goal to better separate data points. As we can see, both mechanisms approach the goal by achieving comparable results, except for the faults that tend to propagate throughout the system. As we will see in the second experiment, this is no longer a case when multiple faults exist in the system.

4.6.2 Multifault Tests

In this set of experiments, different types of faults are simultaneously injected onto 0-20 nodes in the system. The results for two-fault tests are listed in Table 3. We have also conducted experiments with three and four types of faults. The results are very similar to those shown in the table, so we omit them here. Meanwhile, Fig. 8 pictorially shows some two-fault results achieved by PCA and ICA-based mechanisms. Again, on each plot, the X, Y, and Z-axis represent the 1st, 2nd, and 3rd principal or independent components, respectively. The points in the ellipses are true outliers, those in the rectangles are false outliers, and those in the triangles are missed outliers.

TABLE 3
Accuracy Results In Case of Multifault Tests There are Two Numbers in Each Cell: The First is *sensitivity* and the Second is *specificity*

Faults	ICA-based	PCA-based	No-Extraction
memory & deadlock	1.00/0.97	0.52/0.93	1.00/0.80
memory & CPU	1.00/0.96	0.80/0.91	1.00/0.79
CPU & deadlock	1.00/0.97	1.00/0.92	1.00/0.81
memory & IO	1.00/0.93	0.71/0.93	0.80/0.81
memory & network	1.00/0.92	0.55/0.92	1.00/0.81
CPU & network	1.00/0.94	0.52/0.93	0.70/0.78
CPU & IO	1.00/0.93	0.75/0.81	1.00/0.79
IO & deadlock	1.00/0.93	0.51/0.91	1.00/0.76
deadlock & network	1.00/0.94	0.52/0.89	1.00/0.78
IO & network	1.00/0.92	0.25/0.85	0.70/0.76
Average	1.000/0.941	0.613/0.900	0.920/0.789

The results demonstrate that PCA-based mechanism is not effective in identifying multiple simultaneous faults and ICA-based mechanism outperforms PCA-based mechanism in terms of both sensitivity and specificity.

Consistent with the results shown in Table 2, without feature extraction (i.e., no-extraction), *specificity* is low. This demonstrates that for both single-fault and multifault tests, feature extraction is indispensable as it can not only reduce data size but also improve *specificity*. Low *specificity* means that there are nontrivial amount of false alarms. False identifications can be caused by noise in the original data or by the closeness of data points in the original space. Feature extraction is capable of reducing noise effects by keeping the most relevant information in the data and distinguishing groups by amplifying the difference between normal and abnormal behaviors.

ICA-based mechanism produces the best results by detecting every anomalous nodes, with the average *specificity* above 0.94. The results are also consistent with those shown in the single-fault tests. As shown in Fig. 8, ICA can effectively define a basis set of statistically independent axes, where data points in the ICA-defined subspaces are better separated. The majority of data points join together as a cluster, which represent normal nodes. The points that are substantially away from the major cluster are true outliers (i.e., in the ellipses). On each plot, there are several data points close to the major cluster, indicating false outliers.

A very interesting observation is that PCA does not work well when multiple types of faults occur simultaneously. According to the data of the table, in 6 out of 10 tests, the values of *sensitivity* are lower than 0.60,

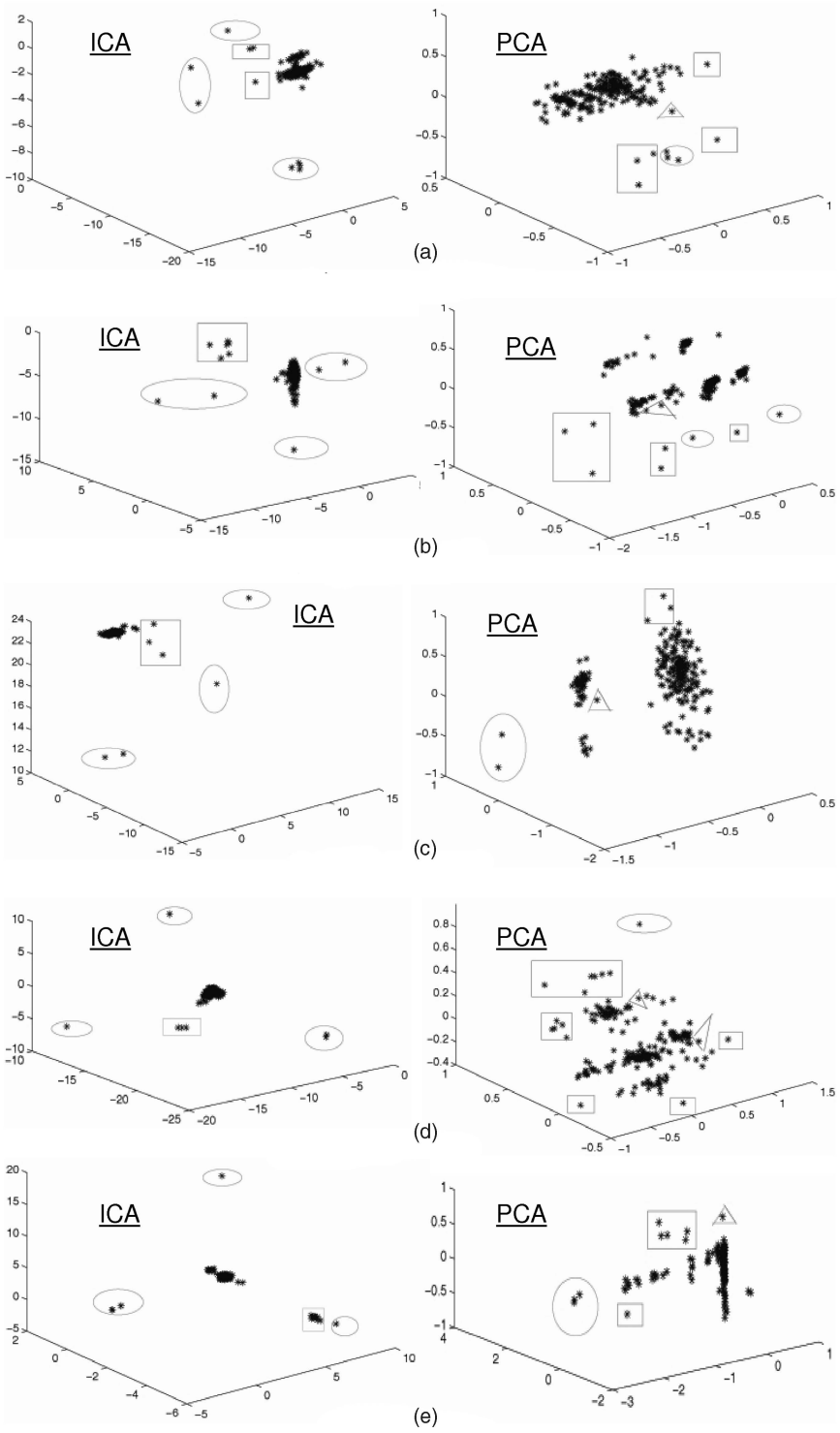


Fig. 8. Multifault test results by using ICA- and PCA-based mechanisms. The ones in the ellipses are true outliers, those in the rectangles are false outliers, and those in the triangles are missed outliers. PCA does not work well when multiple types of faults occur simultaneously. (a) Memory-deadlock, (b) memory-CPU, (c) CPU-network, (d) IO-deadlock, (e) IO-network.

meaning that more than 40 percent of anomalies are missed by using PCA. The extreme case is when IO-related faults and network-related faults coexist in the system for which *sensitivity* is only at 0.25. In single-fault tests, we mentioned that network- and IO-related faults are difficult to detect due to their easy propagation characteristics. PCA produces a terrible result when a mixture of these faults occurs in the system. We attribute this to the nature of PCA: PCA focuses on finding the directions of *maximal variance* by using the second order statistic. As a result, in case of multiple faults, PCA chooses the basis vectors that pertain to one "strong" fault and ignores the information that is relevant to other faults. Hence, PCA is not effective in identifying simultaneous faults of different types.

Both single-fault and multifault tests have shown that ICA outperforms PCA. The main reason is that ICA is

designed to maximize independence by using higher order statistics in the data, whereas PCA focuses on maximized variance by using the second order statistics in the data. As a result, ICA is more robust to noise and its defined subspaces can effectively encode more information of node behaviors than PCA-defined subspaces. It also indicates that second-order statistic is not sufficient to describe the behavior of faulty nodes. As a result, PCA is not effective in detecting faults that are easily propagated throughout the system. Also, it should not be adopted when there are multiple faults coexisting in the system.

4.6.3 Computation Overhead

Both PCA and ICA-based mechanisms are time efficient. In our experiments, both take less than 0.25 second to produce an anomaly list, which makes them feasible for practical usage. As stated before in Section 3.2, this is due to the fact that we only need to calculate a few Eigenvectors and Eigenvalues by using PCA and FastICA (usually converges in less than 40 iterations).

In summary, our experiments have shown that:

- Feature extraction can substantially improve *specificity* by reducing false identifications.
- Both PCA and ICA-based mechanisms are time efficient, e.g., taking less than 0.25 second to process voluminous amount of noisy, high-dimensional data in our experiments.
- PCA-based mechanism works well for some faults; however, it is inefficient when there are multiple types of faults coexisting in the systems or when faults have strong propagation characteristics.
- ICA-based mechanism is a promising approach for automated anomaly identification in large-scale systems. It can effectively identify a variety of faults by capturing every fault injected (i.e., *sensitivity* is always at 1.0), with the false alarm rate less than 8 percent (i.e., *specificity* is always above 0.92). This will guarantee that little amount of human effort is needed for final validation.

5 Discussions

This study has presented a general framework for node-level anomaly identification. In particular, it has compared two techniques, namely PCA and ICA, for feature extraction. Our results have demonstrated that ICA-based mechanism always outperforms PCA-based mechanism. The limitation of PCA is derived from its assumptions. First is its assumption on the statistical importance of mean and covariance. It only uses the second order statistics for the derivation of basis vectors and the basis vectors have to be orthogonal and in order. It does not guarantee that the directions of maximal variance will contain good features for discrimination. Second is due to its assumption that large variances have important dynamics. It is only effective when the observed data have a high signal-to-noise ratio such that the principal components with larger variance correspond to interesting dynamics and lower ones correspond to noise. Our studies have shown that feature data collected for troubleshooting are noisy, thus, PCA-based mechanism is not effective in detecting some faults. ICA overcomes these limitations by using a different way to construct the basis vectors.

The anomaly identification framework presented in Fig. 1 is general and extensible as it works with various data sources as long as the data can well capture node behaviors. As defined in Section 1, this study emphasizes the identification of anomalies, where anomalies are the nodes in error or failed states and they may be caused by system faults in hardware or software. In case that we are finding the nodes in error states, meaning that the nodes are not failed yet, the presented framework can be used as a failure prediction support. Otherwise, if the nodes are already in the failed states, then the presented framework can be considered as a failure diagnosis support.

Similar to other automatic approaches, our mechanism is also subjected to some errors. For example, the ICA-based approach may result in a false alarm rate by up to 8 percent. False alarms may be caused by background noise or fault propagation. We believe that a promising way to address this issue is to involve the human in data analysis, which is exactly what we propose in Fig. 1. The anomaly list produced by our mechanism can be sent to system administrators for further investigation. Other research projects have also pointed out the importance of keeping the human in failure analysis [48].

Our mechanism relies on the assumption that nodes in the same group should exhibit similar behaviors. This can be achieved by the use of dynamic grouping. While this study focuses on homogeneous collection of nodes, in order to be of systemwide use, our mechanism needs to interact with resource management and job scheduling tools to properly divide system resources into different groups. For instance, the nodes performing comparable activities, e.g., a set of dedicated IO nodes or a set of compute nodes allocated to the same parameter sweep application, can be grouped together. Further, the nodes allocated to parallel applications can also be divided into different groups by examining the call stacks during process initialization stage [55]. In other words, the master nodes and worker nodes may be allocated into different groups. How to effectively integrate with resource management and job scheduling tools is one of our ongoing efforts. In addition, our mechanism does not work when a fault occurs across the entire system. Fortunately, this kind of faults can easily be detected via existing technologies.

Our study has some limitations that remain as our future work. First, in our current design, the number of independent axes in ICA is set based on a criterion used by PCA, which may discard some useful information in the data. This could be addressed by exploiting techniques like the one presented in [32]. Second, the presented framework can be extended by tracing the change of outliers and this could address the fault propagation issue in large-scale systems. Finally, in our experiments, we have mainly collected performance data from the operating system for constructing feature matrix. An interesting direction is to investigate how to include other data sources for anomaly identification. As an example, how to transfer system events into our feature space is worthwhile to explore.

6 Related Work

For large-scale systems, a key challenge facing by their system operators is to understand failure trends and modes. Over the past, numerous studies have been conducted on diagnosing and/or predicting failures based on system events. Broadly speaking, we can classify existing approaches into two groups: *model-based* and *data-driven*. A model-based approach derives a probabilistic or analytical model of the system. A warning is triggered when a deviation from the model is detected [43]. Examples include an adaptive statistical data fitting method called MSET presented in [47], naive Bayesian-based models for disk failure prediction [23], and Semi-Markov reward models described in [49]. In large-scale systems, errors may propagate from one component to other components, thereby making it difficult to identify root causes of the problem(s). A common solution is to develop fault propagation models (FPMs), such as causality graphs or dependency graphs [4]. Generating dependency graphs, however, requires a priori knowledge about the system structure and the dependencies among different components, which is hard to obtain in large-scale systems. The major limitation of model-based methods is the difficulty of generating and maintaining an accurate model, especially given the unprecedented size and complexity of large-scale systems.

Recently, data mining and statistical learning theory (SLT) have received growing attention for failure diagnosis and prognosis. These methods extract fault patterns from system normal behaviors and detect abnormal observations based on the learned knowledge without assuming a priori model ahead of time [37]. For example, the group at the Berkeley RAD laboratory has applied statistical learning techniques for failure diagnosis in Internet services [2], [12]. The Statistical Learning, Inference, and Control (SLIC) project at HP has explored similar techniques for automating fault management of IT systems [50]. In [10], [13], the authors have presented several methods to predict failure events in IBM clusters. Liang et al. [8], [34] have examined several statistical-based methods for failure prediction in IBM Blue Gene/L systems. Fu and Xu have developed a framework called hPREFECTS for failure prediction in networked computing systems [3]. In our own studies [44], [45], we have investigated meta-learning-based method for improving failure prediction by adaptively combining various data mining techniques. Other representative studies include system log analysis [14], [27] and fault detection in syslogs [33].

While our approach belongs to the data-driven category (i.e., making decisions by gathering and analyzing large amounts of data), it focuses more on building a systematic framework for automated anomaly identification in large-scale systems. In addition, this study demonstrates that ICA is a promising candidate for anomaly identification in large-scale systems.

Most studies on failure diagnosis determine system errors by examining historical data, which we denote as being based on a *vertical* view of the system. They are often characterized by the requirement of training a classifier on samples of both normal and abnormal data. The fundamental problem with this approach is that it limits us to the detection and localization of faults with known signatures. Distinguished from this approach, our mechanism exploits a *horizontal* view of the system for anomaly identification. It does not require training on good samples, thereby being capable of finding anomalies not yet seen in the data. Like the work of Stearley and Oliner [33], our mechanism is also based on the observation that system nodes performing similar activities should exhibit similar behaviors. However, unlike [33], which uses the encoding of word position in syslogs for fault detection, our work employs feature extraction to reduce data dimensionality and outlier detection to automatically determine the nodes that are significantly different from the majority.

Our mechanism can also aid failure prediction. Unlike failure diagnosis, which focuses on detecting and localizing failures, failure prediction emphasizes on forecasting occurrences of system failures. Earlier studies have shown that it is possible to predict some failures by analyzing reliability, availability, and serviceability (RAS) logs collected and archived in systems [8], [10], [44], [45]. However, due to the lack of information in RAS logs, most often the prediction is on the level of system, meaning that we may predict when the system will likely experience some failure, but unable to tell where. A possible solution is to archive system performance data in addition to RAS logs. Existing predictive methods such as those presented in [8], [10], [45] are used on the RAS events to predict when the underlying system will experience a critical event (e.g., a hardware or software failure) and this work can be utilized to determine the potential localizations of the problem by analyzing performance data.

Our work is inspired by the work of Lakhina et al. [25], which explores PCA to diagnose network-wide traffic anomalies. This paper is fundamentally different from that presented in [25] in three key aspects. First, we target identifying faulty nodes in large-scale systems such as those used for high-performance computing [1]. In doing so, our feature space is quite different from the one used in [25]. Second, they apply classification techniques (i.e., a supervised learning) to identify network anomalies, while we investigate the use of an optimized outlier detection method (i.e., an unsupervised learning) to find out outliers. Finally, we have demonstrated through experiments that ICA outperforms PCA for anomaly identification in large-scale systems.

PCA is a well-known method for feature extraction and has been applied in many fields [16], such as problem diagnosis [25] and application classification [26]. However, as pointed out in our experiments, PCA has several limitations. In particular, it cannot distinguish a mixture of multiple types of faults and does not work well with the faults that are easily propagated across the system. There has been a growing wave of papers on ICA and its algorithms since the mid-1990s [31]. To date, ICA has been used for signal processing, face recognition, industrial processing, chemical engineering, and biomedical engineering [30], [32], [51], [52], [53]. To the best of our knowledge, we are among the first to explore ICA for anomaly identification in high-performance computing systems.

There are a number of other feature extraction techniques, such as linear discriminant analysis (LDA) [16] and nonnegative matrix factorization (NMF) [57]. These methods either are based on supervised learning or have a low

computation convergence, thereby not being examined in this work. Both ICA and PCA are linear extraction methods. Recently, some nonlinear methods have been proposed to address the issues in linear methods like kernel methods [58] and multidimensional scaling (MDS) [56]. The investigation of these nonlinear methods remains as one of our future work.

7 Conclusion

In this paper, we have presented an automated mechanism for anomaly identification in large-scale systems. It applies three interrelated techniques to dynamically get the anomaly lists. Our experiments on a production system via manually fault injection have demonstrated that ICA-based mechanism can provide high accuracy by effectively discovering a variety of faults, with perfect *sensitivity* and extremely high *specificity*. This indicates that ICA has a great potential for anomaly identification in large-scale systems. It is one of our goals that this work will open up a new research direction in using ICA for anomaly detection and localization in large-scale systems.

Acknowledgments

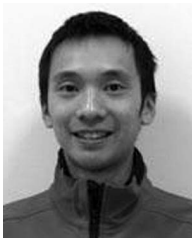
This work is supported in part by the US National Science Foundation grants CNS-0834514, CNS-0720549, CCF-0702737, and a TeraGrid Compute Allocation. Some preliminary results of this work were presented in [29]. This work was performed when Y. Li was a student at Illinois Institute of Technology.

References

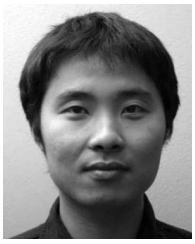
[1] TeraGrid Website, http://www.teragrid.org/, 2009.
[2] M. Chen, E. Kiciman, E. Fratkin, A. Fox., and E. Brewer, "Pinpoint: Problem Determination in Large, Dynamic, Internet Services," *Proc. Int'l Conf. Dependable Systems and Networks (DSN),* 2002.
[3] S. Fu and C. Xu, "Exploring Event Correlation for Failure Prediction in Coalitions of Clusters," *Proc. Conf. Supercomputing (SC '07),* 2007.
[4] M. Steinder and A. Sethi, "A Survey of Fault Localization Techniques in Computer Networks," *Science of Computer Programming,* vol. 53, no. 2, pp. 165-194, 2004.
[5] V. Tabatabaee and J. Hollingsworth, "Automatic Software Interference Detection in Parallel Applications," *Proc. Conf. Supercomputing (SC),* 2007.
[6] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Trans. Dependable and Secure Computing,* vol. 1, no. 1, pp. 11-33, Jan. 2004.
[7] W. Kao and R. Iyer, "DEFINE: A Distributed Fault Injection and Monitoring Environment," *Proc. IEEE Workshop Fault-Tolerant Parallel and Distributed Systems,* 1994.
[8] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo, "Blue Gene/L Failure Analysis and Models," *Proc. Int'l Conf. Dependable Systems and Networks (DSN),* 2006.
[9] G. Hoffmann, F. Salfner, and M. Malek, "Advanced Failure Prediction in Complex Software Systems," *Proc. Int'l Symp. Reliable Distributed Systems (SRDS),* 2004.
[10] R. Sahoo, A. Oliner, I. Rish, M. Gupta, J. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam, "Critical Event Prediction for Proactive Management in Large-Scale Computer Clusters," *Proc. ACM Special Interest Group on Knowledge Discovery in Data (SIGKDD),* 2003.
[11] K. Trivedi and K. Vaidyanathan, "A Measurement-Based Model for Estimation of Resource Exhaustion in Operational Software Systems," *Proc. Int'l Symp. Software Reliability Eng. (ISSRE),* 1999.
[12] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer, "Failure Diagnosis Using Decision Trees," *Proc. Int'l Conf. Autonomic Computing (ICAC),* 2004.
[13] R. Vilalta and S. Ma, "Predicting Rare Events in Temporal Domains," *Proc. Int'l Conf. Data Mining (ICDM),* 2002.
[14] A. Oliner and J. Stearley, "What Supercomputers Say: A Study of Five System Logs," *Proc. Int'l Conf. Dependable Systems and Networks (DSN),* 2007.
[15] J. Zhang and R. Figueiredo, "Application Classification through Monitoring and Learning of Resource Consumption Patterns," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* 2006.
[16] R. Duda, P. Hart, and D. Stork, *Pattern Classification,* second ed. Wiley Interscience, 2001.
[17] E. Knorr, R. Ng, and V. Tucakov, "Distance-Based Outliers: Algorithms and Applications," *The VLDB J.,* vol. 8, no. 3, pp. 237-253, 2000.
[18] Hardware Monitoring by LM Sensors, http://www.lm-sensors.org/, 2009.
[19] Intelligent Platform Management Interface, http://www.intel.com/design/servers/ipmi/, 2009.
[20] Splunk Website, http://www.splunk.com/, 2009.
[21] The Computer Failure Data Repository, http://institute.lanl.gov/data/lanldata.shtml, 2009.
[22] B. Allen, "Monitoring Hard Disk with SMART," *Linux J.,* 2004.
[23] G. Hamerly and C. Elkan, "Bayesian Approaches to Failure Prediction for Disk Drives," *Proc. Int'l Conf. Machine Learning (ICML),* 2001.
[24] A. Mirgorodskiy, N. Maruyama, and B. Miller, "Problem Diagnosis in Large-Scale Computing Environments," *Proc. Conf. Supercomputing (SC),* 2006.
[25] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing Network-Wide Traffic Anomalies," *Proc. ACM SIGCOMM,* 2004.
[26] J. Zhang and R. Figueiredo, "Application Classification through Monitoring and Learning of Resource Consumption Patterns," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* 2006.
[27] B. Schroeder and G. Gibson, "A Large-Scale Study of Failures in High Performance Computing Systems," *Proc. Int'l Conf. Dependable Systems and Networks (DSN),* 2006.
[28] A. Girard and B. Sansó, "Multicommodity Flow Models, Failure Propagation, and Reliable Loss Network Design," *IEEE/ACM Trans. Networking,* vol. 6, no. 1, pp. 82-93, Feb. 1998.
[29] Z. Zheng, Y. Li, and Z. Lan, "Anomaly Localization in Large-Scale Clusters," *Proc. IEEE Int'l Conf. Cluster Computing,* 2007.
[30] A. Hyvärinen and E. Oja, "Independent Component Analysis: Algorithms and Applications," *Neural Networks,* vol. 13, nos. 4/5, pp. 411-430, 2000.
[31] C. Jutten and J. Karhunen, "The Nonlinear ICA and BSS Problems," *Proc. Fourth Int'l Symp. Independent Component Analysis and Blind Signal Separation (ICA),* 2003.
[32] M. Bartlett, J. Movellan, and T. Sejnowski, "Face Recognition by Independent Component Analysis," *IEEE Trans. Neural Networks,* vol. 13, no. 6, pp. 1450-1464, Nov. 2002.
[33] J. Stearley and A. Oliner, "Bad Words: Finding Faults in Spirit's Syslogs," *Proc. Workshop Resiliency in High Performance Computing,* 2008.
[34] Y. Liang, Y. Zhang, H. Xiong, and R. Shaoo, "Failure Prediction in IBM BlueGene/L Event Logs," *Proc. Int'l Conf. Data Mining (ICDM),* 2007.
[35] L. Yang, C. Liu, J. Schopf, and I. Foster, "Anomaly Detection and Diagnosis in Grid Environments," *Proc. Conf. Supercomputing (SC),* 2007.
[36] Y. Rao and J. Principe, "A Fast, On-Line Algorithm for PCA and Its Convergence Characteristics," *Proc. IEEE Signal Processing Soc. Workshop,* 2000.
[37] W. Peng, T. Li, and S. Ma, "Mining Logs Files for Computing System Management," *Proc. Int'l Conf. Autonomic Computing (ICAC),* 2005.
[38] S. Venkataraman, J. Caballero, D. Song, A. Blum, and J. Yates, "Black-Box Anomaly Detection: Is It Utopian?," *Proc. Workshop Hot Topics in Networks (HotNets),* 2006.
[39] Ganglia Monitoring System, http://ganglia.sourceforge.net/, 2009.
[40] M. Scottile and R. Minnich, "Supermon: A High-Speed Cluster Monitoring System," *Proc. IEEE Int'l Conf. Cluster Computing,* 2002.
[41] S. Smallen, C. Olschanowski, K. Ericson, P. Bechman, and J. Schopf, "The Inca Test Harness and Reporting Framework," *Proc. Conf. Supercomputing (SC),* 2004.

[42] A. Grama, V. Kumar, A. Gupta, and G. Karpis, *Introduction to Parallel Computing*, second ed. Addison-Wesley, 2003.
[43] J. Hellerstein, F. Zhang, and P. Shahabuddin, "A Statistical Approach to Predictive Detection," *Computer Networks: The Int'l J. Computer and Telecomm. Networking*, vol. 35, pp. 77-95, 2001.
[44] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "A Meta-Learning Failure Predictor for Blue Gene/L Systems," *Proc. Int'l Conf. Parallel Processing (ICPP)*, 2007.
[45] J. Gu, Z. Zheng, Z. Lan, J. White, E. Hocks, and B.H. Park, "Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems: A Case Study," *Proc. Int'l Conf. Parallel Processing (ICPP)*, 2008.
[46] B. Park, Z. Zheng, Z. Lan, and A. Geist, "Analyzing Failure Events on ORNL's Cray XT4," *Proc. Conf. Supercomputing (SC '08)*, (research poster), 2008.
[47] K. Vaidyanathan and K. Gross, "MSET Performance Optimization for Detection of Software Aging," *Proc. Int'l Symp. Software Reliability Eng. (ISSRE)*, 2003.
[48] P. Bodik, G. Friedman, L. Biewald, H. Levine, G. Candea, K. Patel, G. Tolle, J. Hui, A. Fox, M. Jordan, and D. Patterson, "Combining Visualization and Statistical Analysis to Improve Operator Confidence and Efficiency for Failure Detection and Localization," *Proc. Int'l Conf. Autonomic Computing (ICAC)*, 2005.
[49] S. Garg, A. Puliafito, and K. Trivedi, "Analysis of Software Rejuvenation Using Markov Regenerative Stochastic Petri Net," *Proc. Sixth Int'l Symp. Software Reliability Eng.*, 1995.
[50] I. Cohen and J. Chase, "Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control," *Proc. Conf. Operating Systems Design and Implementation (OSDI)*, 2004.
[51] P. Akhlaghi, A. Kashanipour, and K. Salahshoor, "Complex Dynamical System Fault Diagnosis Based on Multiple ANFIS Using Independent Component," *Proc. 16th Mediterranean Conf. Control and Automation*, 2008.
[52] M. Kano, S. Hasebe, and I. Hashimoto, "Evolution of Multivariate Statistical Process Control: Application of Independent Component Analysis and External Analysis," *Proc. Conf. Foundations of Computer Aided Process Operations*, 2003.
[53] S. Makeig, A. Bell, T. Jung, and T. Sejnowski, "Independent Component Analysis of Electroencephalographic Data," *Advances in Neural Information Processing Systems*, vol. 8, pp. 145-151, 1996.
[54] J. Barton, E. Czeck, Z. Segall, and D. Siewiorek, "Fault Injection Experiments Using FIAT," *IEEE Trans. Computers*, vol. 39, no. 4, 1990.
[55] N. Maruyama and S. Matsuoka, "Model-Based Fault Localization in Large-Scale Computing Systems," *Proc Int'l Parallel and Distributed Processing Symp. (IPDPS)*, 2008.
[56] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated Support for Classifying Software Failure Reports," *Proc. Int'l Conf. Software Eng. (ICSE)*, 2003.
[57] D. Lee and S. Seung, "Learning the Parts of Objects by Non-Negative Matrix Factorization," *Nature*, vol. 401, pp. 788-791, 1999.
[58] F. Bach and M. Jordan, "Kernel Independent Component Analysis" *J. Machine Learning Research*, vol. 3, pp. 1-48, 2002.

Zhiling Lan received the BS degree in mathematics from Beijing Normal University, the MS degree in applied mathematics from the Chinese Academy of Sciences, and the PhD degree in computer engineering from Northwestern University in 2002. She is currently an associate professor of computer science at Illinois Institute of Technology. Her main research interests include fault-tolerant computing, dynamic load balancing, and performance analysis and modeling. She is a member of the IEEE Computer Society.



Ziming Zheng received the BS and MS degrees from the University of Electronic Science and Technology of China in 2003 and 2006, respectively. He is currently working toward the PhD degree in computer science at Illinois Institute of Technology since 2007. His research focuses on fault tolerance in large-scale computer systems. He is a student member of the IEEE.



Yawei Li received the BS and MS degrees from the University of Electronic Science and Technology of China in 1999 and 2002, respectively, and the PhD degree in computer science from Illinois Institute of Technology in 2009. He is currently with Google, Inc. He specializes in parallel and distributed computing, in particular, adaptive fault management and checkpointing optimization. He is a member of the IEEE.