Trustworthy Enhancement for Cloud Proxy based on Autonomic Computing

Hui He, *Member, IEEE*, Weizhe Zhang, *Member, IEEE*, Chuanyi Liu, and Honglei Sun

Abstract—Aiming to improve Internet content accessing capacity of the system, cloud proxy platforms are used to improve the visiting performance in network export environment. Limited by complexity of cloud proxy system, trustworthy guarantee of cloud system becomes a difficult problem. Considering the self-government of autonomic computing, it could enhance cloud system trustworthy and avoids system management security and reliable problems brought by complex construction. Based on the idea of self-supervisory, a mechanism to enhance security of cloud system was proposed in this paper. First, a trustworthy autonomous enhancement framework for virtual machines was proposed. Second, a method to extract linear relationship of monitoring items in the virtual machine based on ARX model was put forward. According to the mapping relation between monitoring items and system modules, an abnormal module positioning technology based on Naive Bayes classifier was developed to realize self-sensing of abnormal system conditions. Finally, security threats of virtual machines including malicious dialogue and buffer memory of hot attacks were tested through experiments. Results showed that the proposed trustworthy enhancement mechanism of virtual machines based on autonomic computing could achieve trustworthy enhancement of virtual machines effectively and provide an effective safety protection for the cloud system.

Index Terms—Autonomic computing, virtual machine introspection, trustworthy, self-sensing, independent decision-making

1 Introduction

In the past six decades, information technology (IT) has been progressing continuously. It developed from large computer to minicomputer and then to microcomputer and from personal computer (PC) to ascendant mobile internet. Information industry has changed our lives fundamentally. Enterprises of different industries offer advanced services through large distributed systems, including Cloud system, social networking services (SNS), e-commerce and even banking business. However, these advanced services are accompanied with constant trustworthy threats of large distributed systems. The trusted cloud computing, which combination of trusted computing technology is a major research direction to resolve the data security issues in cloud computing. It represents reliability and security. Such trustworthy threats include hardware failure, software breakdown, malicious attack, mistaken operation, and so on. Computer systems used in key fields like national defense, finance and electricity will cause abnormalities of distributed system once they encountered with trustworthy threats, which results in huge losses to national benefit and people's lives and properties. Hence, people put forward higher and higher requirements on trustworthy of distributed computer system. They hope computer system could recover from abnormality in a short time and even hope it could provide uninterrupted services. This claims higher trustworthy guarantee capability of distributed system. Management, maintenance and repair of distributed information system are important means of trustworthy guarantee. Although the changing IT increases productivity of enterprises significantly, it is accompanied with challenges on trustworthy guarantee of the distributed information systems. These challenges are mainly caused by the increasing complexity of distributed systems. On one hand, increasing complexity of cloud systems will increase cost for system maintenance. It is surveyed that the average maintenance expenditures of IT companies for existing systems account for as high as 80 percent of the financial budgets. On the other hand, system maintenance depends more and more on system management. Therefore, traditional trustworthy enhancement method for distributed system which is at the cost of finance and manpower cannot meet current demands. To adapt to the current and even future complicating and expanding distributed system environments (e.g., cloud platform and intelligent engineering under the background of big data analysis) better, it is press for a new intelligent and autonomous mechanism to protect and enhance trustworthy of cloud systems.

IBM Company developed an autonomic computing in 2001 [1]. Based on the character of "technology managing technology", autonomic computing provides a new way to safeguard and enhance trustworthy of distributed systems. Autonomous trustworthy enhancement of distributed systems was achieved by combining autonomic computing (tool) and trustworthy (goal). It is a powerful mean to solve poor autonomous trustworthy enhancement caused by system complexity.

2 Research Reviews

Autonomic computing is inspired by autonomic nervous system of human bodies. It aims to perceive the changing

- *H. He and W.-Z Zhang are with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, Heilongjiang 150001, China. E-mail: {hehui, wzzhang}@hit.edu.cn.*
- *C.-Y. Liu is with the School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, Guangdong 150001, China. E-mail: cy-liu04@mails.tsinghua.edu.cn.*
- *H.-L. Sun is with the Network information center, Harbin Institute of Technology, Harbin, Heilongjiang 150001, China. E-mail: sunhonglei@hit.edu.cn.*

Manuscript received 14 Feb. 2016; revised 30 June 2016; accepted 31 July 2016. Date of publication 19 Oct. 2016; date of current version 3 Dec. 2020. Recommended for acceptance by M. Parashar, O. Rana, and R.C.H. Hsu. Digital Object Identifier no. 10.1109/TCC.2016.2603508

system environment and its operating environment autonomously and adjust states automatically according to system running conditions. The whole process involves no artificial participation but could realize autonomic repair, optimization and management [2]. Existing researches on autonomic computing mainly focus on (1) self-management skills, universality and development tools of autonomic elements; (2) structure of autonomic computing system, positioning and control of large-scale autonomic computing system; (3) autonomic computing strategy; (4) combination of autonomic computing, grid computing and virtualization technology. With the continuous development of autonomic computing, attributes of autonomic computing have been defined gradually. Specifically, autonomic computing has four attributes, namely, self-protecting, self-configuring, self-healing and self-optimizing. These are the main attributes of autonomic computing. Autonomic computing has another four important attributes: self-awareness, environment-awareness, self-monitoring and self-adjusting [3].

After IBM Company proposed the concept of autonomic computing in 2001, some famous IT companies also proposed their own autonomic computing concepts, such as "N1" concept of SUN Company, Sun Microsystems' strategy for making a network environment as easy to manage as a single machine and the adaptive enterprise of HP Company [4]. Besides, Microsoft Company proposed the concept of Dynamic Systems Initiative and the definition model of system [5]. The core idea is also the self-managing dynamic systems. Intel Company put forward the concept of Proactive Computing [6]. NEC Company put forward the concept of "VALUMO" [7]. Many famous IT enterprises like Toshiba and Fujitsu put forward their own concepts of autonomic computing. The "Recovery-Oriented Computing" project of University of California Berkeley and Stanford University focuses on transferring fault tolerance into self-repairing of system abnormalities.

Automatic computing was studied early in foreign countries. Daniel et al. [8] used the autonomic computing in dynamic distribution of service resources and established a utility function by measuring system indexes (e.g., response time). When the server workload in system changes or the server fails, the global utility function could be maximized through self-adjusting of dynamic service resources distribution. Chen et al. [9] suggested to using the decision tree to realize system self-diagnosis. Feasibility of this self-diagnosis program was verified on the eBay platform. Based on the cluster idea of data mining technology, Kiciman et al. [10] inferred an appropriate system configuration program autonomously from existing configuration samples. Littman et al. [11] modeled the network repair by using reinforcement learning technology, thus realizing the goal of network self-healing and network connectivity protection. Ranganathan et al. [12] emphasized on self-optimizing in autonomic computing and established a multi-dimensional utility function, achieving the optimal way for autonomous selection when executing tasks in complicate distributed system environment. Bohra et al. [13] introduced a method for remote control and repairing of software by using the operating system backdoor and implemented the prototype system by modifying the FreeBSD core. Fork bomb and memory hog were tested through an experiment, which verified that the proposed method is feasible. Jonathan Wildstrom et al. [14] proposed a hardware self-configuring mechanism for TPC-W evaluation distributed system. Different from the traditional autonomic computing idea, this mechanism doesn't employ any middleware, but makes machine learning on service condition and current configuration of system hardware. Therefore, it is able to find the optimum hardware configuration under different operating states. Rutherford et al. [15] proposed a self-configuring mechanism against container model of enterprise JavaBean modules. Autonomic computing also attracts attentions from decision-making researchers. Srivastava et al. [16] summarized problems and challenges that may encounter when using independent decision computing in autonomic computing. Kephart [17] listed current technical and engineering challenges in autonomic computing, such as problem about interface uniformity in autonomic manager design. He also described existing research achievements of some problems. Brown et al. [18] proposed an evaluation system for autonomic computing. Compared to traditional evaluation system, it applied changes based on workload-response pattern and provided a new idea to evaluate capability of autonomic computing. At present, research on autonomic computing is still in the stage of theoretical exploration and simulation experiment. Only few practical research results are available.

As for the cloud platform, Michele et al. [19] introduced autonomic computing techniques into IaaS platform, which can provide better management of energy consumption, quality of service (QoS), and unpredictable system behaviors. Mona et al. [20] proposed Requirements Engineering framework for autonomic systems. Eleni et al. [21] introduces a novel cloud computing network architecture that allows for virtual machine replication and merging along with migration to reduce the overall cost for using a service. Nizar et al. [22] presented a self-configuration algorithm for Organic Computing systems, which aims to equally distribute the load of services on nodes and to assign services with different importance levels to nodes so that the more important services are assigned to more trustworthy nodes. Roberto et al. [23] provide a solution for a generic class of problems that distribute a parallel computation over a set of nodes where trustworthiness of the outsourced computation is important. More important works are presented about automatic memory control [27] and network-aware virtual machine migration [28].

3 Establishing an Autonomous Trustworthy Enhancement Frame for Cloud System

Based on researches on trustworthy enhancement means of autonomic computing and network proxy technology, this paper proposed an autonomous trustworthy enhancement framework based on autonomic element model for cloud system through an autonomic manager model. Framework design is shown in Fig. 1. It is composed of state monitoring, state self-sensing, abnormality positioning, independent decision-making and strategy implementation. With these five parts, it realizes the self-feedback trustworthy enhancement control loop of virtual machine (VM) from state recognition→abnormality positioning→strategy making→strategy implementation. Information flow in this loop is shown in Fig. 1.

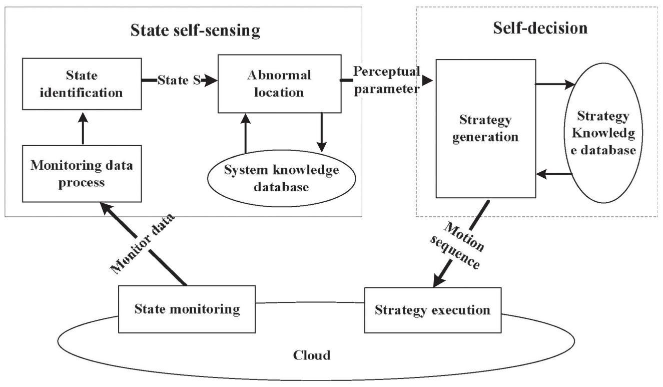


Fig. 1. Trustworthy enhancement framework of cloud system.

Functions of four components of the framework are introduced as follows:

(1) State monitoring. State probes are set on VM nodes, which are responsible for collecting system operation data through provided interfaces. These data include hardware resource usage and service resource usage of the system as well as operating state data of proxy systems like network resources. State monitoring modules take monitoring items as unit and provide these state data to the state self-sensing module.

(2) State self-sensing. State self-sensing module is the core of the whole system. First, it uses sample data degree as monitoring items in the proxy system to establish the ARX (Auto-Regressive with Extra Inputs) linear model. Monitoring item data acquired during state monitoring are analyzed. The current operating state is identified by judging whether the linear relationship of monitoring items is broken. It determines system repair and adjustment according to the perceived state of the proxy system. When adjustment is needed, the analyzer sends the current system abnormal state description (S) to the abnormal positioning component. State self-sensing module includes abnormal positioning sub-components which are responsible for analyzing current system abnormal states based on system knowledge base and locating abnormalities with Naive Bayes classifier. System knowledge base covers system modules information, monitoring item information and action set of system abnormality adjustment and recovery. Actions mainly include parameter adjustment and resetting. Actions are set by the VM administrator. The system administrator which is recognized as field expert is to adjust other contents in system knowledge base according to practical situations. After abnormality positioning, the abnormal positioning component will transmit perceived parameters, including abnormal module information and system abnormal condition information, to the autonomous decision component.

(3) Independent decision-making. Independent decision-making module is to build a group of action sequences that could achieve preset goal by adjusting system running states reasonably through the greedy algorithm based on acquired abnormality information. This group of action sequences is the core of the strategy. Independent decision-making module transfers this group of action sequences to the strategy implementation module. The independent decision-making module has a strategy knowledge base. When an effective strategy is made, it is stored in the strategy knowledge base and correlated with the corresponding S. When system needs similar system adjustment, corresponding strategy in the strategy knowledge base will be activated.

(4) Strategy implementation. Strategy implementation module is to execute strategies made by the independent decision-making module. Every action in the system knowledge base corresponds to one executable shell script file. Therefore, core of the strategy implementation module is one shell script actuator.

4 Abnormality Self-Sensing Technology of VM System

State self-sensing module is the core of autonomous trustworthy enhancement framework. Perception of system trustworthy changes are the precondition to enhance trustworthy of cloud system. System trustworthy changes are manifested by system change from normal state to abnormal state. In this paper, abnormal state is the state when system can only provide poor service ability or is unable to provide service due to fault, service resource shortage or network intrusion. System state changes are reflected by numerical value fluctuation of monitoring items (e.g., CPU utilization and network flow) directly. This chapter focuses on monitoring items and realizes self-sensing of system abnormal state base on correlation of monitoring items. An abnormality positioning method is proposed.

Virtual Machine Introspection (VMI) provides the ability to monitor virtual machines in an agentless fashion by gathering VM execution states from the hypervisor. In this paper we also used VMI monitoring the virtual machine system environment automatically. When virtual nodes are facing some problem, VMI technique will use self-decision ways and self-recover from the terrible situation.

4.1 Correlation Study of Self-Sensing Monitoring Items

4.1.1 *Correlation Analysis of Monitoring Items*

For a system that is running stably, different state information shall be kept stable. Changes of one monitoring item will cause changes of other related monitoring items. System monitoring items could reflect system running well. If the VM system is viewed as an integral, its input and output are user requests and feedback to user requests, respectively. All monitoring items change with user requests. There will be certain correlation among state information of different monitoring items when system is operating normally. Once one module has problems, monitoring items that are closely correlated fluctuate violently, while those which are less correlated are still kept normal. This means that state information of one monitoring item is impossible to reflect state of the whole system. To realize self-sensing

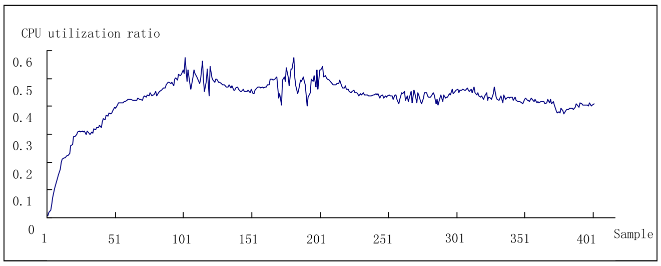


Fig. 2. CPU utilization curve.

state of system monitoring items, it is necessary to find correlations among monitoring items and unify them.

An experimental test of Socks proxy service was made in order to discover correlations among monitoring items. Numerical value changes of CPU utilization, linking number of 1080 port and linking number of system TCP of SocksV5 proxy service are recorded in the experiment. Results are shown in Figs. 2, 3, and 4, respectively. Numerical value curves of these three monitoring items show similar variation trend, indicating that they may have certain linear relationship. A scatter diagram was drawn to confirm the existence of such linear relationship. It uses the linking number of 1080 port as the x-axis and the CPU utilization and linking number of system TCP as the y-axis (Fig. 5). It can be seen from Fig. 5a that all sample points fall close to one straight line, indicating that there's a linear relationship between the linking number of 1080 port and the CPU utilization. Similarly, Fig. 5b could prove the existence of linear relationship between the linking number of 1080 port and the linking number of system TCP.

Linear relationships among CPU utilization, the linking number of 1080 port and the linking number of system TCP of SocksV5 proxy service have certain references. For example, when the linking number of 1080 port (x) increases, user requests increase accordingly. When processing user requests, the front-end authority server has to analyze user identity first, while the background server deals with requests. These operations will surely change CPU service time. If CPU utilization of background server processing is set y, there must be a linear relationship between x and y to make $|y - f(x)| \leq \varepsilon$. Such linear relationship is reflected obviously in previous curves. It often has to set many monitoring items to monitor running conditions of the system. Such linear relationship may also exist among other monitoring items. But it has to illustrate that the existence of linear relationship doesn't mean there's an interaction between these two monitoring items. In some cases, any two monitoring items may only have a linear relationship on numerical values.

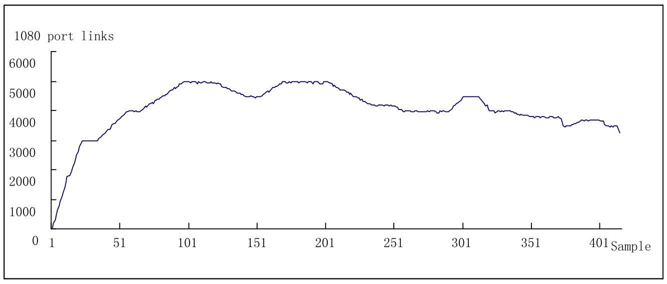


Fig. 3. Linking number curve of 1080 port.

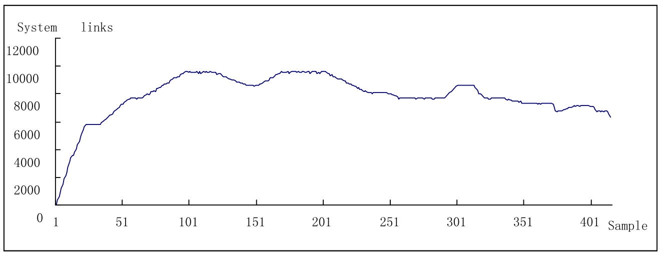


Fig. 4. Linking number curve of system TCP.

4.1.2 Linear Relationship Modeling of Monitoring Items

This part tries to establish models for discovered linear relationships by using a mathematical model. Since the acquired state information of monitoring items are time series, ARX model [24] is chosen. First, the ARX model and its solving method are introduced. ARX model structure is shown as

$$A(z^{-1})Y(k) = B(z^{-1})X(k) + e(k). \tag{1}$$

Where $X \in R^{n_x}$ and $Y \in R^{n_y}$ are system input and output; $e \in R^{n_y}$ is white noise; z^{-1} is backward shift operator, $A(z^{-1})$ and $B(z^{-1})$ are:

$$A(z^{-1}) = I + \sum_{i=1}^{n_a} A_i z^{-i}, B(z^{-1}) = \sum_{i=1}^{n_b} B_i z^{-i}, \tag{2}$$

Where I is $n_y \times n_y$ dimensional unit matrix; A_i is $n_y \times n_y$ dimensional matrix; and B_i is $n_y \times n_x$ dimensional matrix. For single-input system and single-output system, Equation (2) can be converted into:

$$\begin{aligned} &y(t) + a_1 y(t-1) + \cdots + a_n y(t-n) \\ &\quad = b_0 x(t-k) + b_1 x(t-k-1) + \cdots + b_m x(t-k-m). \end{aligned} \tag{3}$$

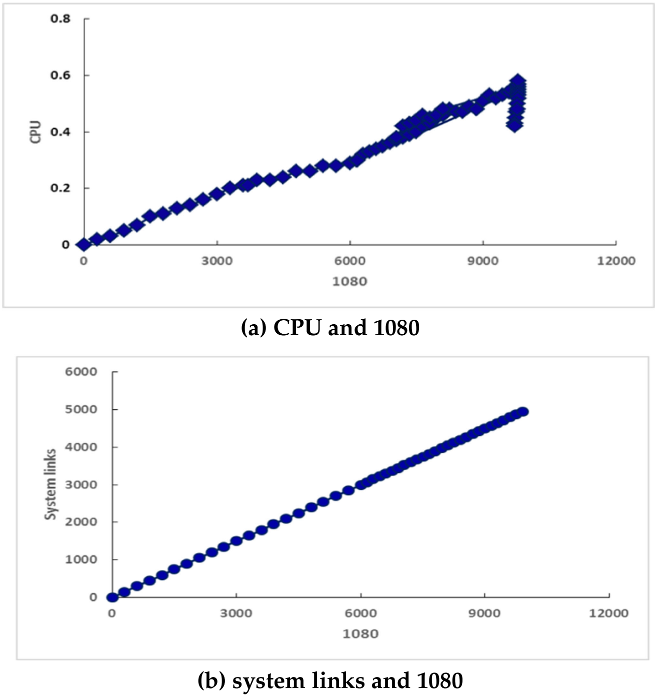


Fig. 5. Scatter diagram for verifying linear relationship among monitoring items.

Where a_i is coefficient of $y(t-i)$ and b_j is coefficient of $x(t-i)$. i, j are time delay.

During modeling, ARX model parameters are estimated through the least square method. Let $\theta = [a_1, a_2 \cdots a_n, b_0, b_1, \ldots b_m]^T$ $\varphi(t) = [-y(t-1), \ldots -y(t-m), x(t-k), \ldots x(t-k-m)]^T$ So it can get:

$$y(t) = \varphi(t)^T \theta. \tag{4}$$

Take one pair of monitoring items (x and y) as samples. Let $O_N = \{x(1), y(1), \ldots x(N), y(N)\}$

$$V_N(\theta, O_N) = \frac{1}{N}\sum_{t=1}^{N} (y(t) - \hat{y}(t|\theta))^2 = \frac{1}{N}\sum_{t=1}^{N} (y(t) - \varphi^T(t)\theta)^2.$$

According to the least square method, $\hat{\theta} = \arg\min V_N(\theta, O_N)$. Calculate the partial derivative of θ,

$$\frac{d}{d\theta} V_N(\theta, O_N) = \frac{2}{N}\sum_{t=1}^{N} \varphi(t)(y(t) - \varphi^T(t)\theta) = 0. \tag{5}$$

Solving equation of parameter $\hat{\theta}$ could be gained:

$$\hat{\theta} = \left[\sum_{t=1}^{N} \varphi(t)\varphi^T(t)\right]^{-1} \sum_{t=1}^{N} \varphi(t) y(t). \tag{6}$$

Orders of ARX model are necessary to solve ARX model of linear relationship of one pair of monitoring items. In the experiment, ideal $\hat{\theta}$ that won't influence efficiency could be gained when $[m, n, k] < 5$. To get the optimum order of the ARX model and the corresponding $\hat{\theta}$, an evaluation function has to set, such as Akaike information criterion (AIC) and Minimum Description Length (MDL). In this paper, AIC was used.

In general case, AIC is

$$AIC = 2k - 2\ln(L), \tag{7}$$

where k is number of parameters and L is the likelihood function. Suppose that error of the established model obeys independent normal distribution. Let n be sample size and RSS be residual sum of squares. So calculation formula of AIC is

$$AIC = 2k + n\ln(RSS/n). \tag{8}$$

In Equation (8), k is increased to enhance fitting. However, AIC also introduces n to avoid overfitting. Since modeling is to find the model that contains the fewest parameters but could explain sample data the best, model with minimum AIC is the first choice.

To sum up, parameter extraction algorithm of ARX model is shown in Algorithm 1.

When establishing the ARX model for two monitoring items, it has to determine which one used as y value. In this paper, one of two monitoring item was chosen randomly as y to calculate $\hat{\theta}$ according to Algorithm 1. Next, the other monitoring item was used as y to calculate $\hat{\theta}$ again. The higher $\hat{\theta}$ in AIC evaluation was used as the ARX model parameter of these two monitoring items.

4.1.3 Screening of Linear Relationships Between Monitoring Items

Linear relationship between two monitoring items could be established by ARX modeling. However, not every two monitoring items will have linear relationship. For instance, proxy service program will apply for memory space according to needs and memory usage will increase accordingly. When requests decrease, these memory spaces won't be set free immediately and the memory usage won't change greatly. Therefore, there's no linear relationship between memory usage and CPU service time. Linear relationships between any two monitoring items are calculated through violent search first. Later, appropriate linear relationships are screened out from them.

Algorithm 1. Parameter Extraction Algorithm of Linear Relationship Between Monitoring Items

Input: monitoring data of a pair of monitoring items: $\{x(1), y(1), \ldots x(N), y(N)\}$
Output: parameter $\hat{\theta}$ of ARX model

```
1: for each [m, n, k]< 5
2:     Calculate $\hat{\theta}_i$ according to Equations (6)
3:     Calculate $AIC(f_{y,x}(\hat{\theta}_i))$ according to Equations (8)
4: $\hat{\theta} \leftarrow \arg\min_{\hat{\theta}_i} AIC(f_{y,x}(\hat{\theta}_i))$
5: Return $\hat{\theta}$
```

Upper limit of AIC_T is set. In other words, only linear relationships which meet Equation (9) are chosen.

$$AIC_T > AIC(f_{y,x}(\hat{\theta})). \tag{9}$$

Noise may exist in monitoring data and disturb AIC calculation, making a pair of monitoring items with linear relationship omitted. As a result, linear relationships are screened by calculating existence probability. Monitoring data of monitoring items are divided into segments. Each segment contains monitoring data within a certain time length (s). As time goes on, several monitoring data segments (length = s) could be gained, recorded as I_i. The function $p_k(f_{y,x}(\hat{\theta}))$ is defined:

$$p_k(f_{y,x}(\hat{\theta})) = \frac{card(\{I_i | AIC_T > AIC_i(f_{y,x}(\hat{\theta})), i = 1, 2, \ldots k\})}{k} \geq P_T. \tag{10}$$

Equation (10) shows probability of $AIC_T > AIC(f_{y,x}(\hat{\theta}))$ in k monitoring data segments (length = s). In this way, not only inappropriate linear relationship could be filtered, but also linear relationships can be screened within every monitoring segment, thus realizing online operation of the algorithm and protecting timeliness of linear relationships.

4.2 Abnormal Module Positioning

4.2.1 System Mapping of Monitoring Items

To realize abnormality positioning, the cloud proxy system is divided into several modules according to monitoring items. Changes of different modules affect changes of several monitoring items. Fluctuation of one monitoring item may be caused by changes of different modules. Therefore, a one-to-more correlation between monitoring items and

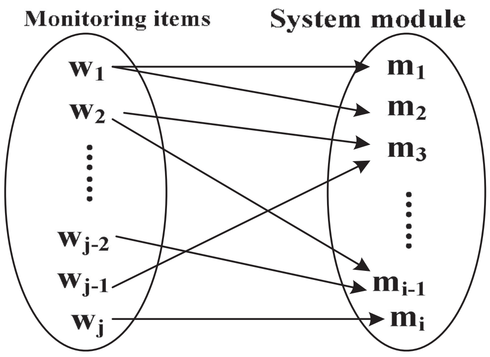


Fig. 6. Mapping relations between monitoring items and system modules.

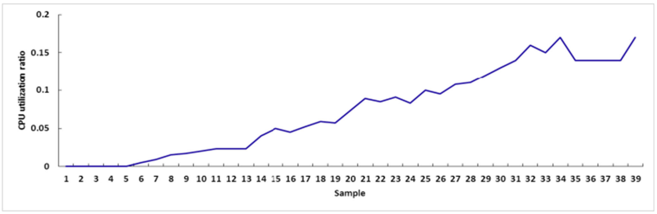


Fig. 7. Variation curve of CPU utilization.

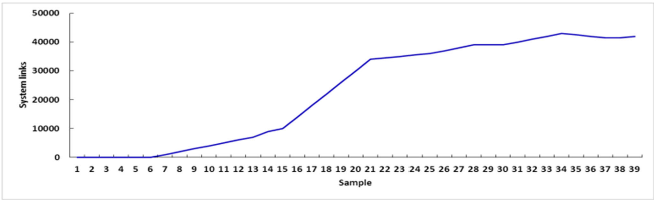


Fig. 8. Variation curve of linking number of TCP.

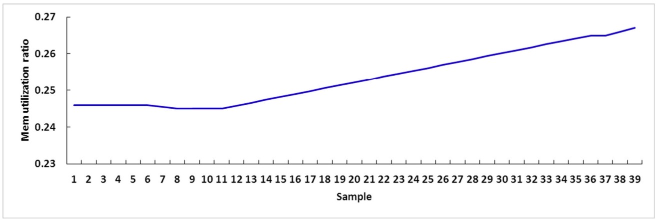


Fig. 9. Variation curve of memory usage.

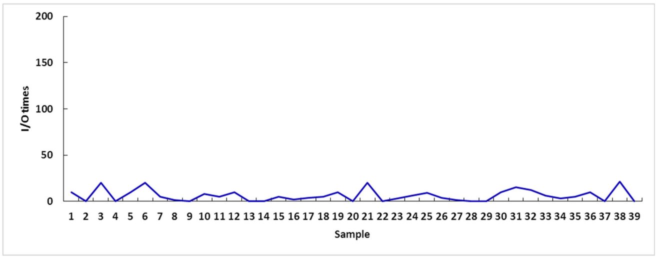


Fig. 10. Variation curve of disk read-write times.

system modules could be built. System modules are represented by m_i and monitoring items are expressed w_i. Therefore, the one-to-more correlation could reflect the mapping relation between monitoring item set W and system module set M: $W \rightarrow M$ (Fig. 6).

Such one-to-more correlation between monitoring items and system modules could be concluded from module test. Module test, or called unit test, is the basic test activity during software development. It is to test whether functions of modules are normal under execution of one single module. At the same time, changes of every monitoring item are recorded and monitoring items that are strongly correlated with module are discovered, so correlations between system modules and monitoring items could be identified. One module can influence changes of one monitoring item, indicating that violent fluctuation of numerical value of this monitoring item may be caused by abnormality of this module. On this basis, correlations between monitoring items and system modules could be concluded.

A verification experiment was carried out for better explanation of the correlations between system modules and monitoring items. Effect of different system modules on some monitoring items when user requests per unit time increase continuously is shown in Figs. 7, 8, 9, and 10. Request module is mainly to analyze user requests in Socks proxy process and initiate connections to application server or construct monitoring port. It can be seen from Figs. 7, 8, 9, and 10 that as user requests increase continuously, CPU utilization, the linking number and memory usage all increase gradually, but no trend variation of read-write times of disk is discovered. This is because request module has to analyze user requests and will occupy CPU time, while connections between request module and the application server will increase the linking number. Increase of memory usage is caused by continuous increase of request processing in the request module, but there's no file reading and writing during execution flow of request module. Based on independent test of request modules, it can conclude that request module is related with CPU utilization, the linking number and memory usage, but is uncorrelated with read-write times of the disk.

For the convenience of managing correlations between system modules and monitoring items, system knowledge base is added in the autonomous trustworthy enhancement framework. System knowledge base is mainly managed by system administrator. Correlations between system modules and monitoring items are recorded as across linkers in the system knowledge base and could be expressed by data structure of sparse matrix.

4.2.2 Determine Abnormality Threshold of Linear Relationship Between Monitoring Items

After $w_i = f(w_j)$ and $M \rightarrow W$ are established, abnormal module positioning is based on certain system knowledge. If one or some modules have abnormalities, numerical values of corresponding monitoring items will fluctuate. Since linear relationships between monitoring items are established under normal system operation, numerical values of monitoring items fail to meet $w_i = f(w_j)$. Therefore, the abnormal module positioning problem is to judge whether $w_i = f(w_j)$ is supported.

Equation (11) is introduced to judge whether the linear relationship between one pair of monitoring items is satisfied.

$$r(t) = w_i(t) - \overline{w_i}(t). \tag{11}$$

Equation (11) determines whether a linear relationship is broken, where $\overline{w_i}(t)$ is w_i value at t, which is calculated

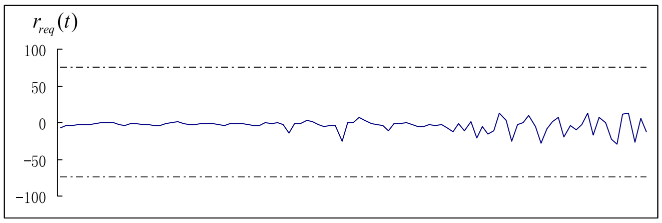


Fig. 11. Variation curve of $r_{req}(t)$ under normal operation.

according to $w_i = f(w_j)$. $r(t)$ reflects the fluctuation degree of monitoring items. During stable running of the system, $r(t)$ varies within a threshold interval. Once $r(t)$ exceeds the threshold, the linear relationship is broken and some modules may become abnorma

Selection of threshold shall consider that different monitoring items have different amplitudes of variation and it also shall avoid influence of noise data. In this paper, threshold was set in the numerical value interval where 97 percent $r(t)$ of training data of monitoring items fall in

$$\gamma = n \cdot \arg_{\hat{r}}\{p(|r(t)| \leq \hat{r}) = 0.97\}, \tag{12}$$

where n is adjusting parameters of different monitoring items. Since linear relationships change periodically during normal operation of the proxy system, γ adjust and change automatically. Due to interference of noise data, it couldn't determine that the linear relationship is broken when $\gamma < |r(t)|$ until after it has lasted for one perception period. Then, the abnormal module positioning is triggered.

A simple verification experiment was made for better explanation of the linear relationships between monitoring items and determination of threshold. In the experiment, ARX model of linear relationships between any two of negotiate dialogue number of proxy server (w_{neg}, dialogue number of SocksV5 server during the negotiation period), request dialogue number (w_{req}, dialogue number of SocksV5 server during the request period), the linking number of 1080 port (w_{1080}) and the linking number of system (w_{con}) was established. A total of three linear relationships were calculated from algorithm 1: $w_{req} = f(w_{neg})$, $w_{neg} = f(w_{1080})$ and $w_{con} = f(w_{1080})$. Corresponding ARX models are shown in Equations (13), (14), and (15). Thresholds of w_{req}, w_{con} and w_{neg} were calculated 74, 242 and 98 respectively according to training data.

$$\begin{aligned} w_{req}(t) = &-0.79w_{req}(t-1) - 0.58w_{req}(t-2) \\ &- 0.15w_{req}(t-3) + 0.99w_{neg}(t) + 0.79w_{neg}(t-1) \\ &+ 0.6w_{neg}(t-2) + 0.15w_{neg}(t-3) \end{aligned} \tag{13}$$

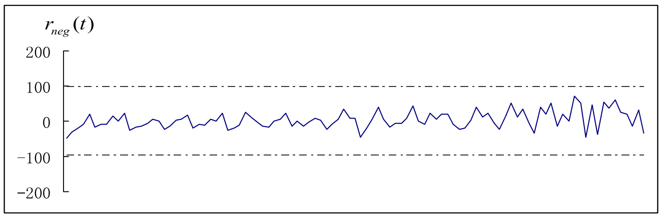


Fig. 13. Variation curve of $r_{neg}(t)$ under normal operation.

$$\begin{aligned} w_{con}(t) = &\,0.33w_{con}(t-1) + 0.2w_{con}(t-2) + 0.21w_{con}(t-3) \\ &+ 0.09w_{neg}(t-4) + 1.48w_{1080}(t) - 0.15w_{1080}(t-1) \\ &- 0.4w_{1080}(t-2) - 0.37w_{1080}(t-3) - 0.25w_{1080}(t-4) \end{aligned} \tag{14}$$

$$\begin{aligned} w_{neg}(t) = &\,0.34w_{neg}(t-1) + 0.22w_{neg}(t-2) + 0.42w_{neg}(t-3) \\ &+ 0.02w_{neg}(t-4) + 1.04w_{1080}(t) - 0.38w_{1080}(t-1) \\ &- 0.21w_{1080}(t-2) - 0.44w_{1080}(t-3). \end{aligned} \tag{15}$$

Judgment equations of w_{req}, w_{con} and w_{neg} are

$$r_{req}(t) = w_{req}(t) - \overline{w}_{req}(t) \tag{16}$$

$$r_{con}(t) = w_{con}(t) - \overline{w}_{con}(t) \tag{17}$$

$$r_{neg}(t) = w_{neg}(t) - \overline{w}_{neg}(t). \tag{18}$$

Figs. 11, 12, and 13 show variations of $r_{req}(t)$, $r_{con}(t)$, and $r_{neg}(t)$ under normal operation of the VM server. To describe determination of linear relationships, malicious dialogue attacks are applied during the request period of SocksV5. In other words, malicious users send bad requests to SocksV5 server and the target server in request has problems. SocksV5 server couldn't establish correct connection or are unable to get normal services. As a result, the malicious dialogues cloud maintain for a certain period, during which service resources of SocksV5 server are occupied maliciously. When malicious dialogues reach to a certain extent, request process that is responsible to response to user requests couldn't handle requests timely, leading to a server abnormality.

Variations of $r_{req}(t)$, $r_{con}(t)$, and $r_{neg}(t)$ under abnormal conditions of the VM server are shown in Figs. 14, 15, and 16. Upon attacking by malicious dialogues, abundant malicious dialogues take place in the request period. Request processing capacity declines, so many dialogues which have finished negotiation period couldn't be processed. Therefore, w_{req} is far lower than the normal level compared to w_{neg} and user requests couldn't receive responses. No

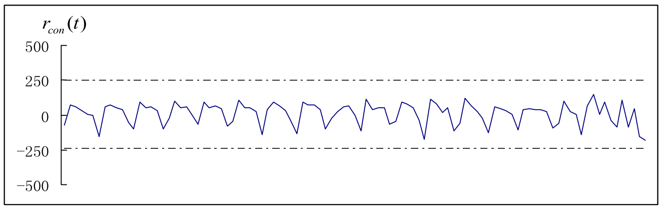


Fig. 12. Variation curve of $r_{con}(t)$ under normal operation.

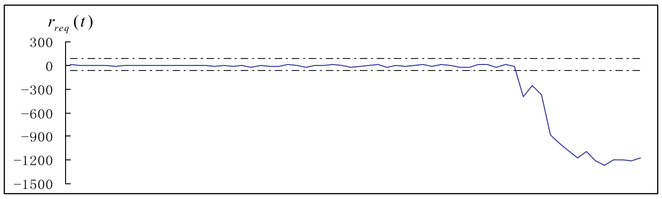


Fig. 14. Variation curve of $r_{req}(t)$ under malicious dialogue attacks.

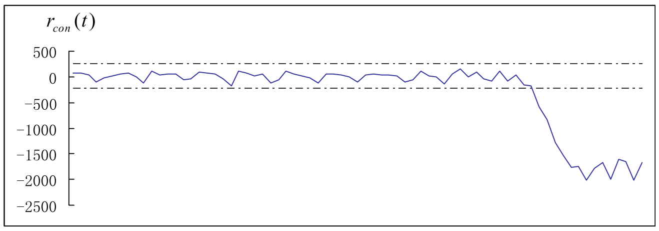


Fig. 15. Variation curve of $r_{con}(t)$ under malicious dialogue attacks.

application server is availabe to establish connections, so w_{con} is far lower than the normal level compared to w_{1080}. w_{req} and w_{con} abnormalities could be self-sensed through $w_{req} = f(w_{neg})$ and $w_{neg} = f(w_{1080})$. Although w_{req} and w_{con} are abnormal, $w_{con} = f(w_{1080})$ still has to be analyzed, determining that the system is under abnormal state. Fig. 15 is the variation curve of $r_{neg}(t)$, which is concluded through $w_{con} = f(w_{1080})$. Obviously, $r_{neg}(t)$ is within the normal threshold interval, indicating that w_{1080} and w_{neg} are normal. Therefore, w_{req} and w_{con} are determined abnormal.

4.2.3 Abnormal Module Positioning Based on Bayesian Classification

Some set definitions are introduced to better explain how to use Bayes classifier for abnormal module positioning. Whether monitoring items meet $w_i = f(w_j)$ could be determined through $R(t)$ and γ. After the monitoring item set $W_{error} = \{w_i | (|R_{w_i}(t)| > \gamma_{w_i})\}$ that couldn't meet linear relationship is concluded, abnormal modules are located based on $M \to W$. The alternative abnormal module set M_{error} corresponding to W_{error} could be gained through the mapping relation $W \to M$.

The goal of abnormal module positioning with Bayes classifier is to screen M_{error} and find out modules that are most likely to suffer abnormalities. In Section 4.2, the mapping relations between monitoring items and system modules are expressed by data structure of sparse matrix. To make full use of Bayes classifier, attributes $P(w_i|m_j)$ have to be added to every node, which reflect probability of numerical fluctuation of w_i when module m_j is abnormal. Therefore, correlation between w_i and m_j could be expressed in Fig. 17. "X" means that there's no mapping relation between w_i and m_i.

Naive Bayes Classifier is a classic classifier in pattern recognition and Bayes formula is its theoretical basis. With prior probability in probability model, naive Bayes classifier could be trained through supervised learning. The maximum posterior probability is chosen as the affiliation of classification objects [25]. The Bayes formula is

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{19}$$

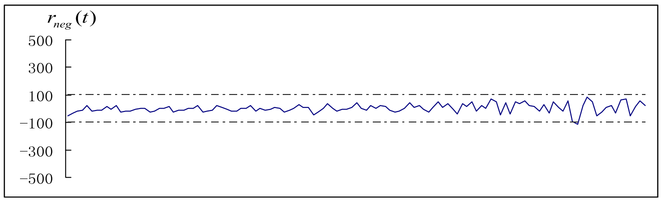


Fig. 16. Variation curve of $r_{neg}(t)$ under malicious dialogue attacks.

$$w_i \left\{ \overbrace{\begin{array}{cccc} P(w_1|m_1) & P(w_1|m_2) & \cdots & P(w_1|m_n) \\ P(w_2|m_1) & \mathrm{X} & \cdots & \mathrm{X} \\ \cdots & \cdots & \cdots & \cdots \\ P(w_m|m_1) & \mathrm{X} & \cdots & P(w_m|m_n) \end{array}}^{m_j} \right.$$

Fig. 17. Sparse matrix with attributes $P(w_i|m_j)$.

where $P(A)$ is prior probability; $P(B|A)$ is joint probability; $P(A|B)$ is posterior probability. The general expression of naive Bayes classifier is

$$classify(f_1, \ldots..f_n) = \arg\max_c p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c). \tag{20}$$

Given M_{error} and W_{error}, abnormal module positioning problem can be converted into the Bayes problem of posterior probability $P(m_j|w_i, \ldots.w_k)$ when m_j is abnormal. $w_i, \ldots.w_k \in W_{error}$ and $m_j \in M_{error}$. Calculation formula of $P(m_j|w_i, \ldots.w_k)$ is

$$P(m_j|w_i, \ldots.w_k) = \frac{P(w_i, \ldots.w_k|m_j) \cdot P(m_j)}{P(w_i, \ldots.w_k)} \tag{21}$$

Since fluctuation of monitoring items is conditionally independent with respect to other monitoring items and $P(w_i, \ldots.w_k)$ is a relative invariant, Equation (21) can be rewritten into

$$\begin{aligned} P(m_j|w_i, \ldots, w_k) &= P(w_i|m_j) \cdot \cdots \cdot P(w_k|m_j) \cdot P(m_j) \\ &= P(m_j) \prod P(w_l|m_j) \end{aligned} \tag{22}$$

Where $w_l \in W_{error}$, In Equation (22), $P(w_l|m_j)$ is attribute values added to correlations between monitoring items and system modules. The initialization value of $P(w_l|m_j)$ is reciprocal of the number of monitoring items which are mapped into the module m_j. The initial value of $P(m_j)$ is $1/|M|$, that is, the reciprocal of the number of elements in the system module set. $P(m_j)$ and $P(w_l|m_j)$ will be updated immediately upon abnormalities of the proxy system.

To sum up, it can be believed that the module with the biggest $P(m_j|w_i, \ldots, w_k)$ is most likely to suffer abnormalities. Number of the abnormal module $(\hat{j})$ could be discovered according to following equation:

$$\hat{j} = \arg\max(P(m_j|w_i, \ldots, w_k)). \tag{23}$$

In practical application, to avoid false alarm and influences on strategy making, $P(m_j|w_i, \ldots, w_k)$ will be ordered. Besides, the alternative abnormal module list will be formed, and TOP n in the list are determined as abnormal modules. Abnormal module information collected using naive Bayes classifier can support systematic formulation of security enhancement strategy and artificial strategy making.

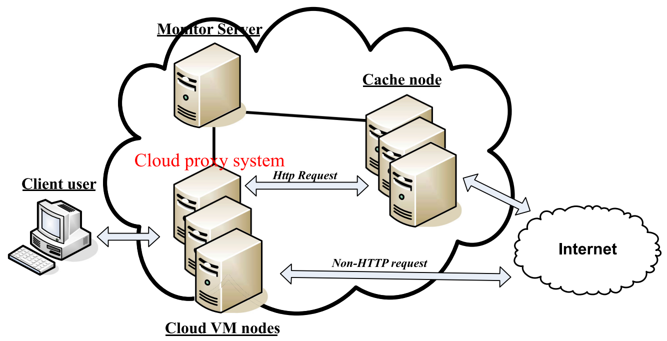


Fig. 18. Cloud proxy system.

TABLE 2
Universal Monitoring Items of Cloud Proxy System

Category	Sub-category	Name of monitoring item
Hardware resources	Processor	#CPU service time #System time #User time #IO time
	Memory	#Memory usage #buffer usage #cached usage #swap usage #share usage
	Disc	#Disc read times # Disc write times
Network resources	Network	#Uplink packet #Downlink packet #Flow #System connection number

5 Autonomic Evaluation and Analysis

5.1 Experimental Setting

5.1.1 VM System Environment

The cloud system used in the experiment is shown in Fig. 18. It is composed of SocksV5 VM node, HTTP Cache VM node and monitoring server. The whole cloud poxey system is centered at SocksV5 VM node. Since SocksV5 node protocol doesn't support HTTP Cache, Dante of the SocksV5 software is reconstructed, so GET requests of HTTP protocol could be separated from CONNECT requests of all users. These GET requests are sent to the HTTP Cache VM node of deployed and reconstructed Squid software. The HTTP Cache server is responsible for collecting HTTP resources for user requests and returning them to the SocksV5 node. Meanwhile, whether these HTTP resources shall be stored in HTTP Caches is determined according to preset caching rules. SocksV5 server chooses HTTP Cache server by using Chord ring. It calculates hash of URL, which are then used to find the correct HTTP Cache server in the Chord ring. SocksV5 server acquires resources from Internet directly for non-GET requests.

Cloud experimental environment consists of 25 machines, of which 10 proxy servers, 10 cache servers, 5 monitoring server. 105 virtual machines are simulated on the physical machine. All physical machine servers in cloud system are Sugon Tiankuo A610r-G and have same configurations (Table 1).

5.1.2 Description Software and Function Library of Cloud Proxy System

Dante is used as Socks proxy software in the experiment. It is an open-sourcing free socks proxy software developed by Norway Inferno Nettverk A/S Company and supports SocksV4 and SocksV5. It can be used as a firewall between networks to control outflow. Moreover, Dante supports Socks server ad Socks proxy client. In this experiment, Dante 1.2.3 is reconstructed and SocksV5 VM node is established.

TABLE 1
Basic Configuration of Physical Machine Servers in Cloud System

Configurations
Sugon Tiankuo A610r-G
AMD Opteron 6128, 2 GHz, 8 cores
300 GB
4 GB
Integrated dual Gigabit-NIC
Red hat Enterprise Linux 5.4 64 bit

HTTP Cache VM node is established by Squid. Squid is an open-sourcing free web cache proxy software that supports HTTP, HTTPS and FTP protocols. Since only HTTP Cache of Squid is needed in the experiment, functions of Squid are simplified.

It is necessary to implement abundant matrix operations to establish the ARX model. IT++ matrix operations library, a C++ function library of mathematical functions that is derived from Sweden Chalmers University of Technology, is used.

When establishing the strategy knowledge base, B+tree is established using B+tree interface in Berkeley DB [26]. Berkeley DB is a lightweight open-sourcing database system and a kind of NoSQL database developed by Berkeley University. It provides common data access algorithms, such as B+tree, Hash, etc.

5.1.3 Experimental Analysis on Linear Relationships of Monitoring Items

Linear relationships of monitoring items are the key of trustworthy enhancement mechanism for cloud proxy system. Construction algorithms and screening method of linear relationships are introduced in Section 3.1. This part is experiment about establishing linear relationships among monitoring items. Experimental results are analyzed.

Since any two monitoring items in SocksV5 VM node or HTTP Cache virtual node change similarly under normal system operation, only one server in the SocksV5 virtual node and one server in the HTTP Cache node are monitored in the experiment. Numerical values of monitoring items are acquired and linear relationships among these monitoring items are established. Monitoring items involved in the experiment are listed in Tables 2 and 3. According to resource attributes, they are divided into hard resources, network resources and service resources. Monitoring items in Table 2 are universal to SocksV5 server and HTTP Cache server, whereas monitoring items in Table 2 are common monitoring items of two servers. All monitoring items concerning processor are CPU jiffies number in one monitoring period (10s).

In the experiment, a data network test platform SmartBits6000C is used to simulate that client initiates accessing requests to servers. The experiment lasts for 4,000s and

TABLE 3
Non-Universal Monitoring Items of Cloud Proxy System

Category	Sub-category	Name of monitoring item
Service resources	Proxy service resources	#linking number of 1080 port
		#Number of proxy request
		#Number of successful requests
		#Number of negotiation dialogue
		#Number of request dialogue
		#Number of I/O dialogue
	Cache service resources	#Number of cache requests
		#Usage of cache capacity
		#Number of minor page

numerical values of every monitoring item are recorded every 10s. A total of 400 groups of monitoring data are acquired. During this process, 24 monitoring items are monitored by the monitoring program. Since the first 15 monitoring items are universal, 21 monitoring items are collected from SocksV5 server and 18 monitoring items are gained from the HTTP Cache service. Data of these 39 monitoring items are used as sample data to establish linear relationships of monitoring items.

In the experiment, the first 80 experimental data are used as samples. Linear relationship between any a pair of monitoring items are established according to the Algorithm 1. A total of C_{39}^{2}, 741 linear relationships are established. Accumulative distribution of AIC evaluation values of these 741 linear relationships is shown in Fig. 19. These linear relationships are screened. Combining results of several experiments, it determines $AIC_T = 313$. Since AIC evaluation value will be influenced by sample size, length of monitoring segment during linear relationship screening is set 80, that is, 800s. Linear relationships are screened for four times with rest 320 groups of monitoring data. Screening results are shown in Fig. 20. After the first screening, 123 groups of linear relationships meet $AIC_T > AIC$, which reduces to 103 after second screening and 102 after the third screening. Subsequently, number of linear relationships that meet $AIC_T > AIC$ becomes stable, which proves that screened linear relationships could reflect numerical relationship between monitoring items truly.

In the experiment of establishing linear relationships, screening conditions are very strict, because only linear relationships that meet the screening condition could reflect numerical relation between a pair of monitoring items correctly. The experiment confirms stability of existing linear relationships in the system. According to

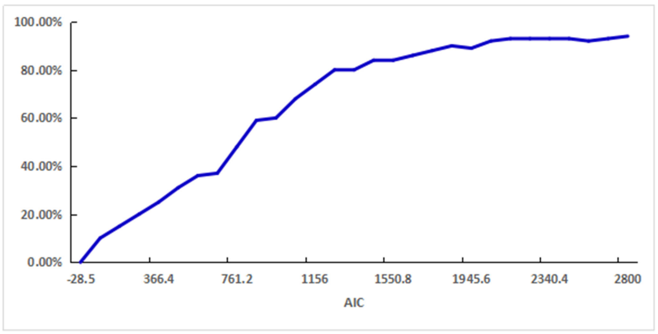


Fig. 19. Accumulation distribution curve of AIC evaluation values.

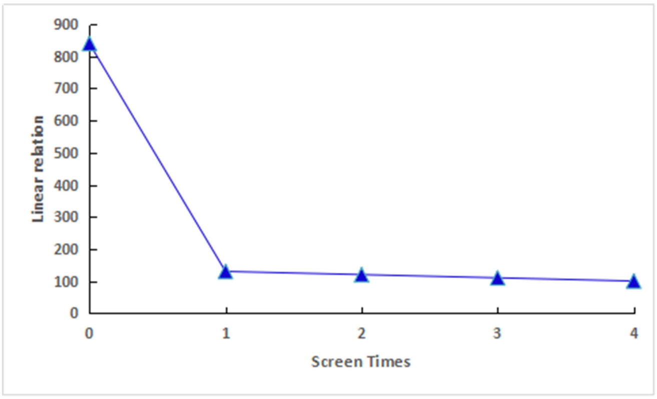


Fig. 20. Linear relationship screening results between monitoring items.

experimental results, there are 102 groups of satisfying linear relationships in cloud proxy system. With respect to 39 monitoring items, these 120 groups of linear relationships still could meet requirements of abnormal module positioning and independent decision-making on trustworthy enhancement.

5.2 Trustworthy Enhancement of Cloud Proxy System

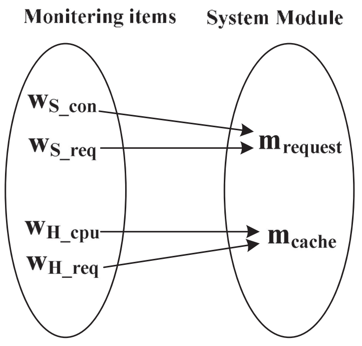
5.2.1 Construction of System Knowledge Base

In the experiment, proxy system confronts with two trustworthy threatens: malicious dialogue and cache hotspot. As stated above, malicious dialogue is a resource depletion attack mode against proxy system. Malicious users initiate CONNECT request of abnormal application server to SocksV5 server. But SocksV5 server cannot establish normal connection with abnormal application server or it suffers connection timeout. When connecting with SocksV5 server, malicious dialogue will last for a certain time. In this period, service resources of SocksV5 are occupied maliciously. Threats of cache hotspot are consequences of HTTP URL abundant users visiting the same HTTP Cache. Since hashed value of URL is used as basis of Cache resource positioning, there must be a group of URL which are all cached into the same HTTP Cache server. When this group of URL becomes the visit hotspot at some time, Cache hotspot problem takes place.

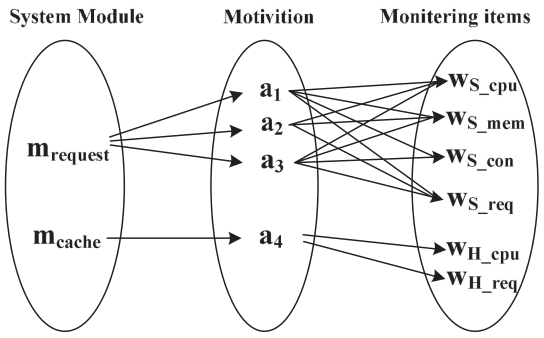
The process that normal user visits proxy server is simulated by SmartBits6000C to simulate malicious users, malicious dialogue client is operated on another two Sugon Tiankuo A610r-G physical machine servers with same configurations. In experiment of Cache hotspot, SmartBits is configured. Let simulated users visit a group of URL cached by the same HTTP Cache server to simulate Cache hotspot phenomenon.

Malicious dialogue and Cache hotspot will influence SocksV5 request module ($m_{request}$) and HTTP Cache module (m_{cache}) significantly $m_{request}$ is responsible to accomplish tasks in SocksV5 protocol during the request stage and appears as request process group m_{cache} is a single Squid process on the server.

Mapping relations and action settings in the system knowledge base shall be illustrated before experiment. Mapping relations between monitoring items and system modules are displayed in Fig. 21a W_s is monitoring items of



(a) Mapping relations between monitoring items and system modules



(b) Mapping relations among system modules, actions and monitoring items

Fig. 21. Mapping relations.

TABLE 4
Action Settings in System Knowledge Base

No.	Actions	Action description	Execution time
1	Restart request process	Restart the running request process.	2s
2	Adding request process	Create a new request process and add four request process	1.5s
3	Start application server information cache	Application server information Cache is used to store IP, port and causes of failed SocksV5 requests. In a short time, SocksV5 server won't tray to access to this server when it is available in user request again, but will return causes of failure directly. Besides, it will correct the linking number of monitoring items, that is, adding 1 to linking number.	2s
4	Start mutual assistance mode of HTTP Cache server	After two HTTP Cache servers start the mutual assistance mode, SocksV5 server will transfer some HTTP requests in the master Cache server to the slave server.	2.5s

SocksV5 server and W_H is HTTP Cache. For example, W_{s_con} is the linking number of SocksV5 server; W_{s_rep} is the number of request dialogue of SocksV5 server; W_{H_req} and W_{H_cpu} are HTTP requests and CPU service time of HTTP Cache server, respectively. Action settings are presented in Table 4. Effect and execution time of actions are described in Table 4. Mapping relations among system modulus, actions and monitoring items are shown in Fig. 21b. In the experiment, abnormal perception period is set 150s, which is equal to 15 monitoring periods. Recovery period is set twice that of the maximum action execution time (5s).

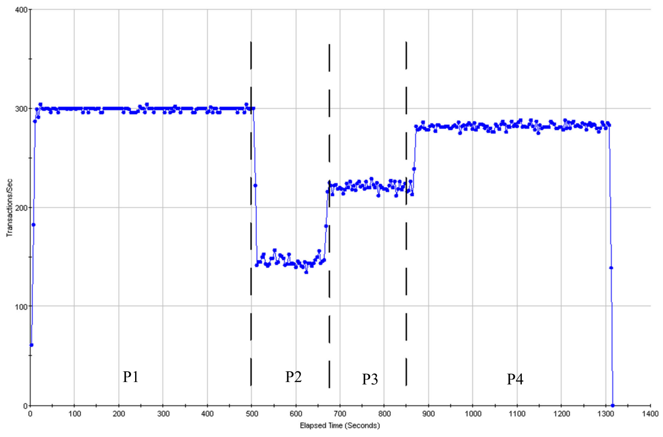


Fig. 22. Curve of SmartBits simulated user visits.

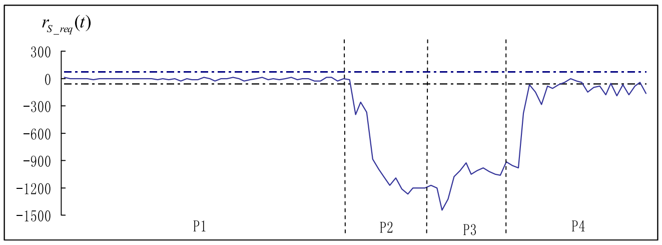


Fig. 23. Variation curve of $r_{S\_req}(t)$.

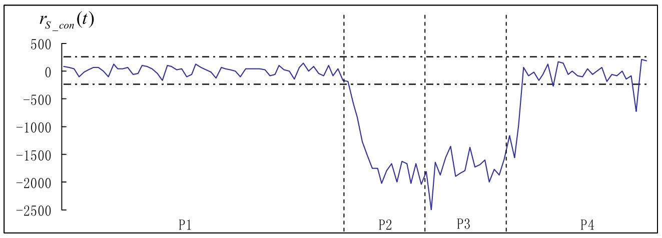


Fig. 24. Variation curve of $r_{S\_con}(t)$.

5.2.2 Malicious Dialogue Attack

In malicious dialogue attack experiment, linear relationships $w_{S\_req} = f(w_{S\_neg})$ Equation (9), $w_{S\_con} = f(w_{S\_1080})$ Equation (10) and $w_{S\_neg} = f(w_{S\_1080})$ Equation (11) which are established in Section 4.2 are used. $r_{S\_req}(t)$, $r_{S\_con}(t)$ and $r_{S\_neg}(t)$ are abnormal decision values of w_{S_req}, w_{S_con} and w_{S_neg} gained from above three linear relationships. Simulated ser visit frequency in SmartBits is 300 visits/s. The variation curve of successful user visits simulated by SmartBits is shown in Fig. 22.

Variation curves of $r_{S\_req}(t)$, $r_{S\_con}(t)$ and $r_{S\_neg}(t)$ during normal system operation are presented in P1 stage of Figs. 23, 24, and 25, respectively. In the P2 stage, 200 malicious dialogue attacks are initiated to SocksV5 server every second through two servers with malicious dialogue client. When the

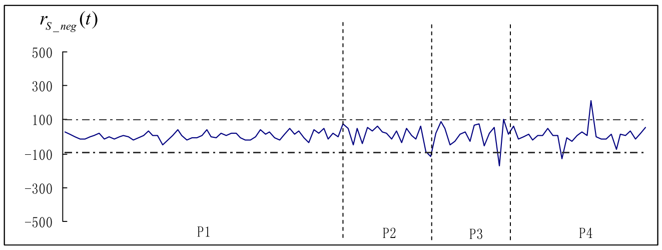


Fig. 25. Variation curve of $r_{S\_neg}(t)$.

system is attacked by malicious dialogue, $r_{S\_req}(t)$ and $r_{S\_con}(t)$ fluctuate, while $r_{S\_neg}(t)$ still maintains the normal level. It can determine that w_{S_req} and w_{S_con} become abnormal. After abnormal fluctuations of $r_{S\_req}(t)$ and $r_{S\_con}(t)$ last for one perception period (150s), it can determine that system abnormality occurs. Based on mapping relations between monitoring items and system modules in Fig. 21, abnormal module positioning determines that $m_{request}$ is abnormal. Later, the alternative action set $\{a_1,\ a_2,\ a_3\}$ is concluded according to mapping relations between system modules and actions in Fig. 22. At this moment, the system accomplishes abnormal state self-sensing and enters into independent decision-making stage. P3 stage in Figs. 23, 24, and 25 are variation curves after the first independent decision-making. At the first independent decision-making, the action sequence gained by using the optimal action sequence generating algorithm is $< a_1,\ a_2 >$. Since restarting request process and adding request process only increases the proxy service resources rather than enhance intrusion tolerance of the system fundamentally, the abnormal state will continue. P4 stage in Fig. 23, 24, and 25 is variation curves of $r_{S\_req}(t)$ and $r_{S\_con}(t)$ after the second independent decision-making and execution of the action sequence $< a_3 >$. After starting the application server information (Cache), the system tolerance to malicious dialogue increases. Therefore, $r_{S\_req}(t)$ and $r_{S\_con}(t)$ curves fall within the threshold again. Although they fluctuate violently, the system has recovered to normal state and service capability is protected, thus realizing the goal of independent intrusion tolerance.

5.2.3 Cache Hotspot

In the Cache hotspot experiment, linear relationships $w_{H\_req} = f(w_{S\_cpu})$ Equation (24), $w_{H\_cpu} = f(w_{S\_cpu})$ Equation (25) and $w_{S\_cpu} = f(w_{S\_con})$ Equation (26) are used.

$$\begin{aligned} w_{H\_req}(t) = {} & 0.6w_{H\_req}(t-1) + 0.25w_{H\_req}(t-2) \\ & + 0.09w_{H\_req}(t-3)\ + 0.07w_{H\_req}(t-4) \\ & + 6.16w_{S\_cpu}(t) - 3.38w_{S\_cpu}(t-1) \\ & - 2w_{S\_cpu}(t-2) - 0.8w_{S\_cpu}(t-3) \end{aligned} \tag{24}$$

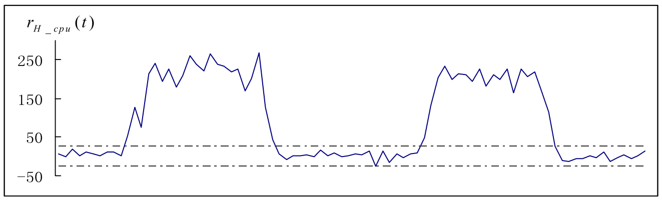


Fig. 27. Variation curve of $r_{H\_cpu}(t)$.

$$\begin{aligned} w_{H\_cpu}(t) = {} & 0.89w_{H\_cpu}(t-1) - 0.28w_{H\_cpu}(t-2) \\ & + 0.3w_{H\_cpu}(t-3)\ + 0.04w_{S\_cpu}(t-4) \\ & + 0.04w_{S\_cpu}(t-5) - 0.06w_{S\_cpu}(t-6) \\ & + 0.05w_{S\_cpu}(t-7) - 0.07w_{S\_cpu}(t-8) \end{aligned} \tag{25}$$

$$\begin{aligned} w_{S\_cpu}(t) = {} & 0.645w_{S\_cpu}(t-1) + 0.342w_{S\_cpu}(t-2) \\ & + 0.067w_{S\_con}(t) - 0.047w_{S\_con}(t-1) \\ & - 0.016w_{S\_con}(t-2)\ .\ - 0.005w_{S\_con}(t-3) \\ & + 0.002w_{S\_con}(t-4) \end{aligned} \tag{26}$$

Variation curves of $r_{H\_req}(t)$, $r_{H\_cpu}(t)$ and $r_{S\_cpu}(t)$ are shown in Figs. 26, 27, and 28. In the beginning, the whole system is unloaded. Later, start Smartbits and let simulated users visit a group of URL cached by the same HTTP Cache server to simulate the Cache hotspot phenomenon. Variations of $r_{H\_req}(t)$ and $r_{H\_cpu}(t)$ after Cache hotspot appeared in the VM system are shown in Figs. 26 and 27. Since HTTP Cache requests in SocksV5 server are not balanced, but are allocated to the same HTTP Cache server, the HTTP Cache server has relatively higher repeated load compared to the SocksV5 server. Therefore, $r_{H\_req}(t)$ and $r_{H\_cpu}(t)$ are higher than the normal levels. Variation curve of $r_{S\_cpu}(t)$ in Fig. 28 reflects that w_{S_cpu} is normal. After independent decision-making, the system implemented the action sequence $< a_4 >$ and load of HTTP Cache server is balanced. $r_{H\_req}(t)$ and $r_{H\_cpu}(t)$ recover to the normal level. One-dimensional eigenvalue of system abnormal state caused by this Cache hotspot is 158.2.

To verify effectiveness of the strategy knowledge base, Cache hotspot phenomenon is simulated again after the mutual assistance mode of HTTP Cache server is stopped. One-dimensional eigenvalue of this abnormal state is 159.9, which is the closest to the eigenvalue (158.2) of abnormal state caused by previous Cache hotspot after query of strategy knowledge base. Therefore, it tries to execute the action sequence $< a_4 >$. System state changes similarly.

To further verify effectiveness of the security strategy base, several simulations on malicious dialogue attack and Cache hotspot are implemented. One-dimensional eigenvalue distribution of abnormal state is shown in Fig. 29,

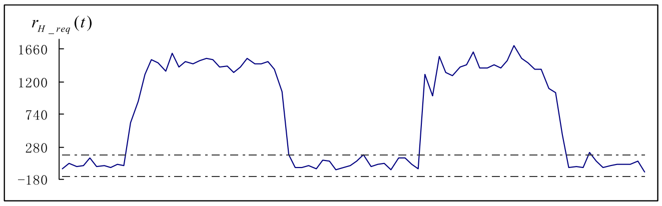


Fig. 26. Variation curve of $r_{H\_req}(t)$.

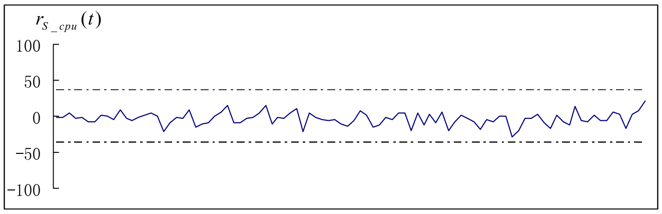


Fig. 28. Variation curve of $r_{S\_cpu}(t)$.

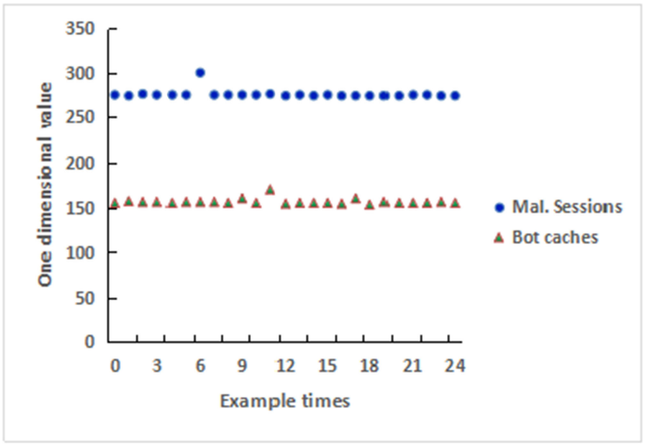


Fig. 29. One-dimensional eigenvalue distribution of abnormal state.

where y axis is one-dimensional eigenvalue of abnormal state and x axis is experiment times. For the same type of security threat with same intensity, monitoring items of the cloud proxy system fluctuate similarly, thus resulting in approximately equal one-dimensional eigenvalue. This proves that it is reasonable to adopt the same security enhancement strategy to abnormal states with similar one-dimensional eigenvalue.

6 Conclusions

Autonomic computing provides a new idea to solve trustworthy problem of cloud proxy system. Due to combination of autonomic computing and trusted computing, trustworthy enhancement technology based on autonomic computing becomes a research hotspot. Based on autonomic computing, this paper studies abnormal state self-sensing technology and independent decision-making on trustworthy enhancement of cloud proxy system. It proposes a trustworthy enhancement mechanism for cloud proxy system based on autonomic computing. Finally, trustworthy threats of malicious dialogue attack and Cache hotspot are used to test the proposed autonomous trustworthy enhancement mechanism. Experimental results are analyzed. Research contents are:

(1) A method to extract linear relationships of monitoring items based on ARX model is put forward to analyze correlations of monitoring items in the proxy VM system.

(2) Based on mapping relations between monitoring items and system modules, abnormal module positioning technique based on naive Bayes classifier is proposed, which realizes self-sensing of system abnormal states.

(3) According to mapping relations between perception parameters and system execution, an autonomous strategy generation algorithm based on greedy algorithm is established to solve the optimal action sequence generation problem in independent decision-making of trustworthy enhancement.

(4) Abnormal state characterization that takes fluctuation of monitoring items as eigenvalue is discussed. The one-dimensional expression method of eigenvalue of abnormal state is developed. A trustworthy enhancement strategy knowledge base with B+tree structure is established with one-dimensional eigenvalue of abnormal state.

To sum up, this paper studies trustworthy enhancement of proxy cloud system based on autonomic computing. Experimental results demonstrate that the proposed method improves trustworthy of proxy VM to a certain extent. But it still has some shortcomings. For instance, it requires system administrator to set related attributes of actions during independent decision-making. Automation degree needs to be further enhanced. Technologies related with machine learning could be introduced to reduce dependence on system administrator to a large extent, which will contribute better autonomous trustworthy enhancement effect.

Therefore, in the future work we will be introduced reinforcement learning such as Q learning method, based on the research of the Q learning self-decision framework, to enhance the decision-making ability of the overall system of cloud platform.

Acknowledgments

We thank the anonymous reviewers' comments, which are all valuable and very helpful for revising and improving our paper, as well as being important guiding significance to our research. This work is supported by the National Key Research and Development Program of China under grant No. 2016YFB0800801, the National Science Foundation of China (NSFC) under grant No. 61672186, 61472108, and the Specialized Research Fund for the Doctoral Program of Higher Education under grant No. 20132302110037.

References

[1] M. Parashar and S. Hariri, eds. "Autonomic computing: concepts, infrastructure, and applications," *CRC press*, pp. 3–18, 2006.

[2] J. O. Kephart, and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[3] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland, "A concise introduction to autonomic computing," *Advanced Eng. Informat.*, vol. 19, no. 3, pp. 181–187, 2005.

[4] Hewlett-Packard Infrastructure and Management Solutions for the Adaptive Enterprise. (2001). [Online]. Available: http://www.hp.com/products1/promos/adaptive_enter-prise/pdfs/vision_for_ae.pdf

[5] System Definition Model Overview White Paper. (2001). [Online]. Available: http://www.microso-ft.com/windowsserversystem/dsi/sdmwp.mspx

[6] R. Want, T. Pering, and D. Tennenhouse, "Comparing autonomic and proactive computing," *IBM Syst. J.*, vol. 42, no. 1, pp. 129–135, 2003.

[7] H. A. Mülle and L. O'Brien, "Autonomic computing," Softw. Eng. Institute, Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU/SEI-2006-TN-006, Apr. 2006.

[8] D. A. Menascé and M. N. Bennani, "Dynamic server allocation for autonomic service centers in the presence of failures," in *Autonomic Computing: Concepts, Infrastructure, and Applications*, S. Hariri and M. Parashar, Eds. Boca Raton, FL, USA: CRC Press, 2006, pp. 353–367.

[9] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2004. pp. 36–43

[10] E. Kiciman and Y.-M. Wang, "Discovering correctness constraints for self-management of system configuration," in *Proc. Int. Conf. Autonomic Comput.*, 2004, pp. 28–35

[11] M. L. Littman, N. Ravi, E. Fenson, and R. Howard, "Reinforcement learning for autonomic network repair," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2004, pp. 284–285

[12] A. Ranganathan and R. H. Campbell, "Self-Optimization of Task Execution in Pervasive Computing Environments," in *Proc. IEEE 2nd Int. Conf. Autonomic Comput.*, 2005, pp. 333–334

[13] A. Bohra, I. Neamtiu, P. Gallard, F. Sultan, and L. Iftode, "Remote repair of operating system state using backdoors," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2004, pp. 256–263.

[14] J. Wildstrom, P. Stone, E. Witchel, R. J. Mooney, and M. Dahlin, "Towards self-configuring hardware for distributed computer systems," in *Proc. 2nd IEEE Int. Conf. Autonomic Comput.*, 2005, pp. 241–249.

[15] M. J. Rutherford, K. M. Anderson, A. Carzaniga, D. Heimbigner, and A. L. Wolf, "Reconfiguration in the Enterprise JavaBean component model," in *Proc. IFIP/ACM Working Con. Component Deployment*, 2002, pp. 47–54.

[16] B. Srivastava and S. Kambhampati, "The case for automated planning in autonomic computing," in *Proc. Int. Conf. Autonomic Comput.*, 2005, pp. 331–332.

[17] J. O. Kephart, "Research challenges of autonomic computing," in *Proc. 27th Int. Conf. Softw. Eng.*, 2005, pp. 15–22.

[18] A B. Brown, et al., "Benchmarking autonomic capabilities: Promises and pitfalls," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2004, pp. 266–267.

[19] M. Amoretti, F. Zanichelli, and G. Conte, "Efficient autonomic cloud computing using online discrete event simulation," *J. Parallel Distrib. Comput.*, vol. 73, no. 6, pp. 767–776, Jun. 2013.

[20] M. A. Yahya, M. A. Yahya, A. Dahanayake, "Autonomic computing: A framework to identify autonomy requirements," *Procedia Comput. Sci.*, vol. 20, pp. 235–241, 2013.

[21] E. Kavvadia, S. Sagiadinos, K. Oikonomou, G. Tsioutsiouliklis, and S. Aïssa, "Elastic virtual machine placement in cloud computing network environments," *Comput. Netw.*, vol. 93, no. 3, pp. 435–447, Dec. 24, 2015.

[22] N. Msadek, R. Kiefhaber, and T. Ungerer, "A trustworthy, fault-tolerant and scalable self-configuration algorithm for organic computing systems," *J. Syst. Archit.*, vol. 61, no. 10, pp. 511–519, Nov. 2015.

[23] R. D. Pietro, F. Lombardi, F. Martinelli, and D. Sgandurra, "AntiCheetah: Trustworthy computing in an outsourced (cheating) environment," *Future Generation Comput. Syst.*, vol. 48, pp. 28–38, Jul. 2015,

[24] L. Ljung. *System Identification: Theory for the User*. Englewood Cliffs, NJ, USA: Prentice Hall, 1987, pp. 8–13.

[25] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, vol. 3. New York, NY, USA: Wiley, 1973, pp. 203–207.

[26] M. A. Olson, K. Bostic, and M. I. Seltzer, "Berkeley DB," in *Proc. FREENIX Track: USENIX Annu. Tech. Conf.*, 1999, pp. 35–37.

[27] W. Zhang, H. Xie, and C.-H. Hsu, "Automatic Memory Control of Multiple Virtual Machines on a Consolidated Server," *IEEE Trans. Cloud Comput.* 2016 [Online]. Available: http://dx.doi.org/10.1109/TCC.2014.2378794

[28] W. Zhang, S. Han, H. He, and H. Chen, "Network-aware Virtual Machine Migration in an Overcommitted Cloud," *Future Generation Comput. Syst.*, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.future.2016.03.009

Hui He received the PhD degree from the Department of Computer Science, Harbin Institute of Technology, China. She is currently an associate professor of network security center in the Department of Computer Science, China. Her research interests include distributed computing, IoT and big data analysis.

Weizhe Zhang is currently a professor in the School of Computer Science and Technology, Harbin Institute of Technology, China. His research interests include primarily in parallel computing, distributed computing, cloud and grid computing, and computer network. He has published more than 100 academic papers in journals, books, and conference proceedings. He is a member of the IEEE.

Chuanyi Liu received the PhD degree in computer science and technology from Tsinghua University, Beijing, China, in 2009. Chuanyi spent one year as a visiting scholar with the Digital Technology Center of the University of Minnesota, Minneapolis, Minnesota. He is now associate professor in Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China. His research interests include computer architecture, cloud security, and data protection. He has published more than 30 papers in related conferences and journals. He became member of the IEEE, in 2011.

Honglei Sun received the MS degree in software engineering from Harbin Institute of Technology, China. He is currently a teacher in Network Information Center of Harbin Institute of Technology, China. His research interests include mainly focused on cloud computing.