A Mechanism of Automated Monitoring Deployment in Grid Environment

Yinfeng Wang
Postdoctoral Research Fellow of Department of Computer Science, Hong Kong Baptist University
+852-34115972
yfwang@comp.hkbu.edu.hk

Kwang Mong Sim
Professor of Department of Computer Science, Hong Kong Baptist University
Kowloon Tong, Kowloon
+852-34115818
bsim@comp.hkbu.edu.hk

Xiaoshe Dong
Professor of Department of Computer Science,Xi'an Jiaotong Univesity
Shannxi,Xi'an
+86-29-82663951
xsdong@mail.xjtu.edu.cn

ABSTRACT
Constructing monitoring systems beforehand cannot satisfy the requirement of dynamically organizing resources. This paper describes a script-based approach to automate the deployment of grid monitoring service components. An automated deployment monitoring mechanism uses the information of dynamically joined nodes to deploy system components, which avoids potential errors and inconvenience by manual deployment. The mechanism is Grid Monitoring Architecture compliant, and provides user-defined XML interfaces that is scalable and integrates with existing grid platforms and monitoring systems smoothly. To decrease the deployment complexity, this paper proposes an N-ary-tree based algorithm that reduces the automated deployment complexity from linearity rate to logarithm and provides more than 10 times improvement on deployment time. Empirical results show that the automated deployment monitoring mechanism can rapidly deploy monitoring services, and monitors heterogeneous and dynamic grid resources efficiently.

Categories and Subject Descriptors
C.2.4 [**Distributed Systems**]: Network operating systems

D.4.8 [**Performance**]: Monitors

General Terms
Management, design

Keywords
Automated Deployment, Grid Resource Management, Monitoring Service.

Qshine08 July 28–31, 2008, Hong Kong, China.

1. INTRODUCTION
GRID monitoring is the act of collecting and providing grid resource status information that is used to facilitate system management, fault detection and performance optimization. Grid resources can potentially be heterogeneous and distributed. In service oriented computing environments, the resources are encapsulated into services to decrease the interoperation complexity and service interoperation based on Open Grid Service Architecture (OGSA) [1] and Web Services Resource Framework (WSRF) provide user-transparent access to resources. Open architectures and common standards including Web Services [2], XML [3], WSDL, SOAP and SLA, meet the grid requirements as the infrastructure of service oriented computing [4], which helps abstract and organize the resources to form the uniform resource view [1].

In service oriented computing environments, dynamic formation of Virtual Organization (VO), and dynamic organization and management of the resources are the key issue to guarantee the Quality of Service for grid resource management. Due to the dynamic nature of Grid computing systems, monitoring and scheduling are indispensable to dynamic resource organization. Following the discovery of resources for task execution, grid computing systems need mechanisms to flexibly monitor the resources to enforce automatic quality control [5] and to satisfy scheduling requirements. Given these requirements, a Grid monitoring system needs to perform automated deployment of monitoring services on new-entry resources. Encapsulating monitoring functions as services enable system functions to be extended more freely. On the foundation of the automated deployment monitoring service, we optimize the scalability of the system further, working on the policy for automated deployment and developing a platform-independent mechanism of an Automated Monitoring Deployment system (AMDs), which already enables automated deployment of monitoring services on Linux and AIX operating systems.

2. RELATED WORK

2.1 Existing Systems Review
Global Grid Forum's Grid Monitoring Architecture (GMA) [6] has been widely accepted as a standard for Grid monitoring systems. GMA defines the relationships among Producers, Consumer and Directory Service. However, Directory Service does not provide a clear definition on data pattern, query language

and transport protocol. In the Globus Toolkit 4.0, MDS [7] is a WS-based Grid monitoring system. It implements the index service (registry and cache) and the trigger service based on the common aggregator framework in WSRF. The function of Achieve that the two services, information providers and Clients (WebMDS) have implemented is an implementation of the GMA. In addition, Relation Grid Monitoring Architecture (R-GMA) [8] defines the data pattern of the monitoring information, query language and the function of Directory Service.

In the field of cluster monitoring, PARMON [9] is a C/S architecture of monitoring system. The customer can obtain real-time information of a set of fixed-nodes from the server and appoints a supervision node. SCMS [10] is an extensible management tool of the Beowulf cluster, it can help customer monitoring, inquiring system information, maintaining configuration, etc. SuperMon [11] is a monitoring system that has a hierarchy architecture. It needs to first install kernel module in each node, then gathers information Point-to-Point. It can divide nodes into groups to collect supervision information.

Ganglia [12] is a scalable distributed monitoring system. It is based on a hierarchical design that uses a multicast-based listen/announce protocol within clusters and a tree of point-to-point connections among representative cluster nodes to federate clusters and aggregate their states. It satisfies the requirement of monitoring multi-clusters. However, when we want to monitor a new cluster, we must manually install Ganglia on every node by configuring the environment, compiling the source codes and executing a make command. In large-scale clusters, the efforts to deploy monitoring system in this way would be considerable.

The information to monitor includes "latest-state", "continuous" and "history" [8]. To effectively decrease the data transmitted, GridEye [13] studies the relationship between the resource information of monitoring service and the information caching. ChinaGrid SuperVision (CGSV) [14] adopts the Producer/Consumer/Register Model. Based on JAVA and Web Service, it builds a three-tier monitoring and management architecture comprising "monitoring information source, information together and user view". Each tier can implement different monitoring functions through deploying and compositing different services.

2.2 Our Contributions

Existing resource monitoring systems have the following features:

1) Construction of monitoring environments is mostly a manual, ad hoc process beforehand;

2) Have high complexity to update or add supervision nodes but lack scalability;

3) High-cost of update, fault-tolerant and maintenance of monitoring service in distributed environments.

In the context of service computing environment, service or resource tend to have dynamic binding as the job execution. To respond to a variety of demands of the service, the VO member changes dynamically. These characteristics put forward a higher request to deploy the monitoring service which satisfies the request of dynamically organizing resources in Grids.

Using an XML-defined-interface approach, this article presents a script-based automated deployment mechanism to deploy resource monitoring service to lower the complexity of the supervision management effectively, ensure the system is scalable, and enables the monitoring of the dynamically organized resources without having to construct monitoring environments in advance. Moreover, we also exploit the system availability and the method of encapsulating monitoring service for integration with grid platforms as well as how to extend monitoring interface, and finally provide an evaluation of the system performance through experimentation.

3. AUTOMATED DEPLOYMENT OF MONITORING SERVICE

Given the heterogeneity among nodes in Grid computing systems, supporting the interoperability between services or between services and resources is an essential issue. The dynamic and interactive features of Grids require that monitoring services can 1) monitor the dynamically composed VOs, 2) organize and manage the dynamic resources, 3) obtain the required state information of the resources, and 4) support the interaction and management of the upper layer services. To address these issues, a Grid monitoring system requires the function of an automated deployment of monitoring service on newly-joined resources.

3.1 The Architecture of AMDs

As shown in Figure1, AMDs is divided into two logical layers: the data layer and the collective layer. The data layer gathers the state information of resources of each node in every cluster. To simplify the data layer, a single monitoring node is treated as a cluster with one node. In the collective layer, a Global Information Collector (GIC) gathers the state information of the resources of every cluster.

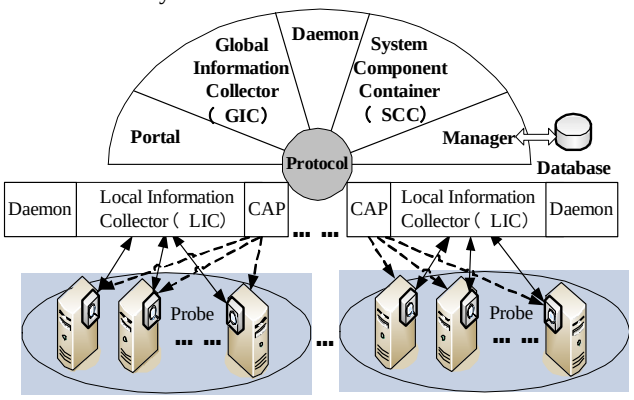

Figure 1. The Architecture of AMD system [17]

Table 1. List of modules function

Module name	Function
Global Information Collector (**GIC**)	Gathers the resource state information of every cluster in a round-robin manner processes the state information and stores it.
Local Information Collector (**LIC**)	Gathers the state information of the resources of the cluster on which LIC is running, and listens to the requests from GIC.

Probe	A script deployed in a node. The Probe waits for the request from LIC. When the request arrives, it immediately gathers the state information of the node, and sends the state information in XML format to LIC.
Daemon	Monitors other system components running in AMDs. It uses heartbeat information to check the status of each component. If it detects a failed component, it will restore the failed component
Manager	Implements the function such as the automatic deployment of system components on newly-joined resources. Administrators can also modify the system configuration through the Manager.
Portal	The Web interface of AMDs. Via the Portal, we can view the current and historical state information of the resources. The Portal also provides the management interface implemented by the Manager component. In addition, the Portal can be encapsulated as a service and also can be published in the grid environment. AMDs is compliant with GMA architecture and supports different administrative domains to use the monitoring service through the proxy.
Component Auto-deploy Proxy (**CAP**)	CAP is a system component that performs the automatic deployment of other system components on each newly-joined cluster.
System Component Container (**SCC**)	SCC is a container that maintains the system component and the corresponding configuration files. It is implemented based on the FTP protocol. When deploying the system components, CAP communicates with SCC and downloads the required system components via the FTP protocol.
Database	Stores the state information gathered from every cluster and the configuration of every cluster

The Daemons have two levels: the *local daemons*, which ensure that the system components of the local cluster are running, and the *global daemon*, which ensures the core components of AMDs (e.g., GIC) are running. The global daemon monitors every local daemon and ensures they are functional. The global daemon and the Manager monitor each other. If the global daemon failed, the Manager will reset it.

3.2 Flow of Automated Deployment System Module

After startup, AMDs listens to the request for deployment. When accepting an Administrator's deployment request or a resource's (e.g., a cluster) request to join the grid platform, AMDs will start a service thread, which can choose the different service approach based on different requests. The service thread starts the automatic deployment after it receives the configuration information for deployment. Figure2 shows the process of automated deployment.

Based on the deployment information of the newly-joined resources, the automated deployment process of system components is as follows:

1) *Deploying system components*. This step is divided into two phases: a) deploying CAP to the front-end machine, and b) starting CAP. According to the deployment configuration file, CAP can generate one N-ary tree source from the front-end machine and the subnet of the cluster. Depending on N-ary tree structure, CAP completes the deployment of system components. All the required components are downloaded from SCC.

2) *Inspecting of other local resource monitoring systems*. CAP inspects whether other local resource monitoring systems exist. If so, LIC obtains the state information of resources from the existing local resource monitoring system prior. Currently, AMDs only supports Ganglia.

3) *Generating the configuration file*. After deployment, CAP generates the configuration file based on the result of deployment. The configuration file contains information of configurations required by each component as well as other useful information, such as the SCC's IP address, and port number. In case any of the system components fail, AMDs will restore the component based on the configuration file.

4) *Starting the LIC.* CAP launches the LIC and the local daemon into execution. When started, LIC immediately gathers the state information of the resources based on the deployment information of the configuration file, and returns the deployment result.

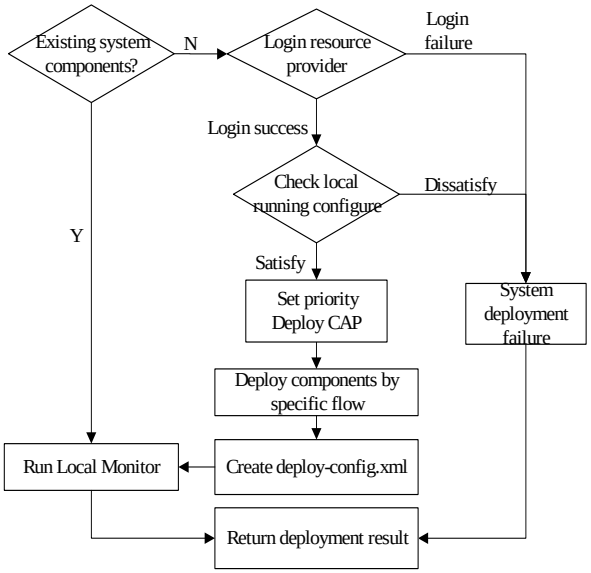

Figure 2. The process of automated deployment

If the deployment succeeds, the Manager will notify the GIC that there is a newly-joined cluster, and sends the deployment result to the GIC. The GIC then updates the list of monitoring clusters, and adds the newly-joined cluster to the list. From this point on the GIC will gather the state information of the resource of the newly-joined cluster.

3.3 Obtain Information of Newly-joined Clusters

Before automatic deployment of monitoring service system components on a cluster, the basic information of the cluster should be provided. This information can be added by the administrator or obtained from cluster registration. The newly-joined cluster needs to supply information that could be

represented by the following tuple:

$$Informaiton = (\text{Name} < OS_{type}, \text{Front-IP} >, \text{Login}<\text{Protocal,Username,Password}>, \text{Intall}<\text{Node,Path}>)$$

The information includes type of cluster operating system, IP address of Front-end machine (like a bridge from other systems to the cluster), hostname of each node, installation path, etc. Additionally, the resource provider needs to assign the rights for the monitoring service to ensure resource security. The new cluster only needs to offer a user account with common user's privilege to complete the automated deployment of system components. The system determines the type of Probe to be used to deploy system components and builds the new cluster's automatic deployment configuration file according to the information of the newly-joined cluster.

4. ENCAPSULATE THE MONITORING SERVICE

Using AMDs, complete system monitoring components deployment can overcome the difficulty of having to install real-time monitoring service in geographically-distributed environments. Furthermore, customers can choose a suitable automated deployment service to satisfy local management policies, and resolve the complexity of maintenance cost in the wide distributed open circumvents.

4.1 Encapsulate the Proxy Service

Adopting the tools such as Globus [15] or other grid develop studio, we can encapsulate the monitoring service into Resource Monitoring service proxy (RscMproxy), and deploying it at the portal. The objects of the service proxy are the commands that can be executed at the host node such as netstat, and Ping. The executable program, Java Class can be encapsulated into the service by writing an execute program shell script and integrated with embedded executable contents. The encapsulated monitoring service proxy process is shown in Figure 3.

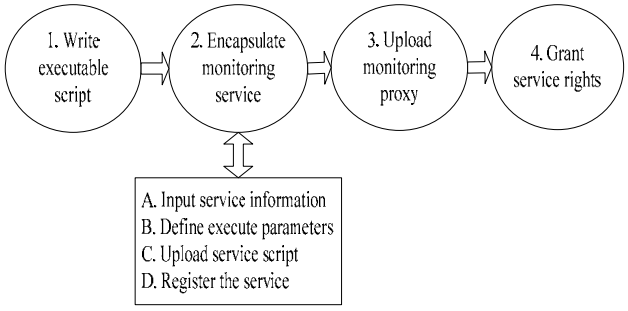

Figure 3. Encapsulate the monitoring service process

1) Write service executable script. Define the script runRscMProxy.sh as follows:

   a. #!/bin/sh
   b. java RscMProxy.class $1 $2 $3

2) $1, $2, $3 are the parameters for running the RscMProxy.class. $1 defines the monitored cluster, $2 defines appointed node, and $3 defines which kind of resources status type need view.

3) Encapsulate service. a) define the service name: RscMProxy; b) define service execute parameters; and c) upload script runRscMProxy.sh. Finally, the grid platform will automatically complete the service register.

4) Upload monitoring service proxy (RscMProxy.class) to the location of runRscMProxy.sh.

5) Grant the script executable rights. If script cannot execute, the access of this service will fail. Hence, the monitoring service needs some execute rights.

4.2 Extensible Probe Interface

To support different heterogeneous resources integration, AMDs provides an extensible XML-based probe interface. Various probe script for heterogeneous resources can be extended as follows:

1) By providing a unified XML interface for probes, local administrators can write or extend existing probes, then continue to extend collector and parser interface functions according to resource status type;

2) Supporting present local monitoring systems to obtain information from Ganglia as shown in Figure 3;

3) If no probe matches the node that joins recently, the local AMDs will discover suitable components released in other grid environment portals, and add probes in local system through the monitoring proxy service;

4) According to the type and version of the OS running on the newly-joined cluster, ADMs determines the type of probe that is used to deploy system components and build the new cluster's automated deployment configuration file.

5. PERFORMANCE ANALYSIS AND EVALUATION

5.1 Analysis and Optimization

The clusters usually have many host nodes which share reliable private network connections and use homogenous operating systems. When a cluster wants to join the monitoring system, all its nodes need to deploy same monitoring components. Hence, the group of nodes can be treated as tree-like topology. The topology or the ordering of the tree can be obtained by the front-end machine and the cluster's connection network. Consequently, a node may represent a separate data structure or a tree of its own.

Starting from the root node, every internal node takes charge of the deployment of its child nodes through the Component Auto-deployment Proxy (CAP) function. Although the multicast function is always disabled for security reasons, if the router (or switch) in the cluster supports multicast routing, the deployment traffic cost will be reduced by simultaneously delivering a single stream of components to group nodes.

The procedure of optimized ADMs deployment is detailed in the N-ary-Tree based Algorithm.

Algorithm name: **N-ary-Tree based** automated deploy algorithm

Input: Deploy component node topology, information of Newly-joined Clusters

Output: Complete the component deployment cover the N-ary

```
        Tree topology
String [] monitorHosts;   // Record all Hosts name
N:   The N-ary of the nodes
Void  AutoDeploy (index)

Begin:

1)   { //Each non-leaf node using multithread to deploy its
       //child node concurrently;
2)     int  startChild, endChild;
3)     startChild = (index - 1) * N + 2; //N-ary tree
4)     endChild  = index * N + 1;
5)     //Deploy the child node
6)     for (int i = startChild; i <= endChild; i++)
7)     {
8)       if ( isLeaf(i) )  // this node is the leaf node
9)       {
10)        Create deploy Probe shell commands based on the
           information of Newly-joined Clusters;
11)        Start thread, execute shell commands, deploy
           Probe  at child node i;
12)      }
13)      else // not the leaf node
14)      {
15)        Create deploy CAP proxy and Probe shell commands
           based on the information of Newly-joined Clusters;
16)        Start thread, execute shell commands, deploy Probe
           at child node i;
17)        Start CAP, using AutoDeploy method deploy the
           child nodes of the node i;
18)      }
19)    } //end for
20)  } // End deployment
End
```

If a lot of nodes are deployed concurrently in one large-scale cluster, the CAP node will have performance bottleneck. The deployment time for monitoring components can be reduced considerably by 1) adopting the N-ary tree automated deployment algorithm to completely automate the components deployment and 2) selecting a reasonable value of the parameter N to construct a N-ary tree by considering the sum of the nodes and its subnets.

The deploying time includes 3 periods: deploying front-end machine; deploying CAP and probe at non-leaf nodes; deploying probe at leaf-nodes. The AMDs deploying time is given by:

$$T_{autodeploy} = T_{front} + T_{N-leaf} + T_{leaf} \qquad (1)$$

Since every internal node needs to deploy the CAP module, the height of a full N-ary tree is $\lfloor \log_n sum \rfloor + 1$, Hence, it follows that the deploying time formula for AMDs is:

$$T = T_{front} + \lfloor \log_n sum \rfloor \times T_{CAP} + (\lfloor \log_n sum \rfloor + 1) \times T_{probe} \quad (2)$$

If concurrent requests arrive, the automated deployment service will generate corresponding threads to handle these requests. Since there is no **Write** operation during download, so there is no critical section of the process. After validating the corresponding request, the thread starts deploying system components.

Using N-ary-Tree based algorithm, we have the complexity $O(\log_N Sum)$. Thus, AMDs can scale well and effectively deploy monitoring services within large-scale clusters and grid environments.

5.2 Test Environment

The tests are based on the High Performance Computing Environment at Xi'an Jiaotong University. The test environment includes IBM workstation cluster, AIX/RS6000 cluster, and Dawning 4000L cluster. Table 2 shows the configuration of the distributed test environment composed by the three clusters.

Table 2. The list of the test environment

Name	OS Type	Node Number	Remark
IBM eServer1	Windows XP	1	SCC
IBM eServer2	RedHat Linux 9.0	1	The control center of AMDs
IBM Workstation Cluster	RedHat Linux 9.0	2	Testing cluster
RS6000	AIX 4.3.3	6	Testing cluster
Dawning 4000L	RedHat Linux 9.0	26	Testing cluster

Whereas the Dauwning4000L cluster is connected to another LAN (202.117.10.xxx), the control center and the System Component Container (SCC) of AMDs and all other test clusters are within the same LAN (202.117.15.xxx).

5.3 Test Result

With Ganglia, we must manually deploy the monitoring system on nodes. In the test, we use version 3.0.2 of Ganglia. Assume that all the dependency packages for Ganglia have been installed, and the installation files of Ganglia have been downloaded to the appointed node. The required time to manually deploy Ganglia on a node is the total time required to successfully execute the following 4 shell commands:

```
cd /path/to/ganglia-3.0.2;
/config. --prefix=/path/to/ganglia;
make;
make install
```

Table 3 shows the manual deployment (Batch run above 4 commands) time of Ganglia and the automatic deployment time of AMDs in the same IBM eServer. From Table 3, it can be seen that the deployment time of AMDs is less than that of Ganglia.

Table 3. The deployment time of one node install Ganglia or AMDS

Name(IBM eServer2)	**OS Type**	**Nodes**	**Deployment time(s,)**	**Deployment approach**
Ganglia	Red hat Linux 9.0	1	123.893	Manual
AMDs	Red hat Linux 9.0	1	45.312	Script-based

Table 4. Workload influence the deployment time

Name	OS Type	Nodes	Deployment time(s,)	
			Workload < 15%	Workload > 90%
Dawning 4000L	Red hat Linux 9.0	6	72.903	73.455

From Table 4, it can be seen that by deploying the same nodes, the cluster workload becomes an influence factor since more time is needed to complete deployment with heavy loads than light loads. The time for deploying front-end machine is shown in Table 5.

Table 5. Deploying time of Dawning 4000L front-end machine

Deployment time (s,)			Total time (s,)	Workload
CAP	Local component	Probe		
14.170	20.657	11.124	45.951	> 90 %

Since the cluster nodes' workloads change with time, taking heavy loaded nodes as non-leaf nodes to distribute and perform deployment and remote control will result in delaying their current process capacities and the amount of delay increases with the amount of workload or number of leaves.

If we deploy 24 nodes as shown in Figure 4, there is about 10% deviation between the deployment completion time and the theoretical value that reflect the performance of the automated deployment mechanism working in real complicated Grid environments.

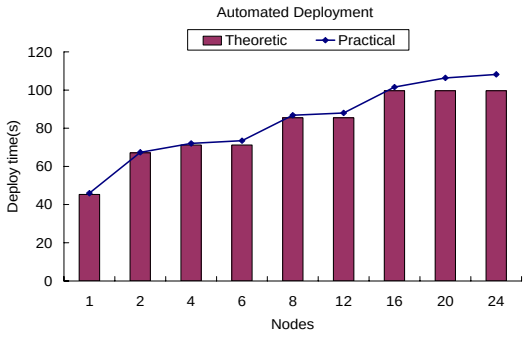

Figure 4. Deployment time of AMDs at Dawning4000L cluster

Figure 5 shows the response time of deploying monitoring services at cluster Dawning4000L using the manual method deploy Ganglia v.s. AMDs. From Figure 5, it is observed that there is a huge drop in deploying time (>90%). AMDs reduces the complexity of deploying time from linearity to logarithm. Hence, the advantage of the script-based deployment approach is more apparent in large-scale deployments.

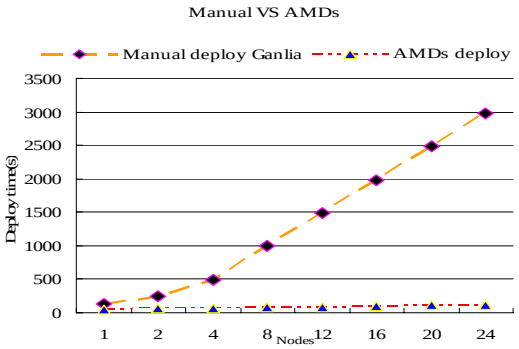

Figure 5. Deployment time of Ganglia at Dawning4000Lcluster Vs AMDs

In addition, with the script-based deployment approach, the cost in administrators' learning curve is also lower [16]. Within AMDs, administrators do need to even prepare any deployment script, but simply to fulfill the configuration web form of the newly-joined cluster. The newly-joined cluster interacts with the automatic deployment service, and the deployment can automatically complete without human intervention.

6. CONCLUSION

The complexity of the grid is ever-increasing as more heterogeneous resources and services are aggregated into various VOs. In such context, we design a mechanism that supports automated monitoring of resources in dynamic, distributed and open environments. By integrating the automated deployment service into existing grid platforms, global consistent views of monitored resources can be generated. Hence, the QoS of dynamic resource organizations can be improved because the required information about resources being monitored can be automatically obtained. Moreover, the script-based automated deployment approach is quite simple and flexible. It dramatically reduces the deployment complexity and achieves more than 10 times saving in deployment time. AMDs can also automatically detect and restore failed system components. As an open mechanism, it can cope with standard protocols and heterogeneous resources are unaware of any outside monitoring tools. By using XML-based probe interfaces the mechanism maintains scalability to work with other grid platforms and existing cluster resources monitoring system (such as Ganglia), and makes full use of available resources in grid environment.

There are several issues for future work. First, the monitoring information share among different domains has to be advanced; maybe these collective layers can be aggregated in P2P manner. Second, although the AMDs can support monitoring user-defined resources, administrators still need to manually extend information interfaces which should be improved. Moreover, AMDs should consider supporting interoperation with different directory service such as Universal Description, Discovery and Integration (UDDI) registry. Once a resource is published the automated deployment mechanism should begin to deploy monitoring service immediately, avoid administrators' intervention and achieve completely automated deployment ultimately.

7. ACKNOWLEDGMENTS

This work was supported in part by a competitive research grant from the Hong Kong Research Grant Council (HKRGC), project code: RGC/HKBU210906.

8. APPENDIX

Implemented using JAVA, AMDs makes full use of its platform independent feature. Our system achieves the status information of resources in MySQL and Round Robin Database (RRD http://oss.oetiker.ch/rrdtool/) database to optimize the system performance. The status information of resources can be displayed in graphical form (Figure. 6).

```
rrdtool create "/path/to/store/location/cpu.rrd"
--start `date +%s` --step 60
DS:user:GAUGE:120:0:100        DS:nice:GAUGE:120:0:100
DS:system:GAUGE:120:0:100      DS:idle:GAUGE:120:0:100
RRA:AVERAGE:0.5:1:60           RRA:AVERAGE:0.5:12:120
RRA:AVERAGE:0.5:24:420         RRA:AVERAGE:0.5:24:1800
RRA:AVERAGE:0.5:120:4380       RRA:AVERAGE:0.5:240:6570
```

Figure6. Shell commands for create CPU history trace in RRD

There are two data collection methods: events trace and sampling. The information is recorded in XML format, and is organized in hierarchical structure (Figure.7). Some attributes are static, (e.g., the memory size); these attributes will not change for a long time. To avoid introducing overheads, after gathering the information, the AMDs will not probe them twice unless active update requests are received. The dynamics status information monitoring is activated either by the specific events (signaling) or by timer interrupts (periodic value).

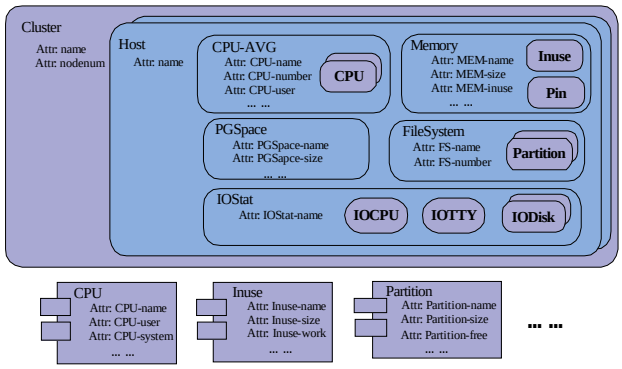

Figure 7. Layered information structure

9. REFERENCES

[1] I. Foster et al. The Open Grid Services Architecture, Version 1.0, 29 January 2005, http://forge.gridforum.org/projects/ogsa-wg.

[2] Hamid R. Motahari Nezhad, Boualem Benatallah, Fabio Casati, et al. Web Services Interoperability Specifications. *Computer*, 2006, 39(5),pp.24-32.

[3] Erik Wilde. XML Technologies Dissected. *IEEE Internet Computing*, 2003, 7(5), pp.74-78.

[4] Michael N.Huhns and Munindar P. Singh. Service-Oriented Computing: Key Concepts and Principles. *IEEE Internet Computing*, 2005, 9(1), pp.75-81.

[5] Ian Foster. Service-Oriented Science. *Science*, Vol 308, Issue 5723, 6 May 2005, pp. 814-817.

[6] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V.Taylor, R. Wolski. A Grid Monitoring Architecture, GWDPerf-16–3, Global Grid Forum (GGF), August2002. Available: http://wwwdidc.lbl.gov/GGF-PERF/GMA-WG/papers/ GWD-GP-16-3.pdf.

[7] http://www.globus.org/toolkit/mds/.

[8] Andy Cooke, Alasdair J G Gray, et al. R-GMA: An Information Integration System for Grid Monitoring. *Proceedings of the 10th International Conference on Cooperative Information Systems*, 2003

[9] R.Buyya. PARMON: a Portable and Scalable Monitoring System for Clusters. *Software-Practice and Experience*, 2000, 30(7), pp.723–739.

[10] http://www.opensce.org/components/SCMS.

[11] Matthew J. Sottile, Ronald G. Minnich. Supermon: a high-speed cluster monitoring system. *Proceedings of the IEEE International Conference on Cluster Computing*, September 2002.

[12] Matthew L. Massie, Brent N. Chun, David E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, Vol 30, June 2004, pp.817-840.

[13] Chu Rui, XIAO Nong, And LU Xi Cheng. A GMA Based Open Grid Resource Information Service. *Journal of Computer Research and Development*, 2004.12, pp.2114-2112.

[14] Weimin Zheng, Lin Liu, Meizhi Hu, Yongwei Wu, Liangjie Li, Feng He, Jing Tie, CGSV: An Adaptable Stream-Integrated Grid Monitoring System, *NPC 2005*, LNCS3779, pp.22-31.

[15] Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *IFIP International Conference on Network and Parallel Computing*. Berlin, Springer-Verlag LNCS, 2006, 3779, pp.2-13.

[16] Vanish Talwar, Dejan Milojicic, Qinyi Wu,Calton Pu, Wenchang Yan , et al. Approaches for Service Deployment. *IEEE Internet Computing*, 2005, 9(2), pp.70-80.

[17] Xiaoshe Dong, Yinfeng Wang, Zhongsheng Qin, et al. Automatic Deployment Mechanism for Monitor Service in Grid Environment. *Proceedings of the GCC2006W*, 2006, Oct.21-23, Changsha, China. Published by the IEEE Computer Society, pp 63-70.