An Anomaly Detection Framework Based on ICA and Bayesian Classification for IaaS Platforms

GuiPing Wang, JianXi Yang, and Ren Li
College of Information Science and Engineering, Chongqing Jiaotong University
Chongqing, 400074, China
[e-mail: w_guiping@163.com]
*Corresponding author: GuiPing Wang

Received January 5, 2016; revised May 3, 2016; accepted May 31, 2016; published August 31, 2016

Abstract

Infrastructure as a Service (IaaS) encapsulates computer hardware into a large amount of virtual and manageable instances mainly in the form of virtual machine (VM), and provides rental service for users. Currently, VM anomaly incidents occasionally occur, which leads to performance issues and even downtime. This paper aims at detecting anomalous VMs based on performance metrics data of VMs. Due to the dynamic nature and increasing scale of IaaS, detecting anomalous VMs from voluminous correlated and non-Gaussian monitored performance data is a challenging task. This paper designs an anomaly detection framework to solve this challenge. First, it collects 53 performance metrics to reflect the running state of each VM. The collected performance metrics are testified not to follow the Gaussian distribution. Then, it employs independent components analysis (ICA) instead of principal component analysis (PCA) to extract independent components from collected non-Gaussian performance metric data. For anomaly detection, it employs multi-class Bayesian classification to determine the current state of each VM. To evaluate the performance of the designed detection framework, four types of anomalies are separately or jointly injected into randomly selected VMs in a campus-wide testbed. The experimental results show that ICA-based detection mechanism outperforms PCA-based and LDA-based detection mechanisms in terms of sensitivity and specificity.

Keywords: Anomaly detection, feature extraction, ICA, Bayesian classification, IaaS.

1. Introduction

As an important application paradigm of cloud computing, Infrastructure as a Service (IaaS) encapsulates hardware resources into a virtual resource pool, thus providing users with on-demand accessible, elastic, and scalable service. Along with the rapid development and widespread application of cloud computing, the scale and the complexity of an IaaS platform continue to grow. Thousands of, even tens of thousands of manageable instances (i.e., virtual machines, VMs) are included in an IaaS platform ([1]). Currently, VM anomaly incidents occasionally occur, which leads to performance issues, and even downtime. Therefore, anomaly detection (maybe combined with further anomaly identification and location) is a key function to ensure stable operation for IaaS.

Usually, performance degradation or even downtime of VMs appears gradually along with anomalous consumption of physical or virtual resources including CPU, memory, I/O, and network. The anomalous consumption is reflected in various performance metrics of VMs. In order to better characterize a VM's state, dozens or even hundreds performance metrics are collected ([2]). However, first, high-dimensional datasets increase the complexity of data processing, which causes much CPU time. Second, there exists much correlation and redundancy among performance metrics. Some metrics may even be mixed with measurement noise. This may reduce the accuracy of anomaly detection and thus lead to the requirement for feature extraction techniques. Last but not least, as testified in the experiments of Section 4.1, the collected performance metrics data under Cloud environment usually do not follow the Gaussian distribution. Therefore, the feature extraction techniques (e.g., PCA, LDA) based on Gaussian distribution assumption cannot guarantee excellent performance.

In sum, detecting anomalous VMs from voluminous non-Gaussian monitored performance data (up to GB per day for a typical IaaS platform) is a challenging task due to the highly dynamic nature and increasing scale of IaaS. In order to successfully solve this issue, this paper designs an anomaly detection framework for IaaS, and proposes an anomaly detection algorithm based on independent components analysis (ICA) and Bayesian classification. One concern of this paper is to transform the original high-dimensional performance metric dataset into a low-dimensional space, while reserving the most useful information at the same time. Principal component analysis (PCA) is a traditional feature extraction and dimensionality reduction method. As an extension of PCA, ICA can extract independent components from non-Gaussian performance metric data. The experimental results show that ICA-based feature extraction preserves most relevant information in performance metric data, thus improving detection sensitivity and specificity.

The main contributions of this paper are listed as follows: 1) It designs an anomaly detection framework, which can detect anomalous VMs from a large amount of VMs in IaaS; 2) It introduces ICA-based feature extraction and compares it with PCA-based technique; 3) It verifies the effectiveness of the designed detection framework through anomaly injection.

2. Related Work

2.1 Feature Selection and Feature Extraction

Generally, in order to characterize the status of an observed system, people need to measure and collect variables which can reflect the system's state. A *feature* is defined as any individually measurable variable of an observed system ([3]). For a VM deployed upon a physical host in an IaaS environment, its performance is affected by both the VM itself and the underlying physical host. Therefore, any performance metrics from subsystems (including CPU, memory, I/O and network) of both the VM itself and the physical host can be measured to reflect the current healthy state of the VM.

Note that, to avoid confusion, in the remainder of this paper, each original performance attribute of a VM or the underlying physical host is called a "(*performance*) *metric*", while each component in the transformed data after PCA-based or ICA-based feature extraction is called a "*feature*".

Generally, in order to better understand the state of an observed system, as many metrics as possible should be collected. However, some metrics may be correlated, while other metrics may be noise. *Feature selection* or *feature extraction* can be introduced to extract most relevant information from original performance metric data, and reduce dimensionality as well.

Feature selection techniques (e.g., [4], [5]) select an optimal subset of the original metrics based on some criteria. However, feature selection usually involves exhaustive searching all the possible subsets of the set of original metrics, thereby being time-consuming ([3]). In contrast, feature extraction techniques transform the original metric data into a lower dimensional feature space in which the information most relevant to the observed system is preserved ([6]). Usually, feature data represented by lower dimensional space are more likely to be separated into different classes, which is beneficial to the subsequent anomaly detection process.

Note that, since the dimensionality of performance metric data can be reduced at the same time when extracting features, the following two terms, "*dimensionality reduction*" and "*feature extraction*", are interchangeable in the remainder of this paper.

PCA is a traditional feature extraction technology. It uses an orthogonal transformation to convert a set of observations of possibly correlated variables (i.e., metrics) into a set of values of linearly uncorrelated variables (i.e., features) called principal components. PCA was invented in 1901 by Karl Pearson ([7]). Since then, PCA is widely employed in many domains (e.g., [8][9]) as the traditional dimensionality reduction technology.

ICA ([10]) is a powerful data analysis tools emerged in recent years. ICA is applied to find underlying independent factors or components from multivariate statistical data. It is considered as an extension of PCA and factor analysis (FA).

The elementary idea of ICA was first proposed by Heranlt and Jutten in 1986 ([11]). Comon gave a rigorous mathematical definition of ICA in 1994 ([12]). Although the history of ICA is not long compared with PCA, it attracts more and more attention and becomes a hot research topic either in theory or application. From the application point of view, the prospect of ICA is very broad. Currently, ICA is mainly used in feature extraction, blind source separation, fault diagnosis and anomaly detection ([3], [13]), speech recognition, image processing, biomedical signal processing, financial time series analysis, data mining and other fields.

FastICA ([14]) is an efficient and popular algorithm for ICA invented by Aapo Hyvärinen and Erkki Oja. The algorithm is based on a fixed-point iteration scheme maximizing non-Gaussianity as a measure of statistical independence. Due to the fast convergence merit of FastICA, this paper employs it to implement ICA-based feature extraction.

Other linear dimensionality reduction technologies include linear discriminant analysis (LDA, [15]), non-negative matrix factorization (NMF). Non-linear dimensionality reduction technologies (e.g., isomap) are not considered in this paper due to high computational complexity.

2.2 Anomaly Detection of Distributed Systems

Anomaly detection is an important issue that has been researched within diverse research areas and application domains including traditional computer networks ([13]), Ad-Hoc Networks ([17]), Wireless Sensor Networks (WSNs) ([18]), Cloud computing systems ([2], [3], [19]-[25]). Anomalies are usually defined as patterns in data that do not conform to a well-defined notion of normal behavior ([16]).

Chandola et al. ([16]) survey the researches on anomaly detection and provide a structured and comprehensive overview. Based on the underlying approaches adopted by existing techniques in anomaly detection, they classify these techniques into six different categories: classification-based, nearest neighbor-based, clustering-based, statistical, information theoretic, spectral. They analyze the advantages and disadvantages of the techniques in each category. They also discuss and compare the computational complexity of the techniques.

In order to detect and prevent routing attacks (i.e., anomalies relative to normal routing) in WSNs, Hai et al. propose a lightweight intrusion detection framework ([18]). The proposed framework utilizes hierarchical architecture. Each sensor node is equipped with an IDS (Intrusion Detection System) agent, which includes two intrusion modules, i.e., local IDS agent and global IDS agent. The framework minimizes the triggered intrusion modules, and reduces alert packets by using an over-hearing mechanism, thus consuming less energy.

For anomaly detection in distributed systems including Cloud systems, existing researches in literature determine the current state of the system or components based on a variety of monitoring data including network traffic ([13]), system logs ([19]), system call, and performance metric data ([2], [3], [19]-[25]).

For network anomalies (i.e., the network behavior deviates from its normal operational baseline), Palmieri et al. ([13]) develop a two-stage anomaly detection strategy. The first step, ICA, extracts the fundamental network traffic components. These obtained components are used to build the baseline traffic profiles needed in the second phase, i.e., machine learning-inferred decision trees, to classify the current network state as anomalous or normal. The designed detection mechanism is a binary classification scheme, and therefore cannot deal with multiple categories of network anomalies.

Mi et al. ([19]) present a framework – CloudDiag, to diagnoses performance problems of Cloud computing systems. CloudDiag periodically collects the end-to-end tracing data (especially, execution time of method invocations), and employs a customized Map-Reduce algorithm to proactively analyze the tracing data. However, the collected monitoring data in CloudDiag cannot fully reflect the state of Cloud computing systems.

In order to analyze the correlation of various performance metrics with failure occurrences in virtualization and non-virtualization environments, Guan et al. ([2]) design a Cloud dependability analysis framework - CDA. Fault injection agents inject random or specific faults into multiple layers including hypervisor, VMs, and applications. Health monitoring sensors reside on each cloud server and periodically record a set of health related performance metrics. The main purpose of CDA is to gain insight into the impact of virtualization on the cloud dependability. Therefore, the function of CDA is not comprehensive enough. It cannot automatically detect anomalous VMs.

Sharma et al. ([21]) employ invariant relationships between performance metrics to detect and locate faults in distributed systems, where invariant refers to dependencies between two or more metrics that hold across time. However, the collected metrics in the anomaly detection system designed in this paper have no direct invariant relationships.

Pannu et al. ([23]) present an adaptive anomaly detection (AAD) framework for Cloud computing infrastructure (i.e., IaaS). AAD monitors the health states of physical servers, and constructs a hypersphere to cover the majority of sample points. Those sample points outside the hypersphere are identified as possible anomalies. AAD reports the identified anomalous states to operators for verification. The verified detection results will be input back to AAD for adaptation. However, a single hypersphere cannot accurately characterize VMs' complicated and diversified behaviors. In addition, AAD cannot deal with multiple categories of anomalies.

In order to cope with new threats to Cloud infrastructures, Watson et al. ([24]) propose an online anomaly detection approach based on one-class Support Vector Machine (SVM). The approach utilizes features gathered at the system and network levels of a Cloud node. For the raw features per process gathered at the system level, they build statistical meta-features such as the mean, variance and standard deviation of each feature across all processes. While the raw network features are extracted using the CAIDA CoralReef suite of tools. After feature extraction, the resulting feature vector totally includes 21 features, which is still a high value considering the voluminous monitored data collected from high-dynamic and large-scale Cloud systems.

Since classic anomaly detection schemes fail in detecting anomalies in the process of VM live migration, Huang et al. proposes an adaptive scheme to detect abnormal statistics when VMs are migrated to new hosts ([25]). They extend classic Local Outlier Factors (LOF) approach by defining novel dimension reasoning rules to figure out the possible sources of anomalies. They adopt resources utilization statistics (i.e., performance metrics) monitored by System Activity Reporter (SAR) to characterize the behavior of a VM. VM live migration is outside the scope of this paper, but their research work provides a good reference for our future work.

In sum, the anomaly detection framework under Cloud environment faces such challenges as high-dimensionality and diversity of monitoring data, the dependency among data, non-Gaussian monitored data, complexity and diversity of anomalies. Despite the above existing researches in designing anomaly detection frameworks under Cloud environment, a framework for accurately detecting anomalous VMs in real time under large-scale and high-dynamic Cloud environment is still a challenging work. For an IaaS environment, this paper introduces an ICA-based technique to extract features from collected performance metrics data of VMs and physical hosts, and further detect anomalous VMs.

3. The Proposed Anomaly Detection Framework

Fig. 1 illustrates the architecture of a typical Cloud data center. The physical hosts are grouped into several clusters, including computing clusters and storage clusters. The computing clusters constitute a virtual computing resource pool. Each host in a computing cluster is divided into multiple execution environments (i.e. VMs) and provides computing power. The management server is responsible for the allocation and management of virtual resources, such as VM creation, deletion, migration, computation and storage capacity assignment. Users can access their own VM instances in the Cloud data center across network, and deploy their application systems upon their VMs.

Fig. 2 shows the hardware/software architecture of a physical host in computing clusters. Multiple VM instances run simultaneously on each physical host, sharing the whole underlying hardware resource. In an IaaS environment, VMs are mutually isolated. Anomalies in a VM will not affect other VMs. Therefore, this paper takes VMs in IaaS as the object of anomaly detection.

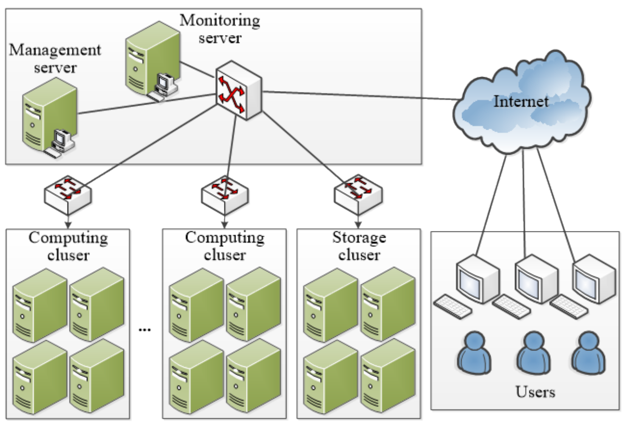

Fig. 1. The architecture of a Cloud data center.

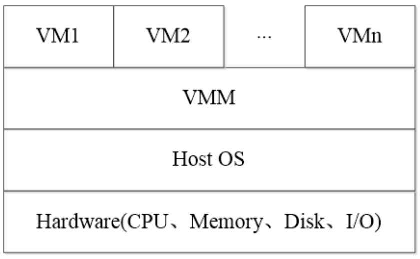

Fig. 2. The hardware/software architecture of a physical host.

Fig. 3 illustrates the framework of the designed anomaly detection system. The *data collection* module in each VM transmits the collected performance metrics data to a dedicated monitoring server deployed in IaaS. The monitoring server stores the received performance data locally, and determines the state of each VM through two key modules: *feature extraction* and *anomaly detection*, which are discussed in detail in the following sections.

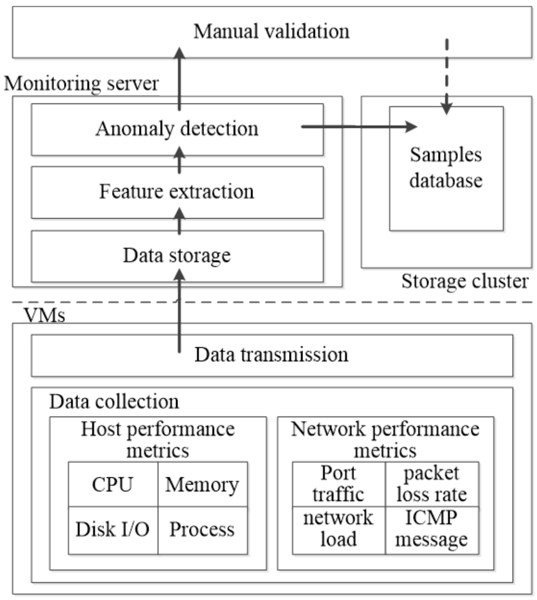

Fig. 3. The framework of the designed anomaly detection system.

The list of detected anomalous VMs is sent to human operators for final validation. The validated detection results are stored in the *samples database* deployed in storage cluster. The database stores both normal and anomalous detected results (i.e., dimensionality reduced features samples plus corresponding labels) spanned the past period of time (e.g., a week or a month) as training samples for future automated detection. The number of stored samples depends on the assigned storage volume of the database.

4. Data Collection and Feature Extraction

4.1 Data Collection

For a VM in an IaaS environment, the consumption of virtual hardware resources (including CPU, memory, disk I/O, network bandwidth) not only reflects the current running state of the VM, but also determines whether there exist anomalies in the VM. The healthy state of a VM also depends on the state of the underlying physical host. Therefore, the designed anomaly detection framework collects real-time performance metrics of VMs and physical hosts, and then detects anomalies based on the availability of physical/virtual resources. Since anomaly detection is based on the analysis of collected metrics, a guarantee for the designed detection system is to ensure the comprehensiveness and accuracy of collected metrics.

The collected performance metrics can be classified into two categories: *host performance metrics* and *network performance metrics*, as shown in **Fig. 3**. Further, the metrics in the former category can be classified into four sub-categories: *computation*, *storage*, *disk I/O*, *process*. Totally, 53 performance metrics are collected, which are listed as follows:

1) Computation resource performance metrics: the former 11 metrics listed in **Table 1** reflect the current computation resource utilization of a VM.

2) Storage resource performance metrics: the 12 metrics (i.e., 12nd-23rd) listed in **Table 1** reflect the current storage resource utilization of a VM.

3) Disk I/O performance metrics: the 9 metrics (i.e., 24th-32nd) listed in **Table 1** reflect the current disk I/O performance of a VM.

4) Process performance metrics: the 6 metrics (i.e., 33rd-38th) listed in **Table 1** reflect the current process performance of a VM.

5) Network performance metrics: the 15 metrics (i.e., 39th-53rd) listed in **Table 1** reflect the current network performance of a VM.

Table 1. The collected performance metrics

Metrics	Description
cpu_idle	Percentage of CPU idle time.
cpu_user	Percentage of CPU utilization that occurred while executing at the user level.
cpu_system	Percentage of CPU utilization that occurred while executing at the system level.
cpu_nice	Percentage of CPU utilization that occurred while executing at the user level with nice priority.
cpu_iowait	Percentage of CPU time waiting I/O operations.
cpu_irq	Percentage of CPU time spent to service hardware interrupts.
cpu_softirq	Percentage of CPU time spent to service software interrupts.
cpu_cs	Percentage of CPU time for process switch.
cpu_running	Number of running tasks in queues.
vcpu_run	Runtime of VCPU
vcpu_run_rate	Percentage of VCPU utilization

continued

Metrics	**Description**
mem_swapd	Amount of occupied swap space.
mem_free	Amount of available memory.
mem_buf	Buffer size of block device.
mem_cache	Buffer size of character device.
mem_si	Amount of virtual memory per second read from physical disk.
mem_so	Amount of virtual memory per second written to physical disk.
mem_total	Total amount of physical memory.
mem_slab	Amount of memory for kernel.
vmem_cur	Amount of virtual memory in VM.
vmem_rate	Utilization rate of virtual memory in VM.
vmem_max	Maximum amount of occupied virtual memory in VM.
vmem_max_rate	Maximum utilization rate of virtual memory in VM.
disk_await	The average wait time (in milliseconds) for I/O requests issued to the device to be served.
disk_svctm	The average service time (in milliseconds) for I/O requests issued to the device to be served.
disk_util	Percentage of I/O time per second.
disk_r	Number of operations of reading I/O devices per second.
disk_w	Number of operations of writing I/O devices per second.
disk_queue	Average length of I/O queues.
disk_rq	Average amount of data in an I/O operation.
vbd_rd	Number of operations of reading virtual block devices per second.
vbd_wr	Number of operations of writing virtual block devices per second.
pro_size	Total amount of memory occupied by processes.
pro_share	Total amount of share memory occupied by processes.
pro_cpu	Percentage of CPU time occupied by processes.
pro_mem	Percentage of memory occupied by processes.
pro_time	Total CPU time occupied by processes.
pro_thread_count	Number of threads.
net_rx_byte	Network data received per second.
net_tx_byte	Network data transmitted per second.
net_rx_packet	Number of received packets per second.
net_tx_packet	Number of transmitted packets per second.
net_rx_loss	Number of lost packets in receiving per second.
net_rx_loss	Number of lost packets in transmitting per second.
icmp_rx_ packet	Number of received ICMP packets per second.
icmp_tx_packet	Number of transmitted ICMP packets per second.
icmp_rx_loss	Number of unreachable packets in received ICMP packets per second.
icmp_tx_loss	Number of unreachable packets in transmitted ICMP packets per second.
net_load	Network load rate.
vnet_rx	Virtual network data received per second.
vnet_tx	Virtual network data transmitted per second.
vnet_rx_rate	Rate of virtual network data received per second.
vnet_tx_rate	Rate of virtual network data transmitted per second.

In order to testify the probability distribution of collected performance metrics, all metrics of a random selected VM are sampled every 10 seconds lasting 3 hours. Therefore, 1080 sampled values are obtained. In **Fig. 4**, four performance metrics are optionally chosen, and their histograms are plotted. From this figure, it is testified that the performance metrics data under Cloud environment usually do not follow the Gaussian distribution.

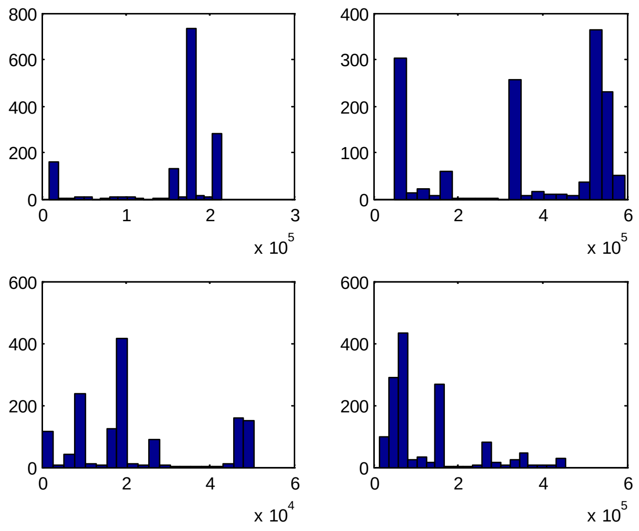

Fig. 4. The histograms of four optionally chosen performance metrics.

4.2 Notations and Preliminaries

1) In this paper, an italic and lowercase letter (maybe with one or multiple subscript(s)) (e.g., x_{ij}) represents a scalar value.

2) A bold and lowercase letter (maybe with a subscript) (e.g., $\mathbf{x}_i$) represents a vector; moreover, all vectors in this paper are column vectors.

3) A bold and uppercase letter (e.g., $\mathbf{X}$) represents a matrix.

4) An italic and uppercase letter (e.g., X) represents a random variable.

5) A bold, italic and uppercase letter (e.g., $\boldsymbol{X}$) represents a random vector. Random vectors in this paper are also column vectors.

Assume that an observed system has m metrics, $\{X_i\}$, i = 1, …, m. Each metric can be considered as a random variable. These m metrics constitute a random vector, $\boldsymbol{X}$.

$$\boldsymbol{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix}. \tag{1}$$

The data of these m metrics in t time points are sampled, whose values constitute an m-by-t original data matrix, $\mathbf{X}$, as shown in Equation (2), e.g, in this paper, m = 53, and t = 360 (sampled every 10 seconds lasting 1 hour).

In $\mathbf{X}$, each row represents all the sample data of a performance metric, X_i; while each column represents the sampling values of all metrics in a point-in-time.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1t} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2t} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \cdots & x_{mt} \end{bmatrix} \tag{2}$$

Assuming the data matrix $\mathbf{X}$ is zero-mean (i.e., centered), then the covariance matrix of the random vector $\boldsymbol{X}$ or the data matrix $\mathbf{X}$, denoted as $\mathbf{C}_\mathbf{X}$, can be obtained by:

$$\mathbf{C}_\mathbf{X} = \frac{1}{t-1}\mathbf{X}\mathbf{X}^T. \tag{3}$$

$\mathbf{C}_\mathbf{X}$ has the following properties:

1) $\mathbf{C}_\mathbf{X}$ is an m-by-m symmetric matrix.

2) The diagonal element of $\mathbf{C_X}$, $\sigma^2_{X_iX_i}$, is the variance of metric X_i.

3) The non-diagonal element of $\mathbf{C_X}$, $\sigma^2_{X_iX_j}$, is the covariance between metrics X_i and X_j.

$\mathbf{C_X}$ reflect the degree of data redundancy or noise:

1) A larger diagonal element indicates a stronger signal, which means it is a more important metric. However, a smaller diagonal element indicates the corresponding metric may be a less important one or there exist noise in the corresponding metric.

2) The values of non-diagonal elements correspond to the degree of redundancy (linear correlation) between metrics.

4.3 PCA-based Feature Extraction

PCA makes four assumptions ([26]): 1) *Gaussianity*; 2) *Linearity*; 3) *Large variances have important structure*; 4) *the principal components are orthogonal*.

PCA tries to solve an orthogonal transformation, which can best re-express original data. Let $\mathbf{Y} = \mathbf{P}^T\mathbf{X}$ be the transformed new *m*-by-*t* data matrix. PCA minimizes redundancy between transformed features (i.e., the non-diagonal elements of $\mathbf{C_Y}$ should be as small as possible), and maximizes transformed features (i.e., the diagonal elements of $\mathbf{C_Y}$ should be as large as possible and in descending order).

The linear transformation from **X** to **Y** is given by:

$$\mathbf{Y}_{m\times t} = \mathbf{P}^T_{m\times m}\mathbf{X}_{m\times t}. \tag{4}$$

Assume that the eigenvalues of $\mathbf{C_X}$ are $\lambda_1, \lambda_2, \dots, \lambda_m$ (in descending order), the corresponding eigenvectors are $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m$. According to the knowledge in linear algebra, the eigenvectors, $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m$ (in the form of column vectors), form a matrix, $\mathbf{P}$. The transpose, $\mathbf{P}^T$, is just the required transformation matrix. The eigenvectors, $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m$, also constitute the set of orthonormal bases for transformed data matrix.

For dimensionality reduction, assuming the cumulative contribution rate of the former *s* principal components meets the threshold *h* (e.g., 0.85 or 0.90), namely:

$$\frac{\sum_{i=1}^{s}\lambda_i}{\sum_{i=1}^{m}\lambda_i} \geq h, \tag{5}$$

then the former *s* principal components constitutes the transformation matrix $\mathbf{P}_s^T$ to transform the original data matrix **X** into a new data set with *s* dimensionalities, and $\mathbf{Y}_s = \mathbf{P}_s^T\mathbf{X}$ is the *s*-by-*t* data matrix with reduced dimensionality.

Therefore, PCA-based feature extraction works as follows:

Step 1: First, it organizes the sampled data into an *m*-by-*t* original data matrix, **X**, where *m* is the number of performance metrics, *t* is the number of samples.

Step 2: It centers the data matrix, **X**; and computes the covariance matrix of **X**, i.e., $\mathbf{C_X}$.

Step 3: It computes the eigenvalues of $\mathbf{C_X}$ (denoted as $\lambda_1, \lambda_2, \dots, \lambda_m$, in descending order), and obtains the corresponding eigenvectors, denoted as $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m$.

Step 4: Assuming the cumulative contribution rate of the former *s* principal components meets a pre-defined threshold *h*, it then takes the former *s* eigenvectors, $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_s$, to constitute the transformation matrix, $\mathbf{P}_s^T$.

Step 5: It transforms the original data matrix using $\mathbf{P}_s^T$, i.e., $\mathbf{Y}_s = \mathbf{P}_s^T\mathbf{X}$.

4.4 ICA-based Feature Extraction

ICA is a data analysis method finding underlying factors or components from multivariate statistical data. The essential difference distinguishing ICA from PCA is that ICA finds both statistically independent and non-Gaussian components in multivariate data.

Assume that the multivariate data collected from an observed system consist of *m* metrics. ICA seeks a transformation function from an *m*-dimensional space to an *n*-dimensional space such that the transformed variables are the underlying factors or components of the observed data; moreover, the obtained components are expected to describe the essential structure of the data or even generate the data.

This paper first assumes the number of independent components is equal to the number of observed variables (i.e., $m = n$), which in fact is not a rigorous assumption in ICA.

ICA only considers linear transformation. Therefore, every underlying component, denoted as Y_i, is represented as a linear combination of the observed variables:

$$Y_i = \sum_j w_{ij} X_i\,,\ \ i, j = 1, \dots, m. \tag{6}$$

where w_{ij} are some coefficients needed to solve. The above linear transformation can be expressed as a matrix multiplication:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix} = \mathbf{W}_{m \times m} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix}. \tag{7}$$

In ICA, the principle determining **W** is independence, i.e., the components Y_i should be statistically independent. ICA makes the following two basic assumptions ([10]):

1) The components underlying the observed data (or generating the data) are assumed statistically independent.

2) The components underlying the observed data must have non-Gaussian distributions.

ICA does not make any assumptions about the gaussianity of the observed data. Considering the non-gaussianity of the performance metrics, ICA is more appropriate than PCA for feature extraction when applied to anomaly detection in IaaS.

A useful pre-processing step in ICA is *whitening*. In ICA-based feature extraction, the whitening problem is concreted as: given the original data matrix, **X**, find a linear transformation, **V**, such that the transformed data matrix, $\mathbf{Z} = \mathbf{VX}$, is white.

Assume $\mathbf{E} = [\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m]$ is the matrix whose columns are the unit-norm eigenvectors of the covariance matrix, $\mathbf{C_X}$; $\mathbf{D} = diag(\lambda_1, \lambda_2, \dots, \lambda_m)$ is the diagonal matrix whose diagonal elements are the eigenvalues of $\mathbf{C_X}$, then a linear whitening transform is given as:

$$\mathbf{V} = \mathbf{D}^{-1/2}\mathbf{E}^T. \tag{8}$$

After whitening, ICA projects the data matrix **X** into a data matrix **Y** as below:

$$\mathbf{Y} = \mathbf{WX}. \tag{9}$$

The goal of ICA is to find an optimal projection matrix, **W**, such that the transformed features Y_i are maximally independent, measured by some statistics or functions ([10]), such as kurtosis, negentropy, and non-Gaussianity. In general, ICA employs an iterative process to search for **W**. FastICA ([14]) is a fast converged ICA algorithm, which is based on a fixed-point iteration scheme maximizing non-Gaussianity.

The FastICA algorithm works as follow:

Step 1: It centers the *m*-by-*t* original data matrix $\mathbf{X} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_t]$ to make its mean zero.

Step 2: It whitens the data to give a new white data matrix $\mathbf{Z} = [\mathbf{z}_1 \quad \mathbf{z}_2 \quad \cdots \quad \mathbf{z}_t]$.

Step 3: It chooses the number of independent components to estimate, *s*.

Step 4: It chooses a random initial matrix, $\mathbf{W} = [\mathbf{w}_1, \ \mathbf{w}_2, \ \dots, \mathbf{w}_s]$, where $\|\mathbf{w}_i\| = 1$.

Step 5: For $i = 1$ to s

$$\mathbf{w}_i^+ = \frac{1}{t}\sum_{j=1}^{t} \mathbf{z}_j \, (\mathbf{w}_i^T \mathbf{z}_j)^3 - 3\mathbf{w}_i, \tag{10}$$

$$\mathbf{w}_i = \mathbf{w}_i^+ / \|\mathbf{w}_i^+\|. \tag{11}$$

The above iterative process ensures that the iteration approaches the maximal independence (Equation 10), and that $\|\boldsymbol{w}_i\| = 1$ (Equation 11).

Step 6: It symmetrically orthogonalizes the matrix **W** by:

$$\mathbf{W} = \mathbf{W}(\mathbf{W}^T\mathbf{W})^{-1/2}. \tag{12}$$

Step 7: If $\left(1 - \left|\mathbf{w}_i^T \mathbf{w}_i\right|\right) < \delta$, where δ is a small predefined threshold which indicates the end of the iteration, FastICA converges; Otherwise, it goes to step 5 for next iteration.

In practice, FastICA works very well on performance metric data. It converges very fast. The convergence speed is cubic ([14]).

5. Anomaly Detection

5.1 Definitions

Combining the definitions in literature ([13], [16]), the formal definitions of anomaly and anomaly detection in this paper are given as follows.

Assumption: although VM anomaly incidents occur occasionally in IaaS, anomalies are rare events after all, and the majority of VMs operate normally.

Definition 1 (Anomaly): Anomalies in a VM are patterns in resource consumption (including computation, storage, disk I/O, network) that the VM does not conform to the majority of other VMs. Anomalies can be caused by faults or failures.

Note that, this paper do not aim to discover the root causes of faults and failures. It only detects anomalous VMs and reports to human operators.

Definition 2 (Anomaly detection): It refers to the problem of finding VMs with anomalous patterns in resource consumption.

5.2 Anomaly Detection based on Bayesian Classification

Bayesian classification is a simple but effective supervised classification method. A naive Bayes classifier is a simple probabilistic one using Bayes' theorem with strong independence assumptions. It can be used to binary or multi-class classification.

Binary classification: Given s transformed system features, $X_1, X_2, \dots, X_s$, and their values in the current time, $x_1, x_2, \dots, x_s$, the task of anomaly detection is to determine the current state of the system is in the normal state (C_{normal}), or the abnormal state ($C_{abnormal}$) according to the current values.

Note that, Since Bayesian classification is based on strong independence assumptions, the given s system features, $X_1, X_2, \dots, X_s$, are expected to be mutually independent. If they are not independent, they can be transformed by PCA (Gaussian distribution case) or ICA (non-Gaussian distribution case) into independent features. Therefore, the transformed features always meet independence assumptions. Usually, the transformed data contain fewer features (e.g., s) after dimensionality reduction.

Let $\boldsymbol{X} = [X_1 \quad X_2 \quad \cdots \quad X_s]^T$ be the random vector including s system features, $\mathbf{x} = [x_1 \quad x_2 \quad \cdots \quad x_s]^T$ be the sampling value of $\boldsymbol{X}$.

The determination rules of Bayesian classification are listed as follows:

1) If $P(C_{normal}|\boldsymbol{X} = \mathbf{x}) > P(C_{abnormal}|\boldsymbol{X} = \mathbf{x})$, it determines that the system is currently in the normal state;

2) If $P(C_{normal}|\boldsymbol{X}=\mathbf{x})<P(C_{abnormal}|\boldsymbol{X}=\mathbf{x})$, it determines that the system is currently in the abnormal state;

3) If $P(C_{normal}|\boldsymbol{X}=\mathbf{x})=P(C_{abnormal}|\boldsymbol{X}=\mathbf{x})$, it randomly determines that the system is currently in the normal or abnormal state.

However, $P(C_{normal}|\boldsymbol{X}=\mathbf{x})$ and $P(C_{abnormal}|\boldsymbol{X}=\mathbf{x})$ can not be directly solved. According to the Bayes' theorem:

$$P(C_{normal}|\boldsymbol{X}=\mathbf{x}) = \frac{P(\boldsymbol{X}=\mathbf{x}|C_{normal})P(C_{normal})}{P(\boldsymbol{X}=\mathbf{x})}, \quad (13)$$

and

$$P(C_{abnormal}|\boldsymbol{X}=\mathbf{x}) = \frac{P(\boldsymbol{X}=\mathbf{x}|C_{abnormal})P(C_{abnormal})}{P(\boldsymbol{X}=\mathbf{x})}, \quad (14)$$

only the numerators need to be calculated.

Due to the independence assumption, the numerators can be re-expressed as:

$$P(\boldsymbol{X}=\mathbf{x}|C_{normal})P(C_{normal}) = P(C_{normal})\prod_{i=1}^{s}P(X_i=x_i|C_{normal}), \quad (15)$$

and

$$P(\boldsymbol{X}=\mathbf{x}|C_{abnormal})P(C_{abnormal}) = P(C_{abnormal})\prod_{i=1}^{s}P(X_i=x_i|C_{abnormal}). \quad (16)$$

The following values can be obtained from the training samples in samples database:

1) $P(C_{normal})$ and $P(C_{abnormal})$: the priori probabilities of each system state;

2) $P(X_i=x_i|C_{normal})$ and $P(X_i=x_i|C_{abnormal})$, i = 1, 2, …, s: the priori probability of the event $P(X_i=x_i)$ under known system state C_{normal} and $C_{abnormal}$, respectively.

3) $P(\boldsymbol{X}=\mathbf{x})=\prod_{i=1}^{s}P(X_i=x_i)$: in fact, this value can also be obtained, but it is useless in Bayesian classification.

The values required in Bayesian classification are $P(C_{normal}|\boldsymbol{X}=\mathbf{x})$ and $P(C_{abnormal}|\boldsymbol{X}=\mathbf{x})$, which are the posterior probabilities. Of course, as mentioned earlier, they can be solved according to the priori probabilities based on the Bayes' theorem, i.e., the Equations (13) and (14).

Note that, in practice, since the values of system features are often continuous, these values need to be discretized before applying Bayesian classification.

Multi-class classification: This situation is similar with the binary classification. Assuming there are k kinds of abnormal system states and one kind of normal state, the basic idea of Bayesian classification is to classify the current state of the system into a specific state with the greatest posterior probability according to the current features values.

6. Experiments and Analyses

6.1 Experiment Setup

The experiments are carried out on a campus-wide cloud computing testbed, in which the computing clusters consist of 60 physical computing nodes. Each node is equipped with an Intel Core i5-2400 CPU 3.10GHZ (with 4 cores) and 4 GB RAM. All the nodes are interconnected by 100Mbps Ethernet switches. 0-4 VMs are deployed on each node (the number is dynamically changed according to the deployment assigned by the management server). The configuration of each VM is similar in the number of virtual CPU and the amount of memory.

The storage cluster consists of 2 SAN servers, providing a virtual storage pool for the whole testbed.

In addition, a management server and a monitoring server are deployed in the testbed. Each server is equipped with an Intel Core i7-4930k CPU 3.40 GHZ (with 6 cores) and 16 GB RAM.

6.2 Parameters

The most important parameters are the number of independent components and that of principal components (both denoted as *s*) in ICA and PCA respectively.

Through experiments, it is found that 8 independent components is the best choice for ICA. More independent components do not apparently improve the performance of ICA-based anomaly detection algorithm. The underlying reason is that, there exists much correlation and redundancy among performance metrics; some metrics may even be mixed with measurement noise; therefore, the former few extracted features (i.e., independent components or principal components) preserves most useful information in performance metric data, while the latter features may be less important components or even noises. Therefore, *s* is set as 8 in ICA.

For the purpose of fair comparison, the parameter, *s*, is also set as 8 in PCA. The contribution rates of the former 8 principal components are shown in **Fig. 5**, whose cumulative contribution rate is 0.926 and larger than the commonly adopted threshold (e.g., 0.85, or 0.90).

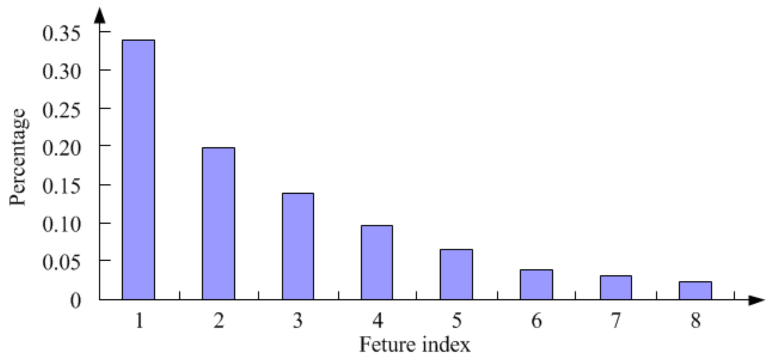

Fig. 5. The contribution rates of the former 8 principal components.

Therefore, through feature extraction, data reduction achieved by both PCA and ICA is over 84.9% (=45/53), which tremendously reduces the data volume of samples database.

6.3 Samples Database

Since Bayesian classification is a supervised method. It needs both normal and anomalous samples, the samples database in the storage cluster stores dimensionality reduced feature data of each VM and underlying physical host with validated detection results lasting a period of time up to one month. One of the limitations of the designed detection system is that initially it needs the human operators to validate the detection results to construct the samples database. After the system stably runs a short period of time, the human operators only need to validate the detected anomalous VMs. One future work of this paper is to exploit other unsupervised techniques (e.g., nearest neighbor-based) that do not need sample labels.

6.4 Anomaly Injection

In order to evaluate the performance of the designed detection system and compare ICA-based feature extraction with PCA-based feature extraction, four types of anomalies are injected to 0-30 randomly selected VMs, which are listed as follows.

1) *Computation resource over-consumption*: A computing-intensive program runs in a VM,

and persistently consumes computation resources.

2) *Memory resource over-consumption*: A running program in a VM continues to apply for dynamic memory through malloc() function without releasing the assigned memory, which causes memory leaks and over-consumes the VM's memory resources.

3) *Disk I/O anomaly*: A running program in a VM persistently reads large files on disk, generating large amounts of disk I/O operations to simulate anomalous disk I/O.

4) *Network anomaly*: Several users run *LoadRunner* from their own physical personal computer to simultaneously access a Web application server deployed on a VM, generating a large number of HTTP connections to simulate anomalous network behavior.

Note that, since the samples database stores the actual states of all VMs spanned the past period of time, these injected anomaly samples will not be included into the database.

6.5 Experimental Results and Discussions

For an anomaly detection system, a detected anomaly is referred to as a *positive*, while a detected normality is referred to as a *negative*.

Compared with the actual state of a VM, the detection results can be classified into four categories: *False positive* (F_P), *False negative* (F_N), *True positive* (T_P), and *True negative* (T_N).

Since anomalies are rare events (e.g., less than 1%), a detection system determining every VM as a normal VM still can achieve extreme high *accuracy* (e.g., over 99%). Note that, *accuracy* $=(T_P + T_N)/(T_P + T_N + F_P + F_N)$. Therefore, two more appropriate performance measures are introduced to evaluate the performance of the designed detection system.

1) *Sensitivity* is the proportion of correctly detected anomalous VMs to the number of actual anomalous VMs.

$$sensitivity = \frac{T_P}{T_P+F_N}. \tag{17}$$

2) *Specificity* is the proportion of correctly detected normal VMs to the number of actual normal VMs.

$$specificity = \frac{T_N}{F_P+T_N}. \tag{18}$$

The experiments evaluate the proposed anomaly detection mechanism from two aspects: a) To compare ICA with another two feature extraction techniques (i.e., PCA and LDA), the designed anomaly detection systems based on Bayesian classification are equipped with PCA-based, LDA-based, and ICA-based feature extraction respectively; b) The experiments also compare two anomaly detection mechanisms: ICA plus Bayesian classification (the proposed one), and ICA plus decision tree ([13]). Both these two aspects are conducted under two scenarios: single-anomaly and multi-anomaly.

The experimental results under four types of single injected anomaly are listed in **Table 2**. Each experiment is conducted 10 times, and the results are averaged. The first number in each cell of **Table 2** is sensitivity, and the second number is specificity.

Table 2. Performance measures under four types of injected anomaly (Single-anomaly)

Anomaly	PCA + Bayesian	LDA + Bayesian	ICA + Bayesian	ICA + Decision tree
Computation	0.961 / 0.912	0.968 / 0.936	0.977 / 0.956	0.974 / 0.952
Memory	0.968 / 0.951	0.977 / 0.951	0.982 / 0.963	0.979 / 0.975
Disk I/O	0.957 / 0.933	0.962 / 0.934	0.976 / 0.955	0.972 / 0.943
Network	0.942 / 0.908	0.946 / 0.912	0.968 / 0.931	0.963 / 0.917
Average	0.957 / 0.926	0.963 / 0.933	0.976 / 0.951	0.972 / 0.947

As shown in **Table 2**, ICA-based feature extraction outperforms another two techniques, which is reflected in the final detection results in terms of both sensitivity and specificity; while LDA-based feature extraction technique slightly outperforms PCA-based one. Averagely, ICA-based detection achieves extremely high sensitivity (0.976) and high specificity (0.951). When compared with the proposed anomaly detection mechanism (i.e., ICA plus Bayesian classification), ICA plus decision tree is less effective.

The experimental results under six types of multiple injected anomalies are listed in **Table 3**. Similarly, the first number in each cell is sensitivity, and the second number is specificity.

Table 3. Performance measures under four types of injected anomaly (Multi-anomaly)

Anomaly	PCA + Bayesian	LDA + Bayesian	ICA + Bayesian	ICA + Decision tree
Computation & Memeory	0.924 / 0.922	0.921 / 0.911	0.967 / 0.953	0.956 / 0.932
Computation & Disk I/O	0.892 / 0.873	0.914 / 0.904	0.958 / 0.923	0.942 / 0.918
Computation & Network	0.925 / 0.918	0.931 / 0.923	0.953 / 0.946	0.945 / 0.939
Memeory & Disk I/O	0.926 / 0.911	0.932 / 0.915	0.961 / 0.927	0.953 / 0.916
Memeory & Network	0.918 / 0.906	0.913 / 0.912	0.939 / 0.922	0.924 / 0.921
Disk I/O & Network	0.902 / 0.865	0.906 / 0.886	0.932 / 0.914	0.908 / 0.892
Average	0.915 / 0.899	0.920 / 0.909	0.952 / 0.931	0.938 / 0.920

As shown in **Table 3**, ICA-based feature extraction still outperforms another two techniques in terms of both sensitivity and specificity; while LDA-based technique still slightly outperforms PCA-based one. Although the detection specificity of ICA plus Bayesian classification drops in multi-anomaly scenario (averagely, 0.931), it still meet the practical requirements of anomaly detection systems. While the anomaly detection mechanism of ICA plus decision tree is still less effective than the proposed mechanism, especially in terms of sensitivity.

As for computation overhead, both these four detection mechanisms are time efficient due to the dimensionality-reduced feature data. In the experiments including single-anomaly and multi-anomaly scenarios, these four detection mechanisms take about 0.32 second to produce a list of anomalous VMs. Therefore, all of these detection mechanisms meet the real-time performance requirements of anomaly detection systems.

It is concluded that ICA-based mechanism outperforms PCA-based and LDA-based ones for both single-anomaly and multi-anomaly scenarios. On the other hand, Bayesian classification is slightly effective than decision tree. The main underlying reasons are listed as follow:

1) Since the performance metrics collected in this paper do not follow the Gaussian distribution, PCA cannot extract the most relevant information; whereas ICA does not make any assumptions about the probability distribution of the observed data.

2) LDA is a supervised feature extraction technique. It assumes that each category of samples follows the Gaussian distribution, which makes LDA less effective than ICA. However, LDA searches the best projection direction through maximizing the ratio of the variance between the classes to the variance within the classes, which is beneficial to separate data samples into different categories. Therefore, LDA slightly outperforms PCA.

3) Both PCA and ICA are unsupervised techniques. PCA maximizes variance by using the second order statistics (i.e., mean and covariance) in the whole data matrix, while ICA maximizes independence by using higher order statistics (which contain more useful information) in the data.

4) PCA extracts the uncorrelated components from original performance metrics, while ICA extracts the independent components. Since Bayesian classification is based on strong

independence assumptions, ICA-based feature extraction is most appropriate to Bayesian classification.

5) Bayesian classification is a kind of probabilistic classification method minimizing error rate. Decision tree tends to fit all the training samples, which makes it more likely to fall into the problem of over-fitting, thereby less effective in classifying new samples.

7. Conclusion and Future Work

This paper designs an anomaly detection framework for IaaS environment, and proposes an anomaly detection algorithm based on ICA and multi-class Bayesian classification. ICA is used to extract independent components from collected performance metrics. The supervised multi-class Bayesian classifier determines the current state of each VM as anomalous (multiple categories) or normal. The experimental results confirm the effectiveness of the designed detection system.

The limitations of the designed detection system (referred in the preceding sections) are left to be solved in future. In addition, the future work of this paper will mainly focus on anomaly identification and location, which will provide more support for the top decision-making applications.

Acknowledgment

The authors are grateful to the editors and anonymous reviewers for their valuable comments on this paper.

The paper is supported by National Natural Science Foundation of China (Grant No. 61272399 and 61572090), Chongqing Research Program of Basic Research and Frontier Technology (Grant No. cstc2015jcyjBX0014), and the Scientific and Technological Research Program of Chongqing Municipal Education Commission under (Grant No. KJ1500538).

References

[1] S. C. Meng, L. Liu, and T. Wang, "State Monitoring in Cloud Datacenters," *IEEE Trans. on Knowledge and Data Engineering*, vol. 23, no. 9, pp. 1328-1344, 2011. Article (CrossRef Link).

[2] Q. Guan, C. C. Chiu, and S. Fu, "CDA: A cloud dependability analysis framework for characterizing system dependability in cloud computing," in *Proc. of IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 11-20, 2012. Article (CrossRef Link).

[3] Z. L. Lan, Z. M. Zheng, and Y. W. Li, "Toward automated anomaly identification in large-scale systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 21, no. 2, pp. 174-187, 2010. Article (CrossRef Link).

[4] F. Marcelloni, "Feature selection based on a modified fuzzy C-means algorithm with supervision," *Information Science*, vol. 151, pp. 201-226, 2003. Article (CrossRef Link).

[5] L. I. Kuncheva, "A stability index for feature selection," in *Proc. of the 25th IASTED International Multi-Conference: artificial intelligence and applications*, pp. 390-395, 2007.

[6] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed., Wiley Interscience, 2001.

[7] K. Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space," *Philosophical Magazine*, vol. 2, no. 11, pp. 559–572, 1901. Article (CrossRef Link).

[8] M. Pechenizkiy, S. Puuronen, and A. Tsymbal, "The Impact of Sample Reduction on PCA-based Feature Extraction for Supervised Learning," in *Proc. of the 21st Annual ACM Symposium on Applied Computing*, pp. 553-558, April 2006. Article (CrossRef Link).

[9] O. Ibidunmoye, F. Hernandez-Rodriguez, and E. Elmroth, "Performance Anomaly Detection and Bottleneck Identification," *ACM Computing Surveys*, vol. 48, no. 1, Sep. 2015. Article (CrossRef Link).

[10] A. Hyvarinen, J. Karhunen, and E. Oja, *Independent Component Analysis*, John-Wiley & Sons, Inc., 2001.

[11] J. Herault, and C. Jutten, "Space or time adaptive signal processing by neural network models," in *Proc. of AIP Conference Proceedings*, vol. 15, no. 1 (on Neural Networks for Computing), pp. 206-211, Aug. 1986. Article (CrossRef Link).

[12] P. Comon, "Independent component analysis, a new concept?" *Signal Processing*, vol. 36, no. 3, pp. 287-314, 1994. Article (CrossRef Link).

[13] F. Palmieri, U. Fiore, and A. Castiglione, "A distributed approach to network anomaly detection based on independent component analysis," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 5, pp. 1113–1129, 2014. Article (CrossRef Link).

[14] A. Hyvärinen, and E. Oja, "A fast fixed-point algorithm for independent component analysis," *Neural Computation*, vol. 9, no. 7, pp. 1483-1492, 1997. Article (CrossRef Link).

[15] C. H. Li, B. C. Kuo, and C. T. Lin, "LDA-based Clustering Algorithm and Its Application to an Unsupervised Feature Extraction," *IEEE Trans. on Fuzzy Systems*, vol. 19, no. 1, pp.152-163, 2011. Article (CrossRef Link).

[16] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM Computing Surveys*, vol. 41, No. 3, Article 15, 2009. Article (CrossRef Link).

[17] M. Jo, L. Z. Han, D. Kim, and H. P. In, "Selfish Attacks and Detection in Cognitive Radio Ad-hoc Networks," *IEEE Network*, vol. 27, no. 3, pp. 46-50, 2013. Article (CrossRef Link).

[18] T. H. Hai, E. N. Huh, and M. Jo, "A Lightweight Intrusion Detection Framework for Wireless Sensor Networks," *Wireless Communications and Mobile Computing*, vol. 10, no. 4, pp. 559-572, 2010. Article (CrossRef Link).

[19] H. B. Mi, H. M. Wang, Y. F. Zhou, et al, "Toward Fine-Grained, Unsupervised, Scalable Performance Diagnosis for Production Cloud Computing Systems," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1245-1255, 2013. Article (CrossRef Link).

[20] Q. Guan, Z. M. Zhang, and S. Fu, "Ensemble of Bayesian predictors and decision trees for proactive failure management in cloud computing systems," *Journal of Communications*, vol. 7, no. 1, pp. 52-61, 2012. Article (CrossRef Link).

[21] A. B. Sharma, H. F. Chen, M. Ding, et al, "Fault detection and localization in distributed systems using invariant relationships," in *Proc. of 43rd Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, pp. 1-8, June 2013. Article (CrossRef Link).

[22] Y. B. Liu, Z. Yuan, C. C. Xing, et al., "A behavioral anomaly detection strategy based on time series process portraits for desktop virtualization systems," *Cluster Computing*, vol. 18, no. 2, pp. 979-988, 2015. Article (CrossRef Link).

[23] H. S. Pannu, J. G. Liu, and S. Fu, "AAD: Adaptive anomaly detection system for cloud computing infrastructures," in *Proc. of the IEEE Symposium on Reliable Distributed Systems*, 2012 Article (CrossRef Link).

[24] M. R. Watson, N. H. Shirazi, A. K. Marnerides, et al. "Malware Detection in Cloud Computing Infrastructures," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 192-205, 2016. Article (CrossRef Link).

[25] T. Huang, Y. X. Zhu, and Y. F. Wu, et al. "Anomaly detection and identification scheme for VM live migration in cloud infrastructure," *Future Generation Computer Systems*, vol. 56, pp. 736-745, 2016. Article (CrossRef Link).

[26] J. Shlens, "A Tutorial on Principal Component Analysis," (Dated: April 22, 2009; Version 3.01). http://www.snl.salk.edu/~shlens/pub/notes/pca.pdf.

GuiPing Wang received his B.S. degree, M.S. degree, and Ph.D. degree at Chongqing University, P. R. China, in 2000, 2003, and 2015, respectively. Currently he is a lecturer in College of Information Science and Engineering, Chongqing Jiaotong University, P. R. China. His research interests include machine learning, dependability analysis and design of distributed systems, and cloud computing. As the first author, he has published over 20 papers in related research areas during recent years at journals including IPL, Optik, and ESWA.

JianXi Yang received his Ph.D. degree at Chongqing Jiaotong University, P. R. China, in 2011. Currently he is a professor in College of Information Science and Engineering, Chongqing Jiaotong University, P. R. China. His research interests include state monitoring, fault diagnosis, big data processing, and cloud computing. He has published over 30 papers in related research areas during recent years.

Ren Li received his B.S. degree, M.S. degree, and Ph.D. degree at Chongqing University, P. R. China, in 2007, 2009, and 2013, respectively. Currently he is a lecturer in College of Information Science and Engineering, Chongqing Jiaotong University, P. R. China. His research interests include cloud computing, big data processing, and semantic web techniques. He has published over 10 papers in related research areas during recent years.