International Journal of Future Generation Communication and Networking
Vol.7, No.1 (2014), pp.1-18
http://dx.doi.org/10.14257/ijfgcn.2014.7.1.01

A Dependable Monitoring Mechanism Combining Static and Dynamic Anomaly Detection for Network Systems

GuiPing Wang [1], ShuYu Chen [2*], Zhen Zhou [1] and MingWei Lin [1]

[1] College of Computer Science, Chongqing University, Chongqing, China
[2] College of Software Engineering, Chongqing University, Chongqing, China
w_guiping@163.com, netmobilab@cqu.edu.cn, zhouzhen1302@163.com, linmwcs@163.com

Abstract

Due to abuse by insiders or penetration by outsiders, network systems usually suffer various security issues. In order to achieve high dependable and low cost monitoring, this paper proposes a dependable monitoring mechanism combining static threshold-based and dynamic anomaly detection. Firstly, the performance metrics of host and network are collected through different methods. In static threshold-based detection phase, the secondary metrics are combined to several group items. When any group item exceeds its threshold, dynamic detection methods are adopt to further detect anomaly. In dynamic detection phase, PCA, joint Gaussian distribution, and Bayesian classification are combined to achieve low cost and efficient anomaly detection. Experimental results in a campus-wide network system show that the proposed dependable monitoring mechanism achieves low false negative (FN) rate and low false positive (FP) rate. The proposed monitoring mechanism outperforms PCA & Bayesian, and grouping detection methods.

Keywords: *Dependable Monitoring, Anomaly Detection, Data Acquisition, PCA, Bayesian Classification*

1. Introduction

Along with continuous improvement of network technologies and wide popularity of network applications, human are increasingly dependent on networks. However, network systems are vulnerable to abuse by insiders or penetration by outsiders. Providing high-reliable, high-available, and low-cost network service has become an ever-growing concern.

Dependable monitoring and anomaly detection are effective measures of dependability assurance for network systems. A dependable monitoring mechanism acquires the performance metrics of each node. Based on the analysis of these metrics, the mechanism detects and identifies anomalies, which provides precise information for further network security countermeasures.

This paper proposes a static threshold-based and dynamic anomaly detection mechanism to achieve high dependable and low cost monitoring. Static threshold-based method is adopted to judge each group item of acquired secondary performance metrics. When any group item exceeds its threshold, dynamic detection methods are adopt to further detect anomaly. Experimental results show that the proposed anomaly detection mechanism achieves low false negative (FN) rate and low false positive (FP) rate, while keeping low overhead.

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 presents the structure of the proposed dependable monitoring mechanism. Section 4 addresses host and network data acquisition. Section 5 proposes the static threshold-based and dynamic anomaly detection mechanism. Section 6 conducts experiments and analyzes experimental results. Section 7 gives conclusions and looks into future work.

2. Related Work

This section summarizes research work related to dependable monitoring, and anomaly detection methods.

2.1. Dependable Monitoring

Dependability is an evolving concept. At a report in 2002, professor Avizienis summarized the origin and integration of the fundamental concepts related to dependability [1]. In the first paper of the initial issue of *IEEE Transactions on Dependable and Secure Computing*, Avizienis gave the main definitions relating to dependability [2]. Dependability is the system property that integrates such attributes as reliability, availability, safety, integrity, maintainability, *etc*.

Currently, a variety of distributed systems form many critical infrastructures strongly linked to people's daily life. Dependability outages are becoming increasingly common. In many situations, it is necessary not only to detect an anomaly but also to diagnose the anomaly, that is, to identify the source of the anomaly. Nevertheless, the premise of anomaly detection is dependable monitoring.

To address the intractable detection problem for large networks, Narasimha *et al.*, [3] propose to set a probe at each node in network. The probes use a variational inference technique to monitor the running status of nodes, and display the status by graphical modes.

For large-scale distributed systems, Khanna *et al.*, [4] propose an autonomous self-checking monitoring mechanism, which is used to provide fast detection to underlying network protocols. In [5], Khanna *et al.*, implement automated rule-based diagnosis through a distributed monitoring mechanism.

Despite the efforts in existing research work, a high dependable and low cost monitoring mechanism is still challenging, which is the main concern of this paper.

2.2. Anomaly detection

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior [6]. These nonconforming patterns are often referred to as anomalies, outliers, exceptions, *etc*.

Chandola *et al.*, [6] provides a structured and comprehensive overview of the research on anomaly detection. They group existing techniques into six different categories based on the underlying approach adopted by each technique: classification based, clustering based, nearest neighbor based, statistical, information theoretic, spectral, *etc*. For each category, they identify the advantages and disadvantages of the techniques in that category. They also provide a discussion on the computational complexity of the techniques.

Chandra and Toueg [7] characterize unreliable failure detectors in terms of two properties, *completeness* and *accuracy*. Completeness refers to the capability of detecting all failed components. Accuracy refers to the capability of avoiding detection mistakes.

Guan and Fu [8] propose an autonomic mechanism with little human intervention, auto-AID, for anomaly identification in networked computer systems. auto-AID is composed of a set of data mining techniques that facilitates automatic analysis of system health data. The identification results are very valuable for system administrators to manage systems and schedule the available resources.

Chang *et al.*, [9] design a random dissemination detection protocol based on flexible grouping. It takes the advantage of the reliability of Gossip-style dissemination, and reduces the network overhead and time consumption as every group is comparatively autonomous.

Principal component analysis (PCA), invented by Karl Pearson [10], is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

Pechenizkiy *et al.*, [11] analyze different feature extraction (FE) techniques as means of dimensionality reduction. They analyze traditional PCA and two eigenvector-based approaches. They conduct experiments on ten UCI data sets, using four different strategies to select samples.

With the introduction of any new technology, there is always a need for developing techniques to assess health status of systems wherein this new technology functions. Anomaly detection is an effective measure to monitor these systems for preventing any anomalies. Due to ever-increasing cloud sizes, as well as the complexity of system components, data collected by health monitoring tools is overwhelming. High metric dimensionality and existence of interacting metrics not only lead to high detection complexity, but compromise the detection accuracy. Guan *et al.*, [12] present a metric selection framework and systematic approaches to effectively identify and select the most essential metrics for online anomaly detection in utility clouds.

3. Structure of the Dependable Monitoring Mechanism

This paper designs a dependable monitoring mechanism applicable for WAN environment. The operating environment of this mechanism is shown in Figure 1.

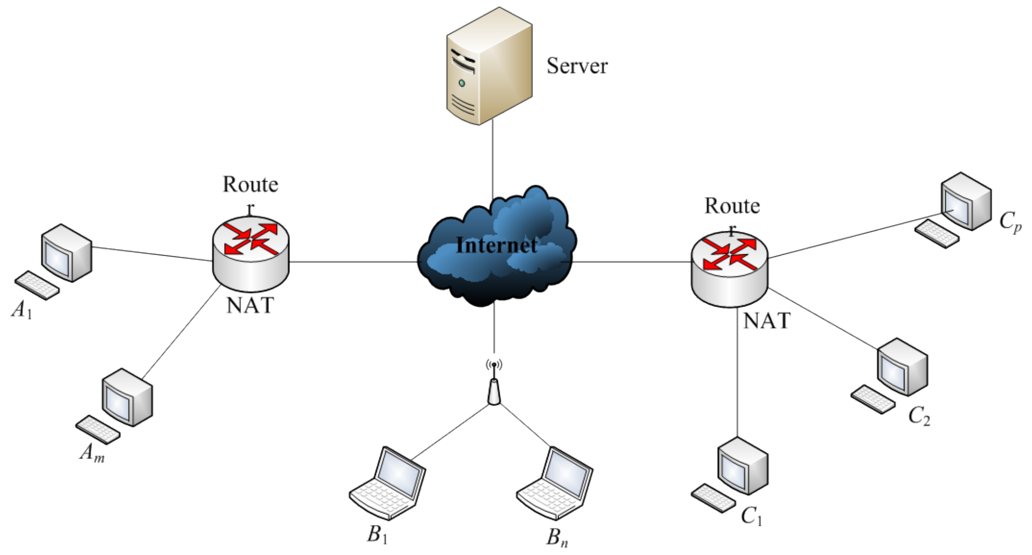


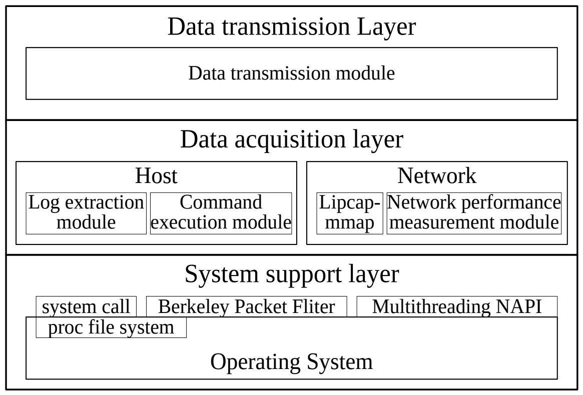
Figure 1. The Operating Environment of the Dependable Monitoring Mechanism

A server is selected for analyzing performance metrics and detecting anomalies. The dependability of the server itself is uncertain. Some key nodes are selected through self-organizing neighborhood algorithms. These key nodes are also servers. Each key node is responsible for providing services for terminals that connect to it, and sending its own operational status data to the server node. The underlying reason of this monitoring mechanism is that end users access resources from the server, but the server is unidirectional transparent to end users under existing network structure. End users do not know the server's operating status. Even when the server fails or behave abnormally, end users still can request for services, despite these request no longer receiving any response.

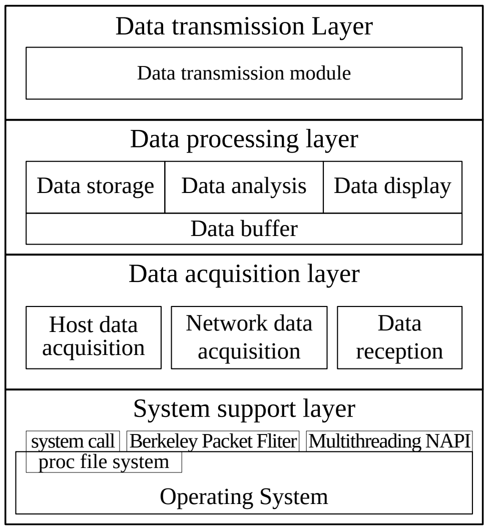
Since the key nodes and the server node may not belong to a same LAN, static NAT technology is adopted to transmit data between nodes. End to end communication is achieved through a NAT translation table in Routers. In the NAT translation table, the address of each host in the internal network is mapped to a legal address in the external network.

The software structure of key nodes, shown in Figure 2(a), includes three layers, which are *system support layer*, *data acquisition layer*, and *data transmission layer*. Multithreading NAPI in system support layer adopts the mutex operation of a circular queue to improve efficiency of NIC (Network Interface Card) interrupt, which provides a more efficient and reliable guarantee for network data acquisition. Data acquisition layer addresses host data and network data acquisition, which is detailed in section 4. Data transmission layer transmits acquired data to the server node.

The software structure of the server node is shown in Figure 2(b), which includes four layers. The system support layer in the server node is same as that in key nodes. Except for host data and network data acquisition, the data acquisition layer includes data reception (from key nodes). The data processing layer includes data storage, data analysis, and data display, *etc*. The data buffer in data processing layer is used to store data acquired from the sever node itself and received from key nodes. Anomaly detection results are transmitted to key nodes and normal nodes through data transmission module in the upper layer.



(a) The software structure of key nodes



(b) The software structure of the server node

Figure 2. The Software Structure of Key Nodes and the Server Node

4. Data Acquisition

The performance metrics of a monitored network are the foundation of dependable monitoring, since these metrics are the direct data sources of anomaly detection. This section addresses acquisition of host and network performance metrics.

4.1. Host Data Acquisition

Hoglund *et al.*, [13] elaborate the impact of system anomalies (due to either virus or illegal invasion) on host performance, and how the performance metrics (such as CPU utilization rate, memory utilization, number of processes currently running on the host, *etc.*) reflect host's abnormal behavior. From the literature, it is known that anomalies in a network system are often reflected in abnormal consumption of various host resource (such as CPU, physical memory, network bandwidth, I/O, *etc.*), as well as illegal modifications to the operating system. Therefore, capturing running status of hosts plays an important role in monitoring a network system.

Existing host anomaly monitoring methods are mainly divided into kernel monitoring and non-kernel monitoring. Kernel monitoring discovers system anomalies from the kernel level, such as analyzing system call sequence [14], analyzing running status of OS kernel [15], *etc*. Non-kernel monitoring judges whether the system is anomalous by analyzing OS operation logs [16], read and write operations of the file system, as well as status of network connection, *etc*.

This paper adopts non-kernel monitoring. The acquired host performance metrics include four primary metrics: CPU, memory, process, disk. Each primary metrics includes several secondary metrics, which amount to 30 secondary metrics, as shown in Table 1.

These secondary metrics are acquired by two methods: reading from relevant files in the /proc file system, or executing system monitoring commands of Linux via pipe. The acquisition method of each secondary metric is determined by the difficulty level of acquiring this metric. The first method is applicable when a metric is given in the /proc file system directly and can be extracted easily. Otherwise, the second method is adopted. The acquisition method of each secondary metric is listed in last column in Table 1.

Table 1. The Host Performance Metrics

Primary metrics	Secondary metrics	Description	File or command
CPU	cpu_user	Percentage of CPU utilization that occurred while executing at the user level.	/proc/stat
	cpu_nice	Percentage of CPU utilization that occurred while executing at the user level with nice priority.	/proc/stat
	cpu_sys	Percentage of CPU utilization that occurred while executing at the system level.	/proc/stat
	cpu_idle	Percentage of CPU idle time without outstanding disk I/O requests.	/proc/stat
	cpu_iowait	Percentage of time that the CPU or CPUs were idle during which the system had an outstanding disk I/O request.	/proc/stat
	cpu_irq	Percentage of time spent by the CPU or CPUs to service hardware interrupts.	/proc/stat
	cpu_softirq	Percentage of time spent by the CPU or CPUs to service software interrupts.	/proc/stat
	cpu_running	Number of running tasks in queues.	vmstat
	cpu_intr	Number of triggered interrupts.	vmstat
	cpu_contxt	Total number of CPU context switches per second.	vmstat
Memory	mem_insw	Amount of read in swap pages of physical memory.	vmstat
	mem_outsw	Amount of swap pages written into disk and deleted from memory.	vmstat
	mem_total	Total amount of physical memory.	/proc/mem

			info
	mem_free	Amount of free memory.	/proc/meminfo
	mem_active	Amount of active memory.	/proc/meminfo
Process	pro_mem_total	Total amount of memory occupied by processes.	top
	pro_mem_share	Amount of share memory used by processes.	top
	pro_cpu_total	Total CPU time occupied by processes.	top
	pro_cpu_perc	Percentage of CPU time occupied by processes.	top
	pro_mem_perc	Percentage of memory occupied by processes.	top
Disk	rrqmps	The number of read requests merged per second that were queued to the device.	/proc/disk
	wrqmps	The number of write requests merged per second that were queued to the device.	/proc/disk
	rps	Number of operations of reading I/O devices per second.	/proc/disk
	wps	Number of operations of writing I/O devices per second.	/proc/disk
	rKBps	Bytes (KBs) read per second.	/proc/disk
	wKBps	Bytes (KBs) written per second.	/proc/disk
	avgqu_sz	The average length of I/O queues.	/proc/disk
	Await	The average wait time (in milliseconds) for I/O requests issued to the device to be served.	/proc/disk
	Svctm	The average service time (in milliseconds) for I/O requests issued to the device to be served.	/proc/disk
	percUtil	Percentage of I/O time per second.	iostat -k -i

4.2. Network Data Acquisition

Network traffic data acquisition methods are divided into two categories: traffic-based [17], and packet-based [18]. The former method analyzes traffic information generated in network equipments. The traffic contains IP address, port, protocol and other metrics in communication. Current state of the network is determined through analyzing these metrics. The latter method determines the running status of network through analyzing the packets captured from the NIC.

Since packet-based data acquisition methods have several merits, such as accessing original data packets, low delay, not relying on third-party network equipment, not increasing burden on network, *etc.*, [18]. This article adopts packet-based methods to acquire network metrics. The acquired network performance metrics are listed in Table 2.

Table 2. The Network Performance Metrics

Primary metrics	**Secondary metrics**	**Description**
Packet loss	PktCntRcvLostPerSec	Number of lost packets in receiving per second.
	PktCntSndLostPerSec	Number of lost packets in transmitting per second.
ICMP	ICMPRcvPerSec	Number of received ICMP packets per second.
	ICMPRcvUnreachPerSec	Number of unreachable packets in received ICMP

		packets per second.
	ICMPSndPerSec	Number of transmitted ICMP packets per second.
	ICMPSndUnreachPerSec	Number of unreachable packets in transmitted ICMP packets per second.
	FlowCntRcvPerSec	Network data received per second.
Port traffic	FlowCntRcvPerSec	Network data transmitted per second.
	PortCnt	Port number of connections.
Network load rate	NetworkLoadRat	Network load rate.

5. Static and Dynamic Anomaly Detection

In order to improve availability and reduce false positive rate of a monitoring mechanism, it should collect as many fine-grained performance metrics as possible. However, high metric dimensionality and high volume consume much computing resource during anomaly detection, which delays anomaly identification. In order to achieve dependable and low-cost monitoring, it is necessary to make a trade-off between availability and high efficiency of a monitoring mechanism. Therefore, this paper proposes a static threshold-based and dynamic anomaly detection mechanism, as shown in Figure 3, to address this contradiction. This mechanism works as follows: in each round, it first adopts a threshold-based method to judge each group item of the secondary metrics; when any group item exceeds its threshold, dynamic detection methods are adopted to further detect anomaly.

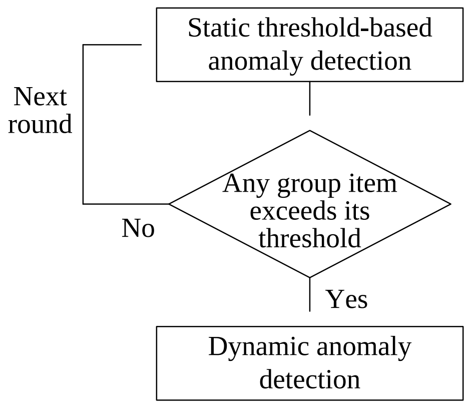

Static threshold-based anomaly detection
Next round
Any group item exceeds its threshold
No
Yes
Dynamic anomaly detection

Figure 3. Static Threshold-based and Dynamic Anomaly Detection

The advantages of static & dynamic anomaly detection are summarized as follows.

(1) The simple static threshold-based anomaly detection with lower computation cost achieves real-time monitoring, which makes the monitoring mechanism is extremely sensitive to anomalies;

(2) The introduction of static method reduces invoking frequency of dynamic anomaly detection. Only when any group item exceeds its threshold, the dynamic anomaly detection is invoked to further detect anomaly. Therefore, the monitoring mechanism does not necessarily work at the dynamic monitoring mode at any moment, which reduces the burden of the server node.

5.1. Static Anomaly Detection

Since a single performance metric does not precisely represent a system anomaly, this paper combines 30 host secondary performance metrics into several group items to represent corresponding anomaly.

Firstly, 12 intermediate items are gained based on these 30 secondary metrics, as shown in Table 3. For example, the intermediate item, cpu_total, consists of 7 secondary metrics. This intermediate item is the sum of these 7 secondary metrics. The gained value represents the total CPU time.

Table 3. The Intermediate Items of Secondary Metrics

Intermediate items	Secondary metrics	Computing method
Cpu_total	{ cpu_user, cpu_nice, cpu_sys, cpu_idle, cpu_iowait, cpu_irq, cpu_softirq }	cpu_total = cpu_user + cpu_nice + cpu_sys + cpu_idle + cpu_iowait + cpu_irq + cpu_softirq
Cpu_used_perc	{ cpu_total, cpu_idle }	cpu_used_perc = (cpu_total - cpu_idle) / cpu_total
Cpu_user_perc	{cpu_user , cpu_total }	cpu_user_perc = cpu_user / cpu_total
Cpu_sys_perc	{ cpu_sys, cpu_total }	cpu_sys_perc = cpu_sys / cpu_total
Cpu_iowait_perc	{ cpu_total, cpu_iowait }	cpu_iowait_perc = cpu_iowait / cpu_total
Cpu_idle_perc	{ cpu_total, cpu_idle }	cpu_idle_perc = cpu_idle / cpu_total
Cpu_intr_perc	{ cpu_total, cpu_intr }	cpu_intr_perc = cpu_intr / cpu_total
Cpu_contxt_perc	{ cpu_total, cpu_contxt }	cpu_contxt_perc = cpu_contxt / cpu_total
mem_swapin	{ mem_insw }	mem_swapin = mem_insw
mem_swapout	{ mem_outsw }	mem_swapout = mem_outsw
mem_used_perc	{ mem_total, mem_free }	mem_used_perc = (mem_total - mem_free) / mem_total
mem_active_perc	{ mem_total, mem_active }	mem_active_perc = (mem_total - mem_active) / mem_total

Further, 8 group items are gained based on the above 12 intermediate items and aforementioned 30 secondary metrics, as shown in Table 4. For example, the group item, S_cpu_userSys, is the sum of two intermediate items, *i.e.*, cpu_user_perc and cpu_sys_perc.

Table 4. Group Items

Group items	Intermediate items	Computing method
S_cpu_userSys	{ cpu_user_perc, cpu_sys_perc }	S_cpu_userSys = cpu_user_perc + cpu_sys_perc
S_cpu_iowait	{ cpu_iowait_perc }	S_cpu_iowait = cpu_iowait_perc
S_cpu_idleRun	{ cpu_idle_perc, cpu_running }	
S_cpu_intrUsed	{ cpu_intr_perc, cpu_used_perc }	
S_cpu_contxtSys	{ cpu_contxt_perc, cpu_sys_perc }	
S_mem_swapinOut	{ mem_swapin, mem_swapout }	
S_mem_used	{ mem_used_perc }	S_mem_used = mem_used_perc
S_mem_leak	{ mem_used_perc, mem_active_perc }	

Lastly, the computing methods of the above 8 group items are shown in Figure 4. Thresholds are established in two modalities: *interval of a group item*, and *set of intervals of intermediate items*. The former modality is detailed as: a group item is calculated by several intermediate items and a definite value is gained; the given threshold is an interval of this definite value. The latter modality is detailed as: a group item consists of several intermediate items; but no calculation proceeds on these intermediate items; the given thresholds are set of intervals of each intermediate item.

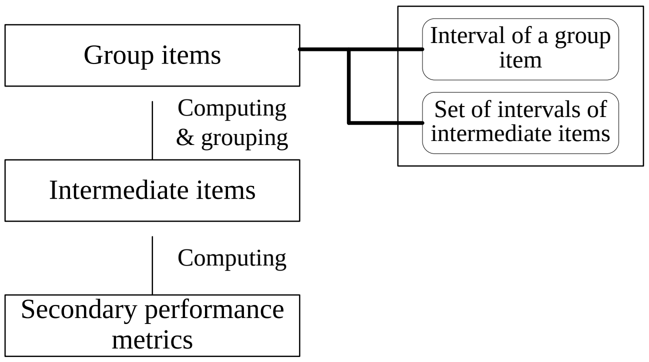


Figure 4. The Computing Methods of Group Items

The thresholds modalities of above 8 group items are shown in Table 5. For any group item, the value beyond its threshold represents a certain anomaly category, which detailed in Table 5.

Table 5. The Thresholds of Group Items

Group items	Thresholds	Anomaly category
S_cpu_userSys	(0.6, 0.85)	Response time is increasing, throughput is descending.
S_cpu_iowait	(0, 0.25)	Disk is overload.
S_cpu_idleRun	(0, 4, 1) && (0, 4)	CPU load is heavy. The number of tasks is excessive.
S_cpu_intrUsed	(0, 0.15) && (0, 0.85)	CPU is overload. Interrupt is frequent.
S_cpu_contxtSys	(0, 5000) && (0, 0.4)	High frequency of context switch.
S_mem_swapinOut	0 && 0	Shortage of memory.
S_mem_used	(0, 0.85)	Shortage of memory.
S_mem_leak	(0, 0.8) && (0, 0.8)	Memory leakage.

The consequent issue is how to determine the results of anomaly detection. Three possible strategies are listed as follows:

(a) If all the of the group items exceed their respective threshold, a system anomaly is determined.

(b) If a certain percentage of the group items exceed their respective threshold, a system anomaly is determined.

(c) If any group item exceeds its respective threshold, a system anomaly is determined.

Due to sensitivity of group items, the first strategy is unreasonable. The percentage in second strategy is hard to determine, which causes this strategy infeasible. Therefore, only the third strategy is feasible and reasonable. Therefore, this paper adopts the third strategy during static threshold-based anomaly detection phase.

5.2. Dynamic Anomaly Detection

The dynamic anomaly detection method (shown in Figure 5) is detailed as follows.

(1) Firstly, PCA is adopted to reduce dimensionality; accordingly, the principal components are selected;

(2) Through observation and experiments, it is found that the aforementioned secondary performance metrics are independent. Moreover, their distributions accord with joint Gaussian distribution model, which helps to determine apriori probabilities in Bayesian classification.

(3) Lastly, for currently acquired performance metrics of the monitored system, Bayesian decision is adopted to classify current status as normal or abnormal.

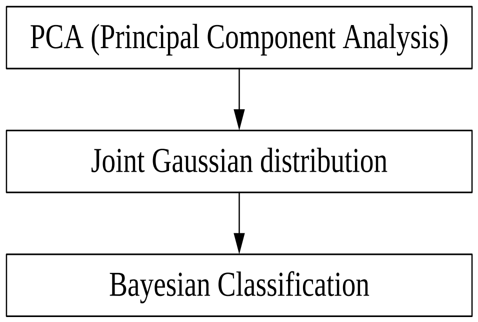

PCA (Principal Component Analysis)
Joint Gaussian distribution
Bayesian Classification

Figure 5. Dynamic Anomaly Detection

(1) PCA (Principal Component Analysis)

Assume that, the acquired data constitute a matrix X_{M*N}; M is the number of sampling data; N is the number of performance metrics in each data.

The data matrix is denoted by: $X_{M*N} = (X_1, X_2, ..., X_i, ..., X_M)^T$.

Each data is denoted by: $X_i = (x_{i1}, x_{i2}, ..., x_{ij}, ..., x_{iN})$.

From the performance metrics point of view, the data matrix X_{M*N} includes N performance metrics: $X_{M*N} = (P_1, P_2, ..., P_j, ..., P_N)$.

Each performance metric is denoted by: $P_j = (x_{1j}, x_{2j}, ..., x_{ij}, ..., x_{Mj})^T$.

In PCA, the processes of calculating and selecting principal components are listed as follows.

1) It computes the mean value, E_j, of each performance metric, P_j.

$$E_j = \frac{\sum_{i=1}^{M} x_{ij}}{M}, \quad j = 1, 2, ..., N \tag{1}$$

2) It obtains the matrix X_e by subtracting the mean value, E_j, from the value of each performance metric in each sampling data, x_{ij}.

$$X_e = X - E \tag{2}$$

3) It computes the covariance matrix C of matrix X_e.

$$C = \begin{pmatrix} C_{11} & \cdots & C_{1N} \\ \cdots & \cdots & \cdots \\ C_{N1} & \cdots & \cdots C_{NN} \end{pmatrix} \quad (3)$$

C_{ij} is the covariance between performance metrics P_i and P_j, which is computed as:

$$C_{ij} = cov(P_i, P_j) = \frac{\sum_{k=1}^{M}[(x_{ki} - E_i)(x_{kj} - E_j)]}{M-1} \quad (4)$$

4) It computes the eigenvalues and eigenvectors of the covariance matrix C. The obtained eignevalues are denoted by $\lambda_1, \lambda_2, ..., \lambda_n$ (in descending order), $n \le N$. The corresponding eigenvectors are denoted by $a_1, a_2, ..., a_n$.

5) This paper proposes a strategy to select the principal components by adopting the correlation distance between components.

Assume that, the distance between two values P and Q is defined as:

$$\partial(P, Q) = \frac{P - Q}{Q} \quad (5)$$

The correlation distance between three values P, Q and R is defined as:

$$\mu(P, Q, R) = \partial(P, Q) - \partial(Q, R) \quad (6)$$

It computes the correlation distances among obtained eignevalues $\lambda_1, \lambda_2, ..., \lambda_n$. It selects an eignevalue, λ_j, as a principal component if three adjacent eignevalues, $\lambda_i, \lambda_j, \lambda_k$, satisfy: $\mu(\lambda_i, \lambda_j, \lambda_k) < 0$.

According to this strategy, it selects p $(p \le n)$ largest eigenvalues. The corresponding eigenvectors form the principal components, $Q = (b_1, b_2, ..., b_p)$.

(2) Joint Gaussian distribution

This paper randomly chooses several performance metrics, and observes the distributions of their values. The results show that the values of each performance metric accord with Gaussian distribution. Since these N performance metrics are independent, they accord with joint Gaussian distribution. The probability density function is determined by the following equation.

$$p(x) = p(x_1, x_2, ..., x_N) = \frac{1}{(2\pi)^{N/2} |K|^{1/2}} \exp\left[\frac{-(x-M)^T K^{-1}(x-M)}{2}\right] \quad (7)$$

In the above equation, M is mean vector, and K is covariance matrix.

According to the obtained principal components, $Q = (b_1, b_2, ..., b_p)$, the values of M and K are calculated as follows.

$$M = E[X] = E[b_1, b_2, ..., b_p]^T = \frac{1}{p}\sum_{i=1}^{p} b_i \quad (8)$$

$$K = E[(X - M)(X - M)^T] \tag{9}$$

(3) Bayesian Classification

Assume that, there are only two pattern categories: C_n denotes the normal status of the monitored system, and C_a denotes the abnormal status of the system. C represents any pattern category.

The currently acquired performance metric vector is denoted by x. $P(C_n \mid x)$ and $P(C_a \mid x)$ denote the aposteriori conditional probabilities of normal status and abnormal status, respectively.

In Bayesian classification, it need know the preconditions as follows.

1) The conditional probability density function of each pattern category, which is determined as: C_n, $X\sim N(M_n, K_n)$; C_a, $X\sim N(M_a, K_a)$.

2) The prior probability of each pattern category.

The decision rules in Bayesian classification are listed as follows.

$$\begin{cases} P(C_n \mid x) > P(C_a \mid x), & \textit{It shows that the system is normal.} \\ P(C_n \mid x) < P(C_a \mid x), & \textit{It shows that the system is abnormal.} \\ P(C_n \mid x) = P(C_a \mid x), & \textit{The status of system is decided randomly.} \end{cases} \tag{10}$$

The aposterior probability is calculated as:

$$P(C \mid x) = \frac{P(x \mid C)P(C)}{P(x)} \tag{11}$$

Therefore, the equation (10) is transformed into:

$$\begin{cases} \dfrac{P(x \mid C_n)P(C_n)}{P(x)} > \dfrac{P(x \mid C_a)P(C_a)}{P(x)}, & \textit{It shows that the system is normal.} \\ \dfrac{P(x \mid C_n)P(C_n)}{P(x)} < \dfrac{P(x \mid C_a)P(C_a)}{P(x)}, & \textit{It shows that the system is abnormal.} \\ \dfrac{P(x \mid C_n)P(C_n)}{P(x)} = \dfrac{P(x \mid C_a)P(C_a)}{P(x)}, & \textit{The status of system is decided randomly.} \end{cases} \tag{12}$$

In the above equation, $P(x|C_n)$ represents the conditional probability when the system is normal, and $P(x|C_a)$ represents the conditional probability when the system is abnormal.

After simplification, the equation (12) is expressed as:

$$\begin{cases} \dfrac{P(x \mid C_n)}{P(x \mid C_a)} > \dfrac{P(C_a)}{P(C_n)}, & \textit{It shows that the system is normal.} \\ \dfrac{P(x \mid C_n)}{P(x \mid C_a)} < \dfrac{P(C_a)}{P(C_n)}, & \textit{It shows that the system is abnormal.} \\ \dfrac{P(x \mid C_n)}{P(x \mid C_a)} = \dfrac{P(C_a)}{P(C_n)}, & \textit{The status of system is decided randomly.} \end{cases} \tag{13}$$

The probability ratio, $P(x \mid C_n) / P(x \mid C_a)$, is represented as $l(x)$; and threshold value, $P(C_a) / P(C_n)$, is represented as η.

Therefore, the decision rules can be transformed into:

$$\begin{cases} l(x) > \eta, & \textit{It shows that the system is normal.} \\ l(x) < \eta, & \textit{It shows that the system is abnormal.} \\ l(x) = \eta, & \textit{The status of system is decided randomly.} \end{cases} \tag{13}$$

6. Experiments and Analysis

This section conducts experiments in a campus-wide network system. The proposed dependable monitoring mechanism is deployed in the network environment to test its performance and efficiency.

6.1. A Sampled Data Set and PCA Results

For the aforementioned 42 performance metrics, a sampled data set including 3580 groups data are acquired. A segment of this sampled data set is shown in Table 6.

Table 6. A Segment of the Sampled Data Set

	cpu_user	cpu_nice	cpu_sys	cpu_idle	cpu_iowait	...
1	4120	2318	3201	316	524	...
2	5203	2410	3410	205	536	...
3	6286	2502	3619	137	548	...
4	5421	2594	3828	124	560	...
5	4556	2686	3810	234	572	...
6	3691	2518	3792	344	581	...
7	4369	2350	3774	454	590	...
8	5047	2182	3756	514	599	...
9	5725	2014	3810	574	608	...
10	5784	1846	3864	634	624	...
...	...	...	...	...	...	...

The proposed mechanism adopts PCA to reduce dimensionality and choose principal components. As shown in Figure 6, ten principal components are chosen, which are: cpu_contxt, cpu_user, mem_outsw, cpu_nice, mem_active, mem_free, cpu_sys, PortCnt, IcmpSndPerSec, perUtil (in the descending order of eigenvalues).

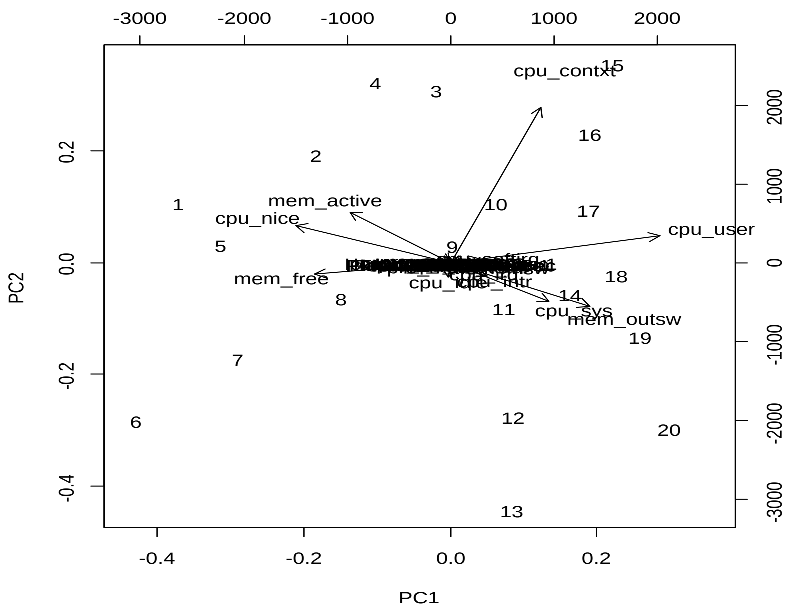


Figure 6. The Performance Metrics in PCA

6.2. Experimental Results and Analysis

This subsection adopts event injection (detailed as follows) to examine the presented dependable monitoring mechanism.

(1) *CPU consumption*: 25 processes execute simultaneously to observe the consumption of CPU.

(2) *Memory*: In order to simulate memory consumption, the *malloc* function is adopted to continuously require memory from the operating system.

(3) *Disk*: To simulate attacks towards disk, data are copied from USB flash disk to host machine.

(4) *Network*: To simulate network consumption, the stream.c program is executed in each node. Meanwhile, malicious links are generated.

The sampling interval has a strong impact on the overhead of the monitoring mechanism. There is no doubt that if the sampling interval is set at a much little value (*e.g.*, 5 second), the overhead is relatively high. Moreover, since system state change is a gradual process, the monitored network system undergoes minor changes after a short elapsed time, which may not be detected. This paper measures the overhead of the proposed monitoring system under different sampling intervals, as shown in Figure 7. When the sampling interval is set at 20 second, the overhead is only 3.6%.

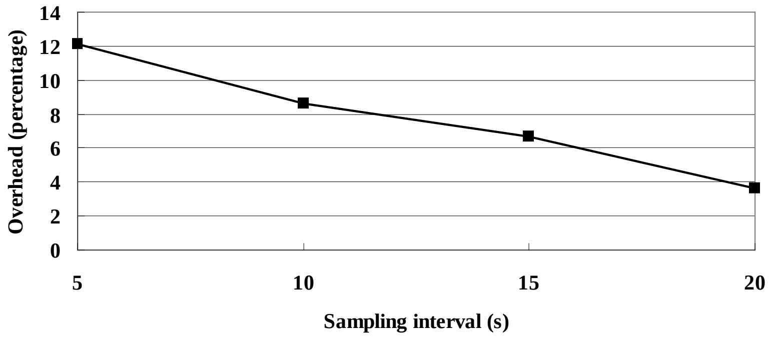


Figure 7. The Relationship between Overhead and Sampling Interval

This paper analyzes 100 samples by adopting four methods (the sampling interval is set at 20 second): static threshold-based, traditional principal component analysis (PCA) & Bayesian Classification, grouping detection [9], and the proposed static & dynamic anomaly detection method. This paper compares these four methods in false positive rate and false negative rate.

For an anomaly detection method, false negative (FN) rate refers to the rate of occurrence of anomaly that the method fails to detect when anomalies (due to either virus or illegal invasion) happen. The FN rates of these four algorithms are shown in Figure 8. From this Figure, it is found that the proposed static & dynamic method outperforms PCA & Bayesian and grouping detection. The static method achieves much lower FN rate. The underlying reason is that the thresholds are set as lower values, which results in high false positive rate (as shown in Figure 9).

False positive (FP) rate refers to the rate of reported anomaly occurrence when there is no anomaly. The false positive rates of these four algorithms are shown in Figure 9. From this Figure, it is found that static & dynamic method still outperforms PCA & Bayesian and grouping detection. However, the static method suffers much high FP rate due to lower threshold values.

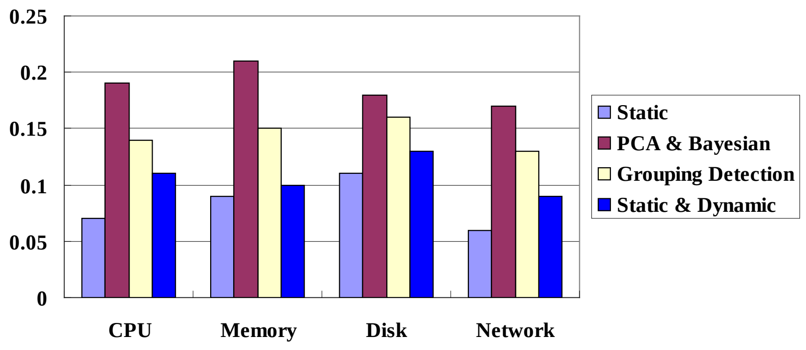


Figure 8. The False Negative Rates of Four Algorithms

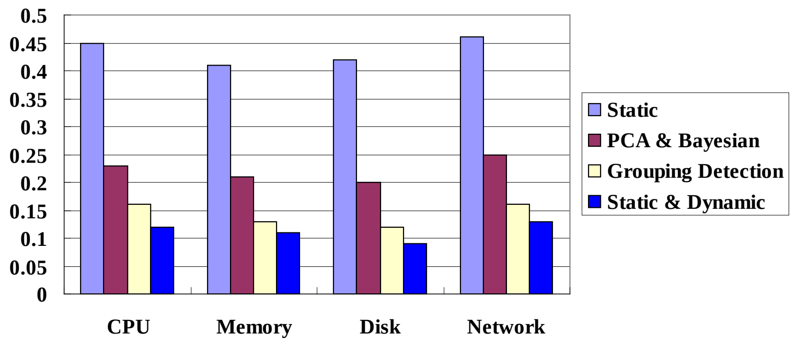


Figure 9. The False Positive Rates of Four Algorithms

7. Conclusion and Future Work

This paper proposes a static threshold-based and dynamic anomaly detection mechanism for network systems. This mechanism makes a trade-off between availability and high efficiency of a monitoring system. Therefore, it achieves high dependable and low cost monitoring. One defect of this mechanism is that the thresholds are experiential values. Nevertheless, the proposed mechanism implements precise determination of anomaly. This mechanism provides a strong support for further locating the detected anomalies.

The future work of this paper will be identifying and locating detected anomalies. In addition, this paper will also concern anomaly detection issue under a promising paradigm, *i.e.*, cloud computing.

Acknowledgments

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

The work of this paper is supported by National Natural Science Foundation of China (Grant No. 61272399) and Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20110191110038).

References

[1] J.-C. Laprie Avizienis and B. Randell, “Fundamental Concepts of Dependability”, Newcastle University Report, no. CS-TR-739, **(2002)** September, pp. 28.

[2] A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing”, IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, **(2004)**, pp. 11-33.

[3] R. Narasimha, S. Dihidar, C. Y. Ji and S. W. McLaughlin, “Scalable Fault Diagnosis in IP Networks using Graphics Models: A Variational Inference Approach”, Proceedings of IEEE International Conference on Communications (ICC), **(2007)**, pp. 147-152.

[4] G. Khanna, P. Varadharajan and S. Bagchi, “Automated online monitoring of distributed applications through external monitors”, IEEE Transactions on Dependable and Secure Computing, vol. 3, no. 2, **(2006)**, pp. 115-129.

[5] G. Khanna, M. Y. Cheng, P. Varadharajan and S. Bagchi, “Automated Rule-Based Diagnosis through a Distributed Monitor System”, IEEE Transactions on Dependable and Secure Computing, vol. 4, no. 4, **(2007)**, pp. 266-279.

[6] V. Chandola, A. Banerjee and V. Kumar, “Anomaly Detection: A Survey”, ACM Computing Surveys, vol. 41, no. 3, Article 15, **(2009)** July.

[7] T. D. Chandra and S. Toueg, “Unreliable failure detectors for reliable distributed systems”, Journal of the ACM, vol. 43, no. 2, **(1996)**, pp. 225-267.

[8] Q. Guan and S. Fu, “auto-AID: A Data Mining Framework for Autonomic Anomaly Identification in Networked Computer Systems”, IEEE International Performance Computing and Communications Conference (IPCCC), **(2010)**.

[9] G. H. Chang, H. W. Lu, S. Y. Chen and I. Shih, “Grouping Fault Detection Protocol under Dynamic Network Environments”, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), pp. 151-155, **(2010)**.

[10] K. Pearson, “On Lines and Planes of Closest Fit to Systems of Points in Space”, Philosophical Magazine, vol. 2, no. 11, **(1901)**, pp. 559-572.

[11] M. Pechenizkiy, S. Puuronen and A. Tsymbal, “The Impact of Sample Reduction on PCA-based Feature Extraction for Supervised Learning”, The 21st Annual ACM Symposium on Applied Computing (SAC), April **(2006)**, pp. 553-558.

[12] Q. Guan, C. C. Chiu, Z. M. Zhang and S. Fu, “Efficient and Accurate Anomaly Identification using Reduced Metric Space in Utility Clouds”, Proceedings of 7th International Conference on Networking, Architecture and Storage (NAS), **(2012)**, pp. 207-216.

[13] A. J. Hoglund, K. Hatonen and A. S. Sorvari, “A Computer Host-based User Anomaly Detection System Using the Self-organizing Map”, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN), **(2000)**, pp. 411-416.

[14] A. P. Kosoresow and S. A. Hofmeya, “Intrusion Detection via System Call Traces”, IEEE Software, vol. 14, no. 5, **(1997)**, pp. 35-42.

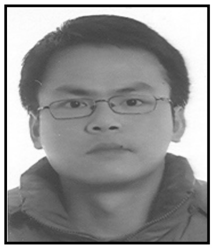
[15] O. S. Hofmann, A. M. Dunn, S. Kim, I. Roy and E. Witchel, “Ensuring Operating System Kernel Integrity with OSck”, Proceedings of 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), **(2011)**, pp. 279-290.

[16] W. Xu, L. Huang, A. Fox, D. Patterson and M. I. Jordan, “Detecting Large-Scale System Problems by Mining Console Logs”, Proceedings of 22nd ACM Symposium on Operating Systems Principles (SOSP), **(2009)**, pp. 117-131.

[17] P. Barford and D. Plonka, “Characteristics of network traffic flow anomalies”, Proceedings of ACM SIGCOMM Internet Measurement Workshop, **(2001)** November.

[18] K. Julisch, “Clustering Intrusion Detection Alarms to Support Root Cause Analysis”, Transactions on Information and System Security (TISSEC), vol. 6, no. 4, **(2003)**, pp. 443-471.

Authors



GuiPing Wang, he received his B.S. degree and M.S. degree in Chongqing University, P. R. China, at 2000 and 2003 respectively. Since July 2003, he acts as a lecturer in Information School, Zhejiang University of Finance and Economy. Currently he is a full-time Ph.D. candidate in College of Computer Science, Chongqing University. His research interests include dependability analysis and fault diagnosis of distributed systems, cloud computing, etc. As the first author, he has published nearly 20 papers in related research areas during recent years at journals such as Expert Systems with Applications, Information Processing Letters, WSEAS Transactions on Computers, etc.

ShuYu Chen, he received his Ph.D. degree in Chongqing University, P. R. China, at 2001. Currently, he is a professor of College of Software Engineering at Chongqing University. His research interests include embedded Linux system, distributed systems, cloud computing, etc. He has published over 120 journal and conference papers in related research areas during recent years.

Zhen Zhou, received his B.S. degree in Chongqing University of Technology, P. R. China, at 2006, and received his M.S. degree in Chongqing University, P. R. China, at 2010. He is currently a full-time Ph.D. candidate in Chongqing University. His current interests include virtualization technology, cloud computing, etc.

MingWei Lin, Lin received his B.S. degree in Chongqing University, P. R. China, at 2009. He is currently a full-time Ph.D. candidate in Chongqing University. He is invited as the reviewer by Journal of Systems and Software, as well as Computers and Electrical Engineering. His current interests include flash memory, Linux Kernel and wireless sensor network.