J. Shanghai Jiaotong Univ. (Sci.), 2015, 20(2): 149-155
DOI: 10.1007/s12204-015-1603-1

Application-Oriented Cloud Monitoring Data Distribution Mechanism

LI Da-zhi[1,2*] (李大志), *LIU Jian-hua*[3] (刘建华), *DONG Xin*[1] (董　鑫)
LI Lu-qun[2] (李鲁群), *CHEN Jun-hua*[2] (陈军华)
(1. School of Electronic Information and Electric Engineering, Shanghai Jiaotong University, Shanghai 200240, China;
2. College of Information, Mechamical and Electrical Engineering, Shanghai Normal University, Shanghai 200234, China;
3. College of Mathematics, Physics and Information Engineering, Jiaxing University, Jiaxing 314001, Zhejiang, China)

Abstract: Cloud computing system consists of private clouds and public clouds. It merges its resources on each layer (e.g. IaaS, PaaS and SaaS), which poses a challenge for resource management. The cloud monitoring system is a solution to managing cloud system data from the heterogeneous resources. This paper discusses the monitoring and collection of the heterogeneous resources, studies the adaptive system, and proposes a real-time extensible distributed framework of monitoring data processing. Based on this framework, a system of monitoring data distribution, publication and subscription is proposed. The simulation results show that the proposed mechanism can adaptively determine the distribution action of monitoring data flow, and effectively reduce the costs for data monitoring and distribution.

Key words: cloud computing, cloud monitoring, data aggregation, resource management
CLC number: TP 399　　**Document code:** A

0 Introduction

Cloud computing is a computing grid. It can provide IT-related services for customers flexibly. It includes three layers: IaaS, PaaS and SaaS. IaaS layer converts physical resources into virtual resources by visual machine for the use of PaaS layer. PaaS layer uses the resources provided by IaaS to achieve a variety of middleware services (e.g. data security and data processing). SaaS application software provides services for consumers. The consumers are able to be accessible to unlimited computing capabilities by Internet with a terminal device that has limited computing and storage capacity. The current terminal device can access the following deployed cloud applications: video-on-demand (VOD), the Internet TV, file sharing, etc. In order to make these cloud applications operate in a more reliable and efficient way and meet users' satisfaction, we should further improve the service of cloud applications through the monitoring data analysis from the network, CPU and memory conditions in cloud systems. However, the three-layer structure of cloud computing system is not designed to monitor the physical resources and virtual resources to improve the service. The existing distributed monitoring technology is designed based on C/S mode. When a large amount of data come to the center node, data processing will be blocked. Moreover, the monitoring data distribution is tightly coupled with various monitoring data communication formats. This paper mainly makes the following contributions: the establishment of a distributed monitoring framework to improve the collection and distribution of the monitoring data, the design of a distributed algorithm in order to reduce the amount of data distribution in the cloud computing system, and the establishment of a prototype system to evaluate the algorithm.

1 Related Works

Efficient resource monitoring is an emerging subject in the field of cloud computing. It is divided into network-oriented monitoring and service-oriented monitoring in view of the current distributed systems. Ganglia is an open source monitoring system. It operates on high performance computers and grid environment, with its monitoring program running on each node in the cluster using multicast transmission protocol to send cluster status information to other cluster nodes[1]. Nagios is an open source monitoring tool for network

Received date: 2014-11-07
Foundation item: the Scientific Research Foundation of Zhejiang Provincial Education Department of China (No. Y201431192)
∗**E-mail:** lijunzhi@sjtu.edu.cn

system. It uses plug-in technology to obtain the monitoring data and uses a set of distributed servers to configure the distributed monitoring[2]. In Ref. [3], a large-scale integrated service-oriented architecture was proposed, and monitoring agents could carry out various monitoring tasks in cooperation with other agents. In Ref. [4], Kanstrén and Savola defined a distributed monitoring framework which includes scalability, accuracy, security and adaptability. In Ref. [5], the authors adopted a service-oriented concept and used information exchange technology to integrate the existing system. In Ref. [6], a framework was proposed to evaluate the index. Amazon CloudWatch, based on the Amazon Web service MaaS, obtains the monitoring data from server nodes[7]. In a cloud computing system, a large number of log files are stored on the server nodes. Apache flume is a Hadoop distributed file system (HDFS) to collect and store large amounts of log data based on the flow processing. Each of its components executes corresponding flow processing[8].

Compared with the above methods, our method focuses on the multicast distributed processing in hybrid cloud system. This paper identifies the extensibility and loosely-coupled cloud monitoring "publish/subscribe" framework to improve the efficiency of data distribution.

2 Loose Coupling Cloud Monitoring Framework

The hybrid cloud monitoring service needs multiple cloud providers. Each cloud provider controls its own infrastructure, and provides a different virtual infrastructure control mechanism. Therefore, the monitoring service must receive monitoring data from multiple heterogeneous data sources. At the beginning, cloud monitoring service can monitor small cluster systems, but later the increase of servers forms a large system, so that a hybrid cloud monitoring framework must have good scalability. The purpose of monitoring is to make the cloud system run reliably. Any monitoring data should not be lost in order to achieve reliable data distribution and node status updates. This paper designs a monitoring service framework based on the above consideration to meet the needs of cloud monitoring infrastructure. The framework is deployed on the cloud computing platform. Loosely-coupled cloud monitoring framework is shown in Fig. 1.

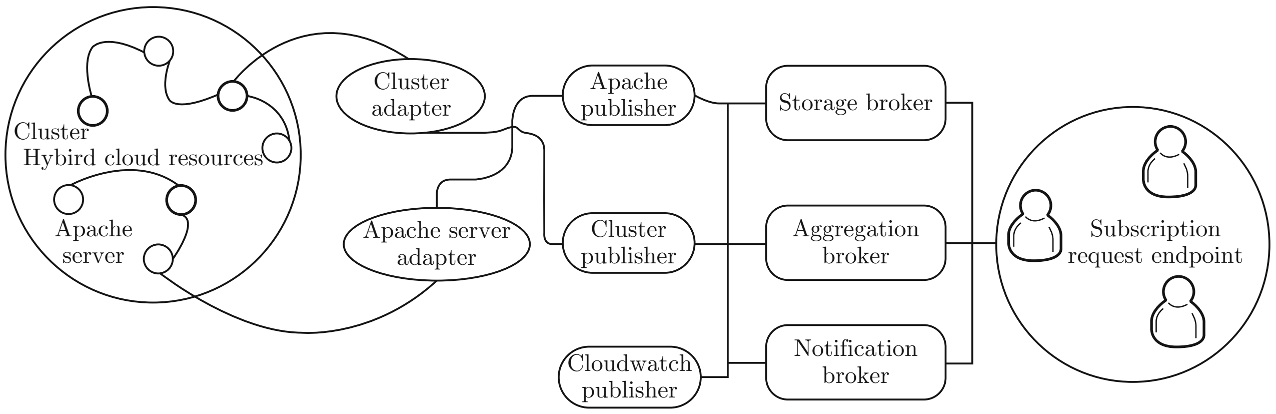

Fig. 1 Loosely-coupled cloud monitoring framework

Data flow serves as a basic unit of the transmission among modules in a "publish/subscribe" model. A broker, as a medium of data transmission, is responsible for receiving and sending the monitoring data flow, as well as receiving customers' requests. Apache adapter is installed on the Apache server. It transmits the original log to the format converter, then to the Apache publisher, and finally transmits the data flow to the broker. Cluster adapter is installed on the cluster, and the cluster publisher transmits the original flow to the broker. Cloudwatch publisher provides a configuration document, listing the examples of monitoring, the measurement attributes to be collected and the frequency of collection. Subscription request endpoint requests the monitoring data. Storage broker monitors the upcoming data flow and saves it in HBase database.

3 The Design of Monitoring Algorithm

3.1 A Distributed Monitoring Method

Loose coupling "publish/subscribe" mechanism sends subscription data flow only to the broker, but does not across the broker to other adapters. Because no subscription data flow is produced from the adapter to the broker, the number of the messages is reduced. But how to further reduce the number of messages from the adapter to the broker remains a great challenge. In Ref. [9], the method of time window was utilized to consider local and global monitoring data polling, and the parameters were optimized by adjusting the parameters of the monitoring, but the dynamic adjustment of local and global monitoring parameters on monitoring nodes was not considered. In Ref. [10], hybrid update threshold and dynamic time interval adjustment method were adopted, but the status updates of threshold and time intervals on distributed monitoring nodes were not considered locally and globally. This paper considers the above two points and studies dynamic time-interval regulating mechanism, namely dynamic announcing with local change and global time consideration (DALCGTC).

3.2 The Reduction of Distributed Monitoring Task Messages

In a hybrid cloud system, monitoring messages increase as the growth of the number of nodes, so that consumption of bandwidth and CPU cycles from the adapter to the broker increase. In fact, the Apache publisher does not have to send a notice of each change of local state that Apache adapter receives to the cluster publisher, so the data flow updates of each change are avoided. However, because the resource state changes are recorded locally in a given time interval, the real-time updates of state change cannot be transmitted to the broker. One solution is to adjust the time window[10]. Based on DALCGTC monitoring task, this paper mainly considers the following aspects: ① the monitoring nodes related with monitoring task which include the Apache publisher sets and cluster publisher sets; ② the dynamic thresholds of global and local resources changes.

3.3 Adaptive Distributed Monitoring Data Flow Processing Strategy

The resource threshold value of local nodes depends on the degree of changes in number that the Apache adapter obtains. Here, N_a denotes the number of messages about the state change of resources that Apache adapter sends to Apache publisher; C_i denotes the value of changing resources; R denotes the set of resources, $R = \{r_1, r_2, \cdots, r_j\}$, $i < j$; ΔR denotes the increment of the resource states, $\Delta R = \{\Delta r_1, \Delta r_2, \cdots, \Delta r_k\}$, $k < j$. The dynamic resource threshold value of the local node Apache adapter is

$$\theta = \frac{1}{N_a}\sum_{i=1}^{m}\left(\Delta r_i^{\text{Cpu}} + \Delta r_i^{\text{Bw}} + \Delta r_i^{\text{Mem}}\right),$$

where Δr_i^{Cpu}, Δr_i^{Bw} and Δr_i^{Mem} are CPU's increment of the resource states, bandwidth's increment of the resource states and memory's increment of the resource states in Apache adapter i, respectively; m is the number of Apache adapter. When the Apache adapter monitors the resource change threshold value $r_i(t)$ on a node i at time t and $r_i(t) > \theta$, then the data flow of resource information is sent to the Apache publisher.

The data flow from the cluster adapter to the adapter publisher includes two parts. One is the data flow from all local nodes' Apache publishers, and the other is the data flow from virtual machines (VMs) in the cluster server. Let N_b be the number of messages about state change of resources that the cluster adapter sends to the cluster publisher.

The dynamic resource change threshold value of each cluster adapter on VMs is

$$\lambda = \frac{1}{N_b}\sum_{j=1}^{q}\left(\Delta r_j^{\text{Cpu}} + \Delta r_j^{\text{Bw}} + \Delta r_j^{\text{Mem}}\right),$$

where Δr_j^{Cpu}, Δr_j^{Bw} and Δr_j^{Mem} are CPU's increment of the resource states, bandwidth's increment of the resource states and memory's increment of the resource states in cluster adapter j, respectively; q is the number of cluster adapter. The dynamic resource change threshold value of the cluster adapter on global nodes is

$$\pi = \frac{1}{N_{\text{v}}}\sum_{l=1}^{N_{\text{v}}}\lambda_l,$$

where λ_l corresponds to the dynamic resource change threshold value of each cluster adapter on the lth VM, and N_{v} is the number of VMs. Let $r_i^{\text{v}}(t)$ be resource changes' total value that is observed by cluster adapter i on VM v at time t. Let $r_i^{\text{s}}(t)$ be resource changes' total value that is observed by the cluster adapter i on the server s at time t. If $r_i^{\text{v}}(t) > \lambda$, then the data flow of resource information is sent to cluster publisher. If $r_i^{\text{s}}(t) > \pi$, then the data flow of resource information is sent to the broker. If the cluster adapter monitors the state changes of the resources on a node, then the resource information data flow processing event is e_1, and its processing actions include save (a_1), notice (a_2) and reject (a_3). The option of actions depends on the state of CPU resources, memory resources and bandwidth resources.

Definition 1 The cluster adapter monitors the change of resource state on a node, and the set of events is

$$E = \{e_1, e_2, \cdots, e_{N_{\text{v}}}\}.$$

Accordingly, the possible set of processing actions is $A = \{a_1, a_2, a_3\}$.

Definition 2 The event set of the cluster adapter is

$$E = \{e_1, e_2, \cdots, e_{N_{\text{v}}}\}.$$

The action set is

$$\begin{aligned}&A = \{a_1, a_2, a_3\},\\&E \times A = \{(e_i, a_j) | e_i \in E, a_i \in A\}.\end{aligned}$$

Denote $s_{ij} = (e_i, a_j)$, where s_{ij} is the processing strategy of data flow.

Definition 3 Let $k = (r^{\text{Cpu}}, r^{\text{Men}}, r^{\text{Bw}})$ be the property of resources, where r^{Cpu}, r^{Men} and r^{Bw} denote the properties of CPU, memory and bandwidth resources, respectively. Let $o_{ij}^{(k)}$ be the action effect of s_{ij} in k. Let $r_1^{\rm Cpu}$ and $r_2^{\rm Cpu}$ be the critical values of resource state changes of $r^{\rm Cpu}$. Let $r_1^{\rm Mem}$ and $r_2^{\rm Mem}$ be the critical values of resource state changes of $r^{\rm Mem}$. Let $r_1^{\rm Bw}$ and $r_2^{\rm Bw}$ be the critical values of resource state changes of $r^{\rm Bw}$. The three-variable decision set of resource state changes of cluster adapter is given by

$$S^{(3)} = \{(r^{\rm Cpu}, r^{\rm Men}, r^{\rm Bw}) | r_1^{\rm Cpu} \leqslant r^{\rm Cpu} \leqslant r_2^{\rm Cpu}, \\ r_1^{\rm Mem} \leqslant r^{\rm Mem} \leqslant r_2^{\rm Mem}, r_1^{\rm Bw} \leqslant r^{\rm Bw} \leqslant r_2^{\rm Bw}\}.$$

Let $o_{ij}^{\rm Cpu}$, $o_{ij}^{\rm Mem}$ and $o_{ij}^{\rm Bw}$ be the action effects of s_{ij} for the properties of CPU, memory and bandwidth resources in k, respectively. If the action effect is o_{ij} and $o_{ij} = (o_{ij}^{\rm Cpu}, o_{ij}^{\rm Mem}, o_{ij}^{\rm Bw}) \in S^{(3)}$ under data flow processing strategy s_{ij}, then data flow processing strategy s_{ij} can get desirable resource status value in the constraints of CPU resources, memory resources and bandwidth resources, and a_j is the desirable action of events e_j of the cluster adapter under the resource state constraints.

Definition 4 If $r_i^{\rm v}(t) > \lambda$, then the cluster adapter sends out the data flow. Set $\lambda = \rho$, then

$$R^{(N_{\rm v})} = \{[r_1^{\rm v}(t), r_2^{\rm v}(t), \cdots, r_{N_{\rm v}}^{\rm v}(t)]\ |(r_1^{\rm v}(t) - \rho_0^{(1)})^2 + \\ (r_2^{\rm v}(t) - \rho_0^{(2)})^2 + \cdots + (r_{N_{\rm v}}^{\rm v}(t) - \rho_0^{(N_{\rm v})})^2 \leqslant \alpha^2\},$$

where $\rho_0 = \{\rho_0^{(1)}, \rho_0^{(2)}, \cdots, \rho_0^{(N_{\rm v})}\}$, $R^{(N_{\rm v})}$ is the threshold value of resources as the center, and α is the radius of the sphere. Then ρ_0 is the optimal threshold of resource change on each VM.

Definition 5 Let $|\rho_1 - \rho_0| = [(\rho_1^{(1)} - \rho_0^{(1)})^2 + (\rho_1^{(2)} - \rho_0^{(2)})^2 + \cdots + (\rho_1^{(N_{\rm v})} - \rho_0^{(N_{\rm v})})^2]^{1/2}$, where $\rho_1 = \{\rho_1^{(1)}, \rho_1^{(2)}, \cdots, \rho_1^{(N_{\rm v})}\} \in R^{(N_{\rm v})}$. Here, $|\rho_1 - \rho_0|$ is the center distance of resource variable ρ_1, and it reflects the superiority of action effect on the incident response vector on the cluster adapter.

Definition 6 Let s_{mn} be the different processing strategies of data flows. Let o_{mn} be the effect of actions of processing strategy s_{mn}, and

$$o_{mn} = (o_{mn}^{\rm Cpu}, o_{mn}^{\rm Men}, o_{mn}^{\rm Bw}),$$

where $o_{mn}^{\rm Cpu}$, $o_{mn}^{\rm Mem}$ and $o_{mn}^{\rm Bw}$ are the action effects of s_{mn} for the property of CPU, memory and bandwidth resources in k, respectively. If $|o_{ij} - \rho_0| \geqslant |o_{mn} - \rho_0|$, then s_{mn} is better than s_{ij}, denoted by $s_{mn} \succ s_{ij}$. When the equation is true in the formula, then s_{mn} is equivalent of s_{ij}, denoted by $s_{mn} \cong s_{ij}$. Let OP be the data flow processing strategy under the constraints of CPU resources, memory resources and bandwidth resources. Algorithm 1 describes the adaptive monitoring data flow decision on the cluster adapter.

Algorithm 1 Adaptive monitoring data flow decision on the cluster adapter

```
(1) Creating set monitoring data flow processing
    $s_{11} = (e_1, a_1)$,
    $s_{12} = (e_1, a_2)$,
    $s_{13} = (e_1, a_3)$
(2) Computing action effect
    $o_{11} = [o_{11}^{\rm Cpu}, o_{11}^{\rm Mem}, o_{11}^{\rm Bw}]$,
    $o_{12} = [o_{12}^{\rm Cpu}, o_{12}^{\rm Mem}, o_{12}^{\rm Bw}]$,
    $o_{13} = [o_{13}^{\rm Cpu}, o_{13}^{\rm Mem}, o_{13}^{\rm Bw}]$
(3) Computing $|o_{1j} - \rho_0|$ according to
    $\rho_0 = \{\rho_0^{(1)}, \rho_0^{(2)}, \rho_0^{(3)}\}$,
    $\mathrm{OP} = \min\{|o_{1j} - \rho_0|\}$
```

In the adaptive monitoring data processing algorithm, if the action save is chosen, the storage time must be considered. There are two time intervals. One is the storage time interval on the Apache server that determines how soon to send the data flow to the Apache publisher, and the other one is the storage time interval that determines how soon to send the data flow from the Apache publisher to the cluster publisher. During the interval when the cluster publisher waits to submit the data flow to the broker, all of the data flow must be transmitted from the Apache publisher cluster to the cluster publisher. Likewise, during the interval when the Apache publisher waits to submit the data flow to the cluster publisher, all of the cache data flows must be transmitted from the Apache adapter cluster to the Apache publisher. Assuming $T_{\rm Cp}$ is the time interval that cluster publisher sends query to Apache publisher. Let $T_{\rm q}$ be the query time to send query to each local node. Let $T_{\rm Ap}$ be the delivery time that Apache publisher sends data flow to cluster publisher. The time to obtain query data is

$$T_{\rm Q} = T_{\rm q} + nT_{\rm Ap},$$

where n is the number of the Apache publisher, then $T_{\rm Q} \leqslant T_{\rm Cp}$. The cluster publisher sets its own waiting time $T_{{\rm Ap},i}^{\rm W}$ for transmission from Apache adapter after Apache publisher i receives query message from the cluster publisher,

$$T_{{\rm Ap},i}^{\rm W} = T_{\rm Q}(i-1) + T_{\rm Ap},$$

where $T_{\rm Q}(i-1)$ is the transmission time that the $(i-1)$th Apache publisher sends data flows to cluster publisher.

4 Monitoring Data Flow Distribution

Definition 7 A monitoring data model is defined as a four-tuple:

$$s = \{\text{Timestamp, Metricname, Value, Source}\},$$

where Timestamp is the time of sending monitoring data flow, Metricname is the property of measurements, Value is the monitored value of measurements, and Source is the object of measurements (e.g. mysql, apache, server, and os). A spread flow can be denoted as $f = (\text{size}_\text{f}, u, v)$, where size_f is the bit of flow, u is the source of measurements, and v is the cluster server. We implement adaptive monitoring data flow distribution by using Java language to call VM monitoring interface. The detail procedure is described in Algorithm 2. The adaptive monitoring data flow distribution on the cluster adapter is described as follows:

Algorithm 2 Adaptive monitoring data flow distribution on the cluster adapter

```
(1) jmxServerConn =
        getMBeanServerConnection (urlForJMX)
        /getMBeanServerConnection is a function of VM
         connection*/
(2) domains = jmxServerConnection.getDomains()
(3) copM = jmxServerConnection.getAttribute
      (memory, "HeapMemoryUsage")
(4) copN = jmxServerConnection.getAttribute
      (bandwidth, "NetworkUsage")
(5) copC = jmxServerConnection.getAttribute
      (CPU, "CPUUsage")
        /* copM, copN and copC are real-time resource
         state in VMs */
(6) maxm = copM.get("max");
    usedm = copm.get ("used")
(7) maxn = copN.get("max");
    usedm = copN.get ("used")
(8) maxc = copC.get("max");
    usedm = copC.get ("used")
        /* maxm, maxn and maxc are maximum resource
         threshold in VMs */
(9) Call adaptive monitoring data flow decision on the
      cluster adapter
        /* call Algorithm 1 */
```

We create task queue to execute monitor on cluster publisher. The detail procedure is described in Algorithm 3. The monitoring task execution on the cluster publisher is described as follows:

Algorithm 3 Monitoring task do Execution() on the cluster publisher

```
(1) for each monitor task mt in current_execution_
      queue do
(2)    if mt.currstage = "run" then
(3)        send notification to broker for starting task
(4)        mt.currnumstage++
(5)        update the Q(mt.currnumstage)
(6)    end if
(7)    if mt.currstage = "Send" then
(8)        insert the flow packet into send_queue
           according to OP
(9)        mt.currnumstage++
(10)       update the Q(mt.currnumstage)
(11)   end if
(12)   if mt.currstage = "Recv" then
(13)       insert the flow packet into recv_queue
(14)       mt.currnumstage++
(15)       update the Q(mt.currnumstage)
(16)   end if
(17)   if mt.currstage = mt.totalNum then
(18)       mt.currstage = Finished
(19)   end if
(20)   if mt.currstage = "Finished" then
(21)       update the total monitoring task execution
           time
(22)       remove the monitoring task from current_
           execution_queue
(23)       insert another monitoring task from waiting_
           queue to current_ execution_queue
(24)       notify the broker about the completion of
           monitoring task
(25)   end if
(26) end for
```

Algorithm 3 describes the execution of monitoring data flow processing on the cluster publisher. For each monitoring task, the scheduling detector checks the current state of task execution. The monitoring tasks on the cluster publisher involve four stages: Send, Receive, Run and Finish. If the current stage Run has been completed, the execution time of the current monitoring task is updated according to the execution time of the next task. If the current state is Send, the cluster publisher task scheduler encapsulates the data flow packets and submits it to the transmit queue. Then the cluster publisher sends the data flow to the broker. If the current state is Receive, the cluster publisher tests whether the current queue has received a data flow packet. If a package has been received, then the current task is updated. If the current status is Finish, the total monitoring task execution time is calculated, and the finished task is removed from the current execution queue. Then, cluster publisher sends the message of finished task to the broker. Algorithm 4 describes the scheduling algorithm of the monitoring task on the cluster publisher.

Algorithm 4 Monitoring task scheduling on the cluster publisher

```
(1) for each monitoring task mt in current_execution_
      queue do
```

```
(2)   repeat
(3)      do Execution()
(4)      if mt.currstage == “Recv” then
(5)         execute the Monitoring Task at the top of
            waiting_ queue
(6)      end if
(7)   until mt.currstage == “Send”
(8) end for
```

Algorithm 4 describes multiple monitoring tasks that can be performed at the same time. When a monitoring task is waiting for input/output operations, other tasks in the waiting queue are also executable. Thus, the efficiency of data distribution is improved with the reduced delay time.

5 The Simulation and Analysis of Performance

The simulation platform is developed by Eclipse + JAVA. In the simulation, 200 users are randomly selected to access the data center server. The number of the servers varies from 20 to 100. Different CPU frequency is allocated for each server: 4, 8 and 16 GHz. The frequency of each virtual host varies from 1 to 4 GHz. We carry out our experiments using VMware server-2.0 that installs Microsoft Windows XP operation system. Hardware configuration is CPU i3-2330, 2.20 GHz and 4 GB RAM. The simulation platform sets three target functions o_{11}, o_{12} and o_{13}. Three thresholds of CPU, memory and network bandwidth resources are factors to make a decision for data distribution. Given a set of threshold parameters $\phi = \{0.55, 0.44, 0.33\}$, a collected set of resource value is $\delta = \{0.6, 0.3, 0.2\}$. Because of $0.6 > 0.55$, it means that only CPU resources are more than the threshold. Because of $0.44 - 0.3 > 0.33 - 0.2$, the memory resource cost is less, so the data can be cache. The experimental structure diagram is shown in Fig. 2.

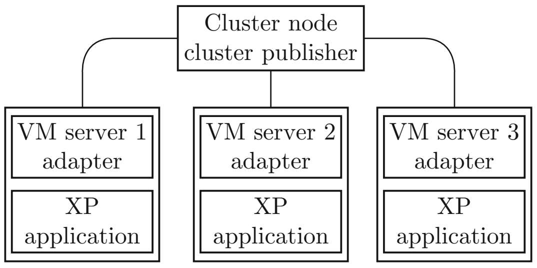

Fig. 2 The experimental structure diagram

Figure 3 shows the resource incremental change frequency of different VMs. VM1 receives 10 user requests; VM2 and VM4 receive 50 user requests; VM3 and VM5 receive 100 user requests. The incremental change of VM1 is the smallest, while the incremental changes of VM2 and VM4 are medium, and the incremental changes of VM3 and VM5 are the largest.

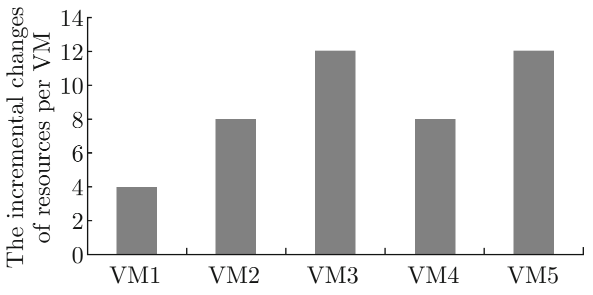

Fig. 3 Incremental change in the VM resources

Figure 4 shows the changes in notice rate of monitoring data in DALCGTC method. As the users increase, the resource increments of each virtual machine VM change from small to large. The notice rate of monitoring data of DALCGTC method changes more slowly than that of the single threshold method, because DALCGTC method combines the multi-threshold parameter decision and reduces the notices of monitoring data.

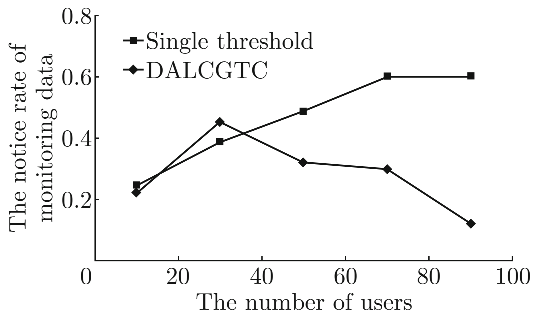

Fig. 4 The notice rate of monitoring data

Figure 5 shows the bandwidth utilization rate of monitoring data in DALCGTC method. As the users increase, the resource increments of each virtual machine VM change from small to large. More data are obtained. Since there is no data processing behavior deci-

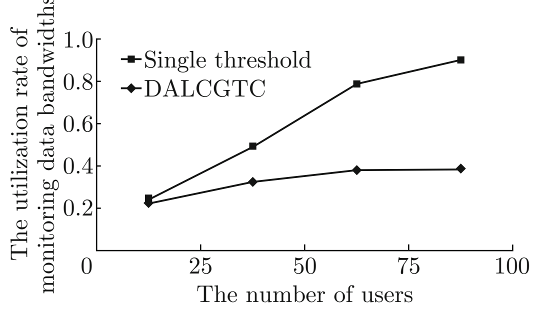

Fig. 5 The utilization rate of monitoring data bandwidths

sion in a single threshold method, the data will be sent to the cluster publisher only when the use of resources exceeds the threshold. The increase in monitoring data and use of bandwidth slows down the transmission of media data. But the DALCGTC method combines the multiple threshold parameter decision by using cache and notice. It discards unnecessary monitoring data, and reduces the volume of monitoring data and the bandwidth utilization.

6 Conclusion

This paper studies the solution to reducing the frequency of data distribution under the premise that the monitoring data are effective and valid. With the establishment of a "publish/subscribe" data distribution framework, this paper provides an efficient monitoring data distribution strategy and model monitoring data distribution as a multi-objective optimization problem, which is solved by using multi-objective decision-making algorithm. In order to test the performance of the strategy, we develop the simulation platform and test different processing strategies of monitoring data flows in the platform. Simulation results show that the use of multiple parameter threshold decision can effectively reduce the distributed data. The future work will focus on the solution to multi-objective optimization problem by considering the time threshold and resource threshold.

References

[1] Khanli L M, Analoui M. An approach to grid resource selection and fault management based on ECA rules [J]. *Future Generation Computer Systems*, 2008, **24**(4): 296-316.

[2] Guo Y, Guo L. IC cloud: Enabling compositional cloud [J]. *International Journal of Automation and Computing*, 2011, **8**(3): 269-279.

[3] Newman H B, Legrand I C, Galvez P, et al. MonALISA: A distributed monitoring service architecture [C]//*Proceedings of the Computing in High Energy and Nuclear Physics Conference*. La Jolla, Ca, USA: IEEE, 2003: 1-8.

[4] Kanstrén T, Savola R. Definition of core requirements and a reference architecture for a dependable, secure and adaptive distributed monitoring framework [C]//*Proceedings of the 3 International Conference on Dependability*. Venice, Italy: IEEE, 2010: 154-163.

[5] Sun Y, Xiao Z, Bao D, et al. An architecture model of management and monitoring on cloud services resources [C]//*Proceedings of the 3 International Conference on Advanced Computer Theory and Engineering*. Chengdu, China: IEEE, 2010: V3207-V3211.

[6] Lahmadi A, Andrey L, Festor O. Performance of network and service monitoring frameworks [C]//*Proceedings of the IEEE International Symposium on Integrated Network Management*. [s. l.]: IEEE, 2009: 815-820.

[7] He Q, Han J, Yang Y, et al. Formulating cost-effective monitoring strategies for service-based systems [J]. *IEEE Transactions on Software Engineering*, 2014, **40**(5): 461-482.

[8] Buyya R, Yeo C S, Venugopal S, et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility [J]. *Future Generation Computer Systems*, 2009, **25**(6): 599-616.

[9] Meng S, Liu L, Wang T. State monitoring in cloud data centers [J]. *IEEE Transactions on Knowledge and Data Engineering*, 2011, **23**(9): 1328-1344.

[10] Chung W C, Chang R S. A new mechanism for resource monitoring in Grid computing [J]. *Future Generation Computer Systems*, 2009, **25**(1): 1-7.