Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 12, December 2019, pp. 3227-3236
DOI: 10.23940/ijpe.19.12.p14.32273236

Application of Ontology and Multivariate Decision Diagram in Cloud Monitor Systems

Han Xu[a,*], William Cheng Chung Chu[b,*], and Jie Luo[c,d]

[a]*School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, 611731, China*
[b]*Department of Computer Science, Tunghai University, Taiwan, 40704, China*
[c]*School of Computer Science and Engineering, Beihang University, Beijing, 100191, China*
[d]*State Key Laboratory of Software Development Environmen, Beihang University, Beijing, 100191, China*

Abstract

Cloud computing is different from distributed computing and grid computing and has its own characteristics. The existing cloud system is not sufficient in the unified identification of cloud resources and the dynamic joining management of new resources. According to the characteristics of cloud computing, this paper introduces the idea of ontology on cloud monitor system (CMS) based on bionic autonomic nervous system (BANS) and uses ontology web language (OWL) language to describe the resources of the system. It also establishes a reusable extended resource expression model. At the same time, the use of the third-party tool Jena for OWL semantic query also gives the monitoring system the characteristics of a rapid semantic query, which further enhances the convenience of cloud resource management. In addition, based on the application of ontology, we also introduce multivariate decision diagram (MDD) multi-valued decision graph technology, which allows B-CMS to self-diagnose complex system faults. The combination of ontology and MDD greatly simplifies the monitoring and management of large-scale systems, providing a fast and standardized means for the intelligent diagnosis of systems.

Keywords: cloud computing; ontology; resource monitoring; fault detection; multivariate decision diagram; bionic autonomic nervous system

(Submitted on October 7, 2019; Revised on November 13, 2019; Accepted on November 17, 2019)

1. Introduction

In recent years, as the popularity and development of cloud computing technology has grown, information technology (IT) is progressing in the direction of intensification, scale, and complexity [1]. Monitoring systems play a vital role in ensuring the normal operation of large and complex systems.

In the context of cloud computing technology, existing monitoring systems are moving toward large-scale, scalable, and fine-grained directions. Cloud resources are virtual. In addition to monitoring physical resources, monitoring systems also need to monitor virtual resources with a host relationship. Moreover, cloud resources are dynamic scalability. The resources dynamically join and exit based on demands. In addition, cloud computing is a new business computing model. The user uses and pays a certain fee based on demands, which results in the development of granularity of the monitoring systems to be refined to the process [2]. As a result, a series of cloud monitoring research for cloud environments have emerged. In our previous work, we proposed the cloud resource monitoring model B-CMS [3] of the bionic autonomic nervous system (BANS). Brinkmann et al. [4] proposed a highly scalable cloud monitoring system architecture for the "EASI-CLOUDS" project. Ward et al. [5] proposed a cloud monitoring system (Varanus) for large-scale cloud systems that are scalable and easy-to-find. Naik et al. [6] proposed an integrated solution for workload monitoring of hybrid clouds that enables the monitoring of public and private cloud workloads separated by enterprise firewalls. The cloud monitoring system greatly facilitates the management of the cloud system. However, the dynamic changes and massive characteristics of the resources in the cloud environment also bring great challenges to the cloud monitoring system, such as system resource representation,

* Corresponding author.
E-mail address: 541347388@qq.com, cchu@thu.edu.tw

fault monitoring, and status monitoring. Achieving simple resource identification and expression to reduce storage overhead and realizing intelligent state inference, fault judgment mechanism, and reduce system management overhead are the main problems that need to be solved urgently.

In order to solve the above problems of resource representation and fault inference, this paper adds ontology and MDD to the cloud resource monitoring model B-CMS [3], which is based on the author's previous results of bionic autonomic nervous system (BANS). Ontology's standardized data recording method and efficient and convenient semantic derivation function can realize simple resource identification and expression and reduce monitoring data storage overhead. Since ontology portrays the intrinsic connection between concepts, it is easier to perform semantic queries and obtain resource information. Importing ontology technology is beneficial to the unified management of large-scale complex system resources for system administrators, and the management of resources is greatly simplified. The import of MDD technology makes it possible to identify more complex system states through derivation. It also realizes the automatic diagnosis of system status and the timely uploading of alarm information, laying a solid foundation for the further realization of the system's independent monitoring function.

2. A Brief Introduction for B-CMS

BANS is similar to the autonomic nervous system in biology. It controls the bodily functions of animals in an unconscious state. For example, when a person's finger touches hot water, he or she will unconsciously and quickly recover. In addition, an animal's heartbeat, respiratory system, knee reflex, blood pressure, and immune system are controlled by the autonomic nervous system [7].

By simulating the tissue structure of the biological nervous system, BANS gives computer systems similar autonomy. BANS consists of four layers: a neurite shaft, neurons, peripheral nerves, and a central nervous system. The BANS system needs to show its behavior. For example, if a heart rate is abnormal, it can be heard through a stethoscope or an electrocardiogram. Therefore, the BANS-based monitoring system requires a human-computer interaction layer that can display both the monitoring input and the instructions.

Based on BANS, a cloud monitoring system based on BANS (B-CMS) is designed. The system model has a five-layer architecture, as shown in Figure 1. Ontology and MDD techniques are applied to the peripheral nervous system layer to enable data collection and storage and system state judgment.

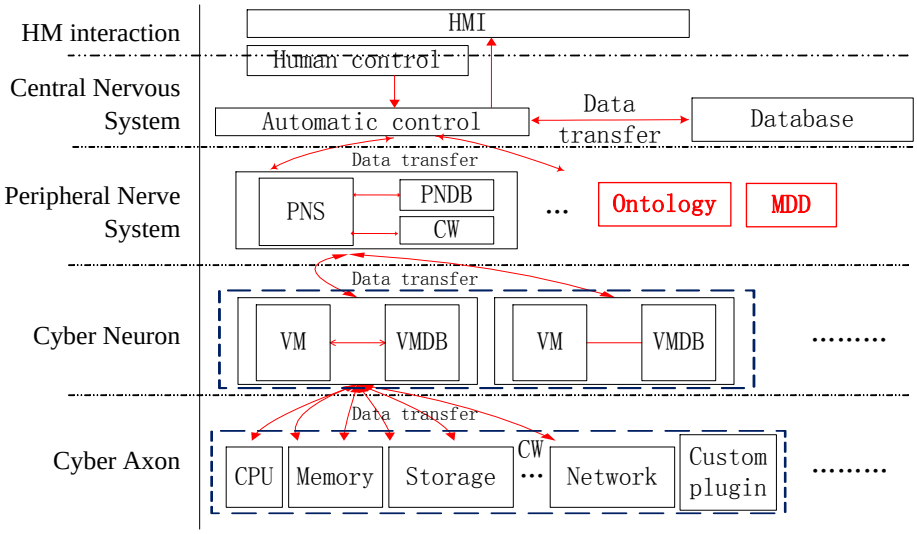

Figure 1. The five-level structure of B-CMS

The cloud resource monitoring model based on the BANS in Figure 1 is described in detail as follows:

1) Human-machine interaction (HMI): The interactive interface is a window for human-computer interaction, similar to the part of the central nervous system that is consciously controlled.

2) Central nervous system (CNS): The monitoring system central is similar to the unconscious control part of the central nervous system, which is responsible for autonomous decision-making and partial communication. Ontology and

MDD technology are applied in this layer, and the system state storage and intelligent determination can be performed after the real-time state of the system is acquired via the peripheral nervous system, neurons, and synapses.

3) Peripheral nervous system (PNS): The PN function module of the host node is similar to the peripheral nerve. Multiple PH modules constitute the peripheral nervous system. Each peripheral nerve PH runs a data collector (CW) to collect the monitoring data of the physical node and report the historical data to the monitoring system hub. The collecting data are stored in the peripheral nerve database (PNDB). The design of this layer is based on the idea of the divide, and this layer is in charge of the re-diagnosis processing of the system failures. However, the final decision is still made by the hub.

4) Cyber neuron: The virtual machine (VM) hosted on a physical node is similar to a neuron. The VM owns the functions of data transmission and basic data analysis. Each neuron VM belongs to a certain peripheral nerve. The VM collects monitoring data through the lower neurite axis and is responsible for completing interactive communication with the peripheral nerve and analyzing, processing, and summarizing the collected information. The VM has a certain local storage function (VMDB) and only reports historical summaries or abnormal feedback. The character of the VM greatly reduces the network communication load.

5) Cyber axon: The main function of the cyber axon is perception and reflection. It is responsible for collecting monitoring information and executing commands issued from the upper layer. The data collector cyber axon is similar to the neurite axis. The cyber axon contains various plug-ins, each of which performs certain collection tasks such as CPU, Memory, Disk, and NetIO. The cyber axon collects the monitoring information. At the same time, it can receive the instructions transmitted from the upper layer and perform corresponding actions, such as changing the collection frequency or the warning threshold.

3. The Description and Application of Ontology

3.1. The Description of Ontology

Ontology is a kind of information expression that consists of four major components (entities, classes, relationships, and methods). Entities represent the concepts described. Classes represent collections of entities. Relationships descript the relationships between entities, including subordinates and inclusions. Methods represent a way to derive another entity or class from an entity or class. Creating a new ontology includes the following process [8-9]: 1) define the classes in ontology (these classes are the abstract descriptions of domain knowledge). 2) Use the subclass-superclass inheritance relationship to organize the taxonomy of these classes. 3) Define the property for the class and describe the constraints of the property (including the type of the value, the range of values, and the number of occurrences) [5].

According to the definition given by [8], "ontology is a clear formal specification of the shared conceptual model". The core of ontology is the relationship between concepts. In traditional ontology, the event class EC (event class) is usually only treated as a special kind of concept. It is difficult to express the nature of the event, and it is also difficult to describe the complex semantic relationship between the event classes. Event ontology adds a description of event classes and their relationships on the basis of traditional ontology.

3.2. The Definition of Ontology

3.2.1. The Definition of an Event and Its Event Class

Definition 1 (Event) is a thing that occurs at a particular time and place, is attended by some characters, and exhibits some action characteristics. It formally uses e to represent the event. The event consists of six elements, which can be represented by a six-tuple, as shown in Equation (1).

$$e ::=< A, O, T, P, S, L > \tag{1}$$

Where A represents the action element and corresponds to the trigger word in the text. It is noteworthy that the action elements (trigger words) in the event are the significant words indicating the occurrence of the event. In addition, the object elements in the event are divided into subject and object. The subject is the action of the event action, and the object is the action of the action.

Definition 2 (Event class) is a collection of events with common characteristics.

$$EC = (E, E_A, E_O, E_T, E_P, E_S, E_L)$$

$$E_i = (e_{i1}, e_{i2}, \cdots, e_{im}, \cdots),\ i \in \{A, O, T, P, S, L\},\ m \geq 0$$

Where E is the set of events, E_i is the set of common features that each event in E has on the i^{th} feature, and e_{im} is a common attribute that each event in E has on the i^{th} feature.

3.2.2. Event (*Class*) *Relationship*

Definition 3 (Hierarchy of event class) event class $EC_1 = \{E_1, E_{1A}, E_{1O}, E_{1T}, E_{1P}, E_{1S}, E_{1L}\}$ and event class $EC_2 = \{E_2, E_{2A}, E_{2O}, E_{2T}, E_{2P}, E_{2S}, E_{2L}\}$. There is a classification relationship between EC_1 and EC_2 if and only if ($E_1 \subset E_2$ or $E_1 = E_2$ and $E_{1j} \subset E_{1j}(j \in \{A, O, T, V, P, L\})$) Then, EC_1 is called the lower event class of EC_2, and EC_2 is called the upper event class of EC_1. This classification relationship is represented by $R_{is-a}(EC_1, EC_2)$.

Definition 4 (Non-hierarchy of event class) (1) Composition relationship: when the event class EC_1 is composed of the event class EC_2, it is said that the two event classes have a composition relationship. (2) Causal relationship: if the occurrence of the event class EC_1 causes the occurrence of the event class EC_2, the two event classes have a causal relationship. (3) Follow-up relationship: in a certain time interval, the occurrence of the event class EC_1 follows the occurrence of the event class EC_2, which is called a follow-up relationship. (4) Concurrency relationship: if the event class EC_2 occurs at the same time as the event class EC_1 within a certain time range, then the two event classes have a concurrency relationship.

3.2.3. The Structure of Event Ontology

Definition 5 (Event ontology) is an event class model that shares objective existence, which can be defined as a five-tuple: $EO =<$ UECS, LECS, R, Rules, Individuals $>$. Among them, (1) UECS (upper event class set) is a hierarchical structure. (2) LECS (lower event class set) is an event class whose lower event class set is a lattice structure. (3) $R = \{r \mid r - (Ris - a$, Rcause , RcomposedOf , Rconcur, Rfollow) }. r is the relationship between the event (class) and the event (class). (4) Rules are some inference rules, including layer event class classification inference rules and event relationship inference rules. (5) An instance collection of the individuals event class.

3.2.4. The Advantages of the Ontology Model

The event class in the model not only contains related knowledge of objects, actions, time, places, etc., but also covers dynamic feature knowledge such as state assertions and language expressions. It can describe the relationship between event classes, such as composition relationships, following relationships, concurrent relationships, and causal relationships. Therefore, the event ontology centered on the "event class" can effectively express the objects, time, places, actions, and some complex relationships between the event class and the event class. It is a good knowledge representation unit.

3.3. The OWL Description of Ontology in B-CMS

In the monitoring system, ontology is first used to describe the concepts in the system, such as servers, virtual machines, hard disks, and memory, and then establish the ontology description of each system resource and their attribute descriptions.

Figure 2 shows an entity class of the server's attribute composition. Its basic attributes include server fault information, basic information, and status information.

After defining the ontology description of the server, the system can assign a value to the ontology entity of the server through simple calculation and generate file storage server information in XML format for each time the system obtains the server information. Taking the server entity as an example, the ontology description format of the entity and the storage assignment method are given. In addition, the system also includes virtual machines, memory, CPU, hard disk, and many other entities. They also can be described through the ontology description. After using the ontology description, the system has a unified resource description method. At the same time, existing entity descriptions can also be uploaded to the network for reuse by other new systems or related personnel without re-description of resources. This will greatly reduce system deployment and expansion overhead.

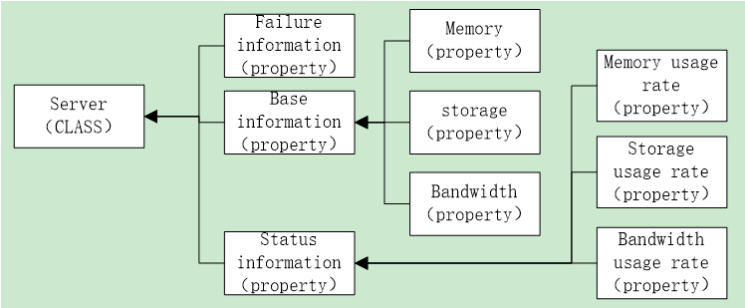

Figure 2. The property of the server entity

3.4. OWL Semantic Query based on Jena

OWL can infer or recognize the concepts described by ontology [10]. OWL is a semantic derivation method, which provides a "machine-readable" specification language by adding vocabulary and a complete set of semantics for XML, RDF, and RDF Schema [11]. By using OWL semantic derivation, resource acquisition or resource status queries can be performed in B-CMS in the form of natural language.

In the following, we give an example of the realization of OWL semantic query based on the third-party tool Jena. It should be pointed out that Jena runs in the Java environment, and we do not describe the details of the deployment of Jena. The example implements a query of 'find all virtual machines with less than 50% memory usage'. We directly give Jena's basic language format - SPARQL statement example.

```
PREFIX info <http://somewhere/VM#>
SELECT? resource
WHERE { ?resource info: VM?VM.
FILTER (?VM. memory capacity < 0.5)};
```

In the example, the red font is the core of the query structure. '?resource' represents a variable, the 'where' statement is used to return the VMs that meet the condition of the statement, and the 'FILTER' statement is a constraint that is used to find the VMs when the memory capacity is less than 50%.

This example shows that by using the third-party tool of OWL to analyze the system described by ontology, the system administrator can conveniently obtain resource information through a semantic query in the form of the database query statement. Therefore, the application of ontology increases the convenience of resource management.

4. The Description and Application of MDD

In the face of multi-state systems (systems with multiple complex states), we introduce the multivariate decision diagram (MDD) multi-value decision graph technique [12] according to ontology-based system resource description information in order to express the state of different situations of multi-state systems more quickly and accurately. The application of this technology allows for the identification of more complex system states through derivation.

When the state of the system is more than two, we call the system a polymorphic system. If each component in the system also has multiple states, we call the system a multi-state system with multi-state components, which is referred to as a polymorphic system. The performance, capacity, or state reliability level of a polymorphic system can be expressed by the state of the system.

The state of a polymorphic system depends only on the state of the system components. A system with n components can be represented as a mapping of a multi-valued function $f(x_1, x_1, \cdots, x_n)$: $R_1 \times R_2 \times \ldots \times R_n \to M$, where each x_i represents a tuple of the R_i state and $R_i = \{0,1,\cdots,r_{i-1}\}$ is the state set of the component. $M = \{0,1,\cdots,m-1\}$ represents the state set of the system. At this time, the multivalued function is called the structural function of the polymorphic system. In many applications, the state of a system and its components are completely ordered, and the operating state of the system component will affect the operating state of the entire system. Therefore, by assigning values to each state in ascending order, the structure function usually becomes a monotonically increasing function. That is, the multivalued function $f(x_1, x_2, \cdots, x_n)$ is a monotonically increasing function for any x_i, i.e.,

$$f(x_1, x_1, \cdots, x_{i-1}, \alpha, x_{i+1}, \cdots, x_n) \leq f(x_1, x_1, \cdots, x_{i-1}, \beta, x_{i+1}, \cdots, x_n)$$

$$\alpha, \beta \in R_i, \text{ and } \alpha \leq \beta$$

It can be defined that x_i equals 0, which represents that the worst state of the component is the complete failure of the component. When x_i equals r_{i-1}, the best state of the component is that the component is completely faultless, when the value of f is 0. The worst state of the system is the complete failure of the system. When the value of f equals 0, the optimal state of the system is that the system is completely faultless. As the value of the component state increases, the state value of the system also increases. The fewer failures of a component, the fewer failures of the system, and the better the operating state.

Multiple-valued decision diagrams (MDDs) are directed acyclic graphs that reflect the relationship between component states and system states. The acquisition of MDD is similar to the acquisition of BDD, which is obtained by applying multivalued functions repeatedly through Shannon's expansion theorem. The individual components of MDD represent the sub-functions of f obtained by assigning values to certain components, with each internal node having multiple output edges corresponding to the component values.

$$f_0(x_1, x_2, x_3) = x_1 + x_2 + 2x_3,$$
$$x_1(0,1,2), x_2(0,1,2), x_3(0,1).$$
$$f = \begin{cases} 0, f_0 = 0 \\ 1, f_0 \in [1,2] \\ 2, f_0 \in [3,4] \\ 3, f_0 \in [5,6] \end{cases}$$

The results obtained by the above multi-valued function f are shown in Table 1, and Figure 3 is its decision diagram. The end nodes 0, 1, 2, and 3 in the figure represent the values of the multi-valued function f.

Table 1. The results of the multi-valued function f

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
0	2	0	1
0	2	1	2
…	…	…	…
2	2	1	3

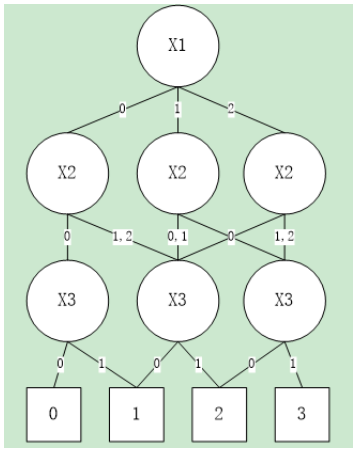

Figure 3. The MDD of the multi-valued function f

We define the concept of status rating to determine the current state of the system. The status rating is a dynamic

attribute that evaluates the state of the entire system based on the operational status of various entity components of the system, such as servers, virtual machines, networks, and buses. Here, we divide the state into three types: normal, good, and faulty. MDD performs system state derivation by establishing a conditional decision tree. According to the input state information, a node is triggered when all of its leaf nodes are triggered. The multi-valued decision graph is an extension of the binary decision graph. It is a directed acyclic graph, which mainly includes three main parts, namely branches, leaf nodes, and non-leaf nodes. Non-leaf nodes represent a basic module in the system, containing values from 0 to *m*. Each of them contains m branches, which point to the final state. The leaf node is the terminal node of all branches, and it represents the state of the final running result of the system.

For example, there is a server that contains three CPUs, two memories, and a bus. CPUs are represented by P1, P2, and P3; memories are represented by M1 and M2; and a bus is represented by B. The state of the server is determined by these components. The state of the system is divided into three levels, normal, good, and faulty, which are represented by 0, 1, and 2, respectively. P1 is normal, P2 is good, P3 is good, M1 is good, M2 is good, and B is faulty. The MDD structure and inference results are shown in Figure 4. The judgment process is indicated by the dotted line, and the detailed construction process [13] is not repeated here. By judgment, the system status is a fault (2).

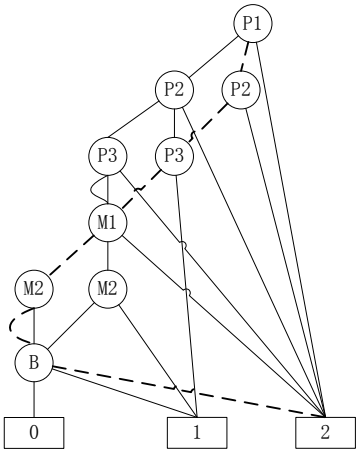

Figure 4. The tree structure of MDD

According to the construction rules of MDD, if the bus fails, the entire system will not normally operate regardless of the status of the other components. Therefore, the current overall server status is also determined to be a failure. In the following, we further show the example of the judgment tree structure when the system status rating is good after adding MDD technology according to the system composition of the monitoring system.

In Figure 5, the conditional judgment tree of the status rating attribute is shown. The tree has three layers, and the lowermost layer is the leaf node (input system state). After receiving the real-time data of the system, the value of the second-layer node is determined, which means that the operational status of the component is determined. The process then enters the MDD decision process. Ultimately, a system failure rating is obtained.

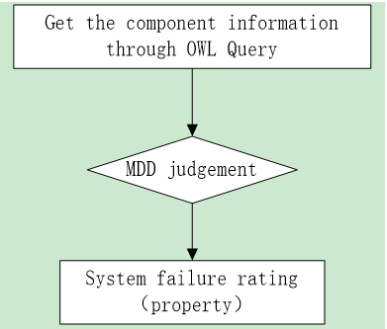

Figure 5. The conditional judgment tree of the status rating attribute

The introduction of ontology and MDD technology makes the monitoring system based on further autonomy and intelligence in the face of system complex state determination, fault detection, and timely reporting of faults. The

introduction of this kind of technology will greatly reduce the workload of administrators in system fault detection so that administrators can focus on the solution of faults.

5. Experiment

In the experimental part, we use three servers to realize the function of the B-CMS bionic autonomic nervous system. Seven desktops are used to realize the functions of the peripheral nervous system. Four virtual machines running on each desktop are used to implement the functions of the neuron. The configuration of the machines is shown in Table 2.

Table 2. The configuration of the machines

Configuration	Server	Desktop
CPU	Intel Xeon 5504	Intel T6600
Memory	4G	2G DDR3
Storage	1000G SAS	500G SATA2
Operation system	Centos 6.7	Ubuntu 12.10

In order to verify the effectiveness of the fault diagnosis function of the cloud resource monitoring system B-CMS based on ontology and MDD that is proposed in this paper, this experiment generates a variety of system fault states with different severities through fault injection. The simulated fault types are shown in Table 3.

Table 3. The different type of failures

Failure type	The way of injecting failure
Usability failure	High priority compute-intensive application memory leak Continuous storage peripheral read and write
Reliability failure	ARP attack Frequent network cable plugging Application piling
Anti-hazard failure	Modify the collected status data
Security failure	Dense port scanning Root steeling Illegal connection

In the experiment, the system will collect the running state data of each resource in real-time, and the data will be described and stored in the manner of ontology. The system will periodically call the OWL language to obtain the resource running data from the ontology description and use MDD for fault diagnosis. We set the leaf node of MDD to 4, and the state of the system resources is defined as four levels, which are minor, intermediate, advanced, and severe.

In total, we input 4,500 instance data, where there are 1,500 minor-level faults, 1,500 intermediate-level faults, and 1,500 high-level faults. It should be noted that for the instance data with severe fault levels, the system cannot obtain the status information of the resource, because the system crashes during the monitoring. Therefore, we do not set the fault level to be serious.

Figure 6 shows the diagnostic results of MDD after all instance data has been entered. Figure 7 shows the average time spent to diagnose each fault type.

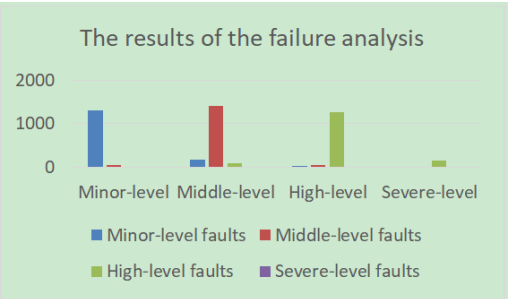

Figure 6. The results of the failure analysis

The results in Figure 6 show that the addition of MDD better implements the fault diagnosis function. However, when the diagnosis of the fault level is mild, intermediate, or advanced, there is still a certain deviation of the diagnosis result. For mild,

intermediate, and advanced symptom inputs, the results were misdiagnosed by 12.7%, 5.65%, and 15.45%, respectively. In the experiment, when the multi-valued decision graph in the peripheral nerve is making a judgment for the fail-safe level, the fault level is directly defined for the fault with the clearly defined path. For a fault that has not been defined, the probability that the fault is in each of the four faults will be first listed, and then the probability of all the route edges will be added. Finally, the leaf node with the largest value will be used as the judgment result.

The results in Figure 7 show that the higher the fault level, the longer the diagnostic time. By analyzing the amount of the back-end data transmission, the analysis of high-level faults requires more resource state data to support the accuracy of the resulting judgment, so more OWL queries are generated. Therefore, high-level fault queries are more time-consuming. It should be pointed out that 1) the diagnosis time is confirmed by the time elapsed between inputting the fault instance data to the diagnosis result, and 2) in order to record the time spent on the diagnosis as accurately as possible, we define that the status information of the system resources is recorded after the system runs a set of instance data.

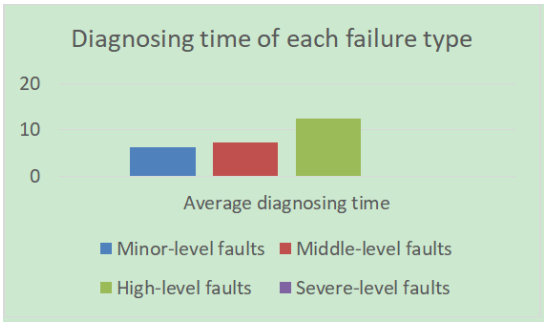

Figure 7. The average diagnosing time of each failure type

6. Conclusions

The ability of monitoring is needed for large and complex systems, and monitoring systems are especially important for cloud platforms. The cloud resource monitoring model based on the BANS structure fully considers the dynamics of cloud resources and gives the monitor system the ability to realize the autonomy of physical machines and virtual machines. Experiments show that after adding ontology and MDD technology, the monitoring model has an intelligent fault detection function, which can diagnose faults in a timely manner, further realizing the timely reporting of faults and demonstrating the use-value of the system in the engineering field. In the future, we can divide different peripheral nervous systems according to different monitoring characteristics, perform task-oriented dynamic regional monitoring, further analyze historical data, realize more independent decision-making mechanisms, and further expand the model.

Acknowledgements

This work is supported by the National Science Foundation of China (No. 61602094) and Fundamental Research Funds for the Central Universities (No. 2672018ZYGX2018J052)

References

1. G. Qiang, "Review of the Development of Cloud Computing in China," *Information Technology*, Vol. 7, pp. 1-4, 2013
2. G. J. Wei, Z. Bo, and F. Y. Qiu, "Study on Resource Monitoring Model in Cloud Environment," *Computer Engineering*, Vol. 37, No. 11, pp. 31-33, 2011
3. P. Sun, H. Xu, and C. J. Jing, "Design and Implementation of Cloud Resource Monitoring System based on BIONICS," *Journal of Computer Applications*, Vol. 36, No. 7, pp. 2051-2055, 2016
4. A. Brinkmann, C. Fiehe, and A. Litvina, "Scalable Monitoring System for Clouds," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pp. 351-356, 2013
5. J. S. Ward and A. Barker, "Varanus: In Situ Monitoring for Large Scale Cloud Systems," in *Proceedings of IEEE 5th International Conference on Cloud Computing Technology and Science* (*CloudCom*), pp. 341-344, 2013
6. V. K. Naik, K. Beaty, and N. Vogl, "Workload Monitoring in Hybrid Clouds," in *Proceedings of IEEE 6th International Conference on Cloud Computing* (*CLOUD*), pp. 816-822, 2013
7. M. Hinchey, Y. S. Dai, and J. L. Rash, "Bionic Autonomic Nervous System and Self-Healing for NASA ANTS-Like Missions," in *Proceedings of the 2007 ACM Symposium on Applied Computing*, pp. 90-96, 2007
8. S. Maere and H. M. Kuiper, "BiNGO: A Cytoscape Plugin to Assess Overrepresentation of Gene Ontology Categories in Biological Networks," *Bioinformatics*, Vol. 21, No. 16, pp. 3448-3449, 2005

9. J. Bhogal, A. Macfarlane, and P. Smith, "A Review of Ontology-based Query Expansion," *Information Processing and Management*, Vol. 43, No. 4, pp. 866-886, 2013
10. M. Dean, D. Connolly, and F. V. Harmelen, "OWL Web Ontology Language 1.0 Reference," *Anaesthesia*, Vol. 59, No. 7, pp. 729-729, 2004
11. D. Martin, M. Burstein, and J. Hobbs, "OWL-S: Semantic Markup for Web Services," *W3C Member Submission*, Vol. 22, No. 4, 2004
12. Y. Dai, Y. Xiang, and Y. Li, "Consequence Oriented Self-Healing and Autonomous Diagnosis for Highly Reliable Systems and Software," *IEEE Transactions on Reliability*, Vol. 60, No. 2, pp. 369-380, 2011
13. S. Nagayama, T. Sasao, and J. T. Butler, "Analysis Methods of Multi-State Systems Partially Having Dependent Components using Multiple-Valued Decision Diagrams," in *Proceedings of IEEE 44th International Symposium on Multiple-Valued Logic*, pp. 190-195, 2014