Cluster Comput (2014) 17:1215–1223
DOI 10.1007/s10586-014-0391-y

Towards an integrated management system based on abstraction of heterogeneous virtual resources

Yongseong Cho · Jongsun Choi · Jaeyoung Choi · Myungho Lee

Received: 22 November 2013 / Revised: 10 April 2014 / Accepted: 28 June 2014 / Published online: 12 August 2014

Abstract Virtualization technology reduces the costs for server installation, operation, and maintenance and it can simplify development of distributed systems. Currently, there are various virtualization technologies such as Xen, KVM, VMware, and etc, and all these technologies support various virtualization functions individually on the heterogeneous platforms. Therefore, it is important to be able to integrate and manage these heterogeneous virtualized resources in order to develop distributed systems based on the current virtualization techniques. This paper presents an integrated management system that is able to provide information for the usage of heterogeneous virtual resources and also to control them. The main focus of the system is to abstract various virtual resources and to reconfigure them flexibly. For this, an integrated management system has been developed and implemented based on a libvirt-based virtualization API and data distribution service (DDS).

Keywords Integrated management · Heterogeneous virtual resources · Hypervisor · Libvirt · Data distribution service

Y. Cho · J. Choi · J. Choi (✉)
School of Computer Science & Engineering, Soongsil University, 369 Sangdo-Ro, Dongjak-Gu, Seoul 156-743, Korea
e-mail: choi@ssu.ac.kr

Y. Cho
e-mail: yongseong.cho@ssu.ac.kr

J. Choi
e-mail: jongsun.choi@ssu.ac.kr

M. Lee
Department of Computer Engineering, Myongji University, 116 Myongji-Ro, Cheoin-Gu, Yongin 449-728, Gyeonggi-Do, Korea
e-mail: myunghol@mju.ac.kr

1 Introduction

A few years ago, many servers were operated independently for each service such as a web server, database, e-mail, FTP, and etc. Each server could be operated independently and reliably. This approach, however, reduced the efficiency of the server. For example, web servers, mail servers, and FTP servers begin to work only if the interactions between users and servers are required. If users do not actively use those services, the usage is not used as much. Therefore, the utilization of the server becomes deteriorated to reduce the efficiency of the server as well.

Until recently, the performance of servers has been improved. However, the efficiency of the server is directly related to the costs spent to maintain the computing resources. The reduced efficiency of the server causes unnecessary maintenance cost and power consumption due to the waste of computing power. These days, virtualization technologies have attracted attention to solve the problem of the decrease in the efficiency of the servers [1–3]. Virtualization technologies are logically designed to separate a single high performance computer into several machines and to allow them to operate at the same time. It makes it possible to run several operating systems at the same time on the server computing resources. Servers with virtualization technologies can provide various services at the same time using web servers, database servers, ftp servers, and etc. with these features. Therefore, it is possible to reduce the idle time, improve the efficiency of computing resources, and finally to reduce the number of physical resources of the server system once the virtualization technologies are applied. These benefits reduce maintenance costs spent for installation and operation of server systems. In addition, those benefits reduce the cost of the system implementation by reducing the installation space and power costs as well

as simplifying the procedures for the configuration of the system.

Aforementioned virtualization technologies are offered in the form of various heterogeneous hypervisors such as XEN [4], KVM [5], VMware [6], and etc. While these hypervisors have their own unique characteristics individually, they provide common methods on how to build the virtualized environments and how to install the virtual machines. Generally, a hypervisor provides a management tool that allows users to control or monitor virtual resources so that users can easily control the server. If users build a server system where multiple operating systems are mounted using heterogeneous hypervisors, an integrated management system is required for the effective management and the control of these hypervisors [7–9].

This paper presents an integrated management system for the management of heterogeneous hypervisors. The main idea of the system is based on two techniques: a libvirt-based virtualization API [10] and data distribution service (DDS) [11]. The libvirt-based virtualization API is a library to integrate and abstract virtual environments provided from heterogeneous hypervisors. By using virtualization API, users can extract the information on virtual machines in heterogeneous virtual environments and control them specifically. Libvirt has been developed as an open source project, and it provides the virtualization API. It supports various hypervisors such as XEN, KVM, VirtualBox, VMware, and etc. DDS is a middleware to support a data-centric publish/subscribe programming model in distributed environments. It is highly scalable and flexibly reconfigurable, so it is not constrained by the number of targets for communication. The system transmits in real-time by using DDS, the state information, and control commands on the virtual machines and even actively responds to the expansion and collapse of hypervisors on the server system.

2 Related works

In this section, recent works on the management of virtual resources [4–8,12,13] will be compared and analyzed with the virtualization API and the DDS described in details.

2.1 Virtual resource management

The research on the virtualization has been carried out by companies of VMware, Microsoft, and etc, and by open source projects such as Xen and KVM. In recent years, not only virtualization of hardware but also virtualization itself has been studied in terms of information, workload, and etc.

The details are as follows: VMware [6] is a leader in the hypervisor market dominating the whole market share. It sells products for the server virtualization and desktop virtualization. These products provide convenient user-interfaces and can be executed on both Windows and Linux. Microsoft provides Hyper-V [12] and System Center VMM. Since most computing environments are currently based on Windows nowadays, these systems are easier to access and they have strengths in that they can work with a wide range of Windows solutions. Xen [4], developed at the University of Cambridge, is a hypervisor based on the Linux operating system. The initial version supported only para-virtualization, so it was troublesome to run the guest operating system. Now the problem has been fixed. KVM [5] is a hypervisor based on Qemu [13], an open source hardware emulator, and it is executed on the Linux operating system. KVM uses the software emulator by installing a guest operating system.

All the hypervisors provide an independent virtualization environment in response to the each virtual machine (VM). Because there are a large number of approaches for getting usage information of VM resources from diverse heterogeneous hypervisors and controlling them, building a server system with them costs more for the management and maintenance. In order to solve this problem, there have been several works carried out to integrate the virtual resources and manage them regardless of the type of the hypervisor as follows: virtual infrastructure environment (VINE) [7] is a solution that can virtualize the virtual machine that runs on heterogeneous hypervisors. It was developed by Distributed Management Task Force (DMTF) standards and open software. VINE is a solution to build a new virtual environment by collecting virtual resources and it can build virtualization regardless of the physical layer.

Grid resource management framework (GridRMF) [8] is a framework acting as an intermediary between the resource provider and the resource requester, and it provides functions of collecting idle resources, analyzing users' requirements, and distributing resources effectively. However, GridRMF manages resources in resource management system (RMS) which has a different internal structure so it is difficult to manage individual virtual resources separately. OpenNebula [14] is an open source cloud platform to manage a number of heterogeneous hypervisors which can execute virtual machines. It supports numerous heterogeneous hypervisors such as XEN, VMware, KVM, and so on. Eucalyptus (elastic utility computing architecture for linking your programs to useful systems) [15] is an open source cloud platform to establish Private, Public, and Hybrid Cloud. It has originally started as a research project of UC Santa Barbara and been commercialized by Eucalyptus Inc., however, it is still being maintained and developed as an open project.

Cloud Stack [16] is an open source cloud computing software for creating, managing, and deploying infrastructure cloud services and it is well designed for the expandability and intensified management of the system. Glance [17] is a lower rank project of OpenStack [18] project and it pro-

vides a control service of virtual machine's images. It is an API server to supply with image service which can search, register, and transport the images of virtual machines.

2.2 Preliminary

DDS [10] is a communication middleware protocol of which the standard specifications have been defined by OMG (Object Management Group). It is a communication middleware standard to ensure the real-time (Pseudo-Real time) centering for the defense businesses and the reliability of messages. DDS supports various OSs (Unix, Windows, Linux, and VxWorks) and programming languages (C/C++, Java, and CORBA), and it is useful to develop the communication programs and to increase the compatibility among heterogeneous devices. In addition, it is possible to reduce the load that the server has to have in the existing client-server approach by forming dynamic network domains, and joining and leaving domains freely. In this paper, we use the DDS middleware to support the delivery of the real-time information for the virtual resources and the dynamic registration/deletion of heterogeneous hypervisors.

Libvirt [11] is a library to manage heterogeneous virtualization software. It provides such a common interface as hypervisor-agnostic API to securely manage the guest operating system running on the hypervisor. Hypervisor for virtualization has a very wide variety of types and each hypervisor provides its own API and operational approach. Libvirt provides common functions such as creating, modifying, managing, monitoring, migration, and suspension to API. Our management system extracts the information of virtual machines managed by heterogeneous hypervisors and controls them using the Libvirt library.

3 Integrated management system of virtual resources

The main focus of the system is to abstract various virtual resources and to reconfigure them flexibly. If it is possible to provide users with multiple hypervisors simultaneously in the system, users can choose one of them for their purposes and convenience. In this section, we describe a conceptual architecture of the system representing system configuration and its major components in detail.

3.1 System architecture

Figure 1 shows an overview of the integrated management system for heterogeneous virtual resources. At the bottom of Fig. 1, there are multiple virtual machines configuring the server system. Each physical computing resource is configured with several virtual machines by using heterogeneous hypervisors.

The brokers implement libvirt which is the virtualization API, and they are used to extract information from virtual machines and to control them so that underlying hypervisors can be managed regardless of their types of hypervisors. The DDS communication middleware collects and sends the information of the virtual machines from distributed brokers to the integrated management system and then passes control information to each broker. The integrated management system shows the information of the virtual machines received from a number of brokers through a GUI-based web application. The virtual resource manager can check the current information of the virtual machine and can control each virtual machine by connecting to the web server of the integrated management system.

As shown in Fig. 1, the system consists of hypervisors to build a virtual environment, HRBs to extract information from the virtual machines and control them, A-HRB to serve as an abstracted interface to access HRB, DDS to control and deliver the information of virtual resources in real-time, and VRMS to monitor and control virtual resources.

3.2 Hypervisor and virtual machine (VM)

VM refers to the hypervisor installed to the physical devices (PD) in order to build a server system, and it also refers to the virtual machine inside the hypervisor. Multiple virtual machines can be installed inside the hypervisor, and each virtual machine is an instance to drive the actual service instance. The system can manage multiple virtual machines.

Definition 3.1.1 (*Heterogeneous and Virtual Machine*)
It has the same functions between Heterogeneous Virtual Machine.
There is C in VM.
C is a communication.
$C = \{DDS_{MiddleWare} \cup Libvirt\};$

3.2.1 Hypervisor resource broker (HRB)

HRB is a component to extract the information from the virtual machines of the hypervisor and to control them. In order to extract the information of a virtual machine connected to the hypervisor, HRB is also connected to the hypervisor. HRB extracts the information of the VMs that the connected hypervisor is managing or it executes a control command of the virtual resource manager. HRB is a component to actually execute all of the functions of the system and it uses visualization API. In this paper, libvirt library is used to implement the HRB which handles several heterogeneous hypervisors.

Definition 3.1.2 (*Hypervisor Resource Broker*)
It takes the libvirt to process the request which is going to be sent from A-HRB. Info is Information.

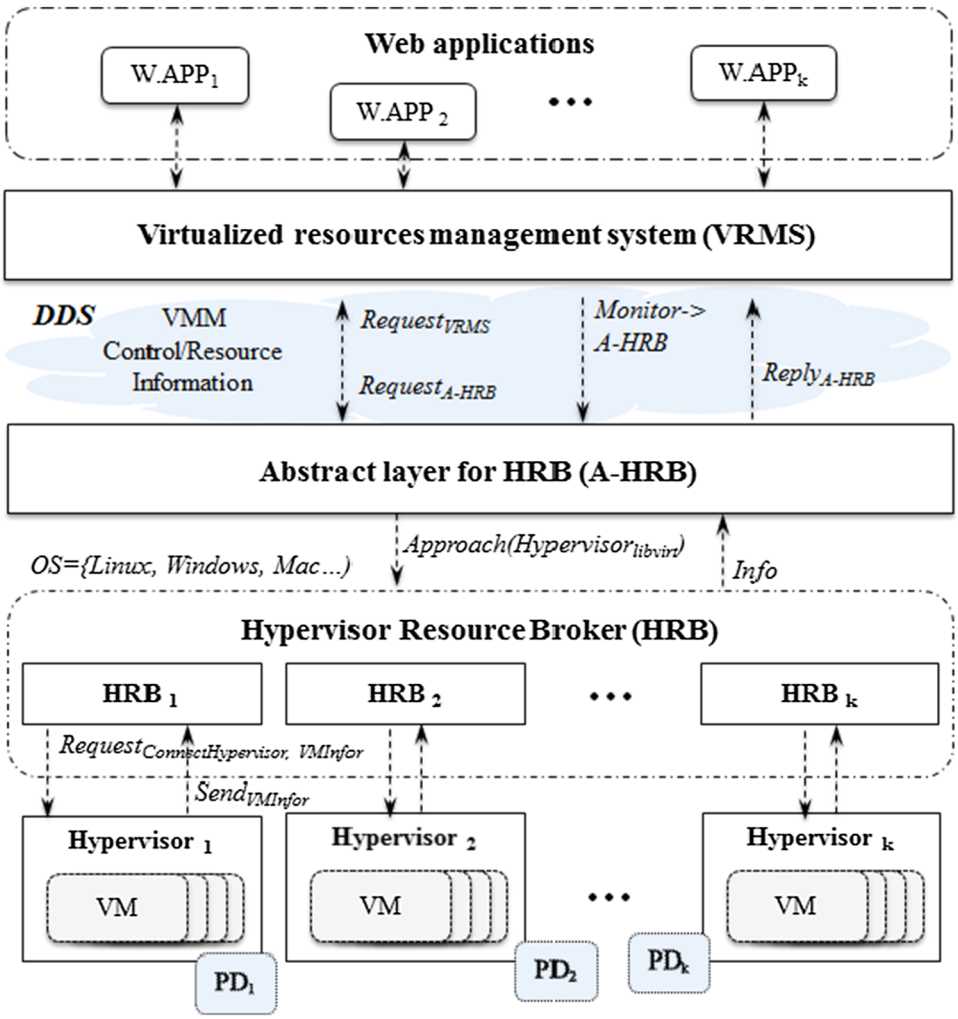

Fig. 1 The system architecture based on abstraction with virtualization API (libvirt) and DDS

$\text{Approach(Hypervisor)}_{\text{libvirt}} \rightarrow \text{VM(Info)}$;
$\text{VM(Info)} \rightarrow$ A-HRB;

3.2.2 Abstract layer for hypervisor resource broker (A-HRB)

A-HRB is an abstraction layer of HRB. In our integrated management system, libvirt library is used to implement the HRB. However, if HRB is implemented by using multiple virtualization APIs, there is no effective method to handle them. Thus, a new layer is required to solve this problem. A-HRB defines several basic functions to be provided by virtualization API. Those functions defined in A-HRB are linked to the libvirt and several virtualization APIs to transfer the information of the virtual machines and execute the control commands of the administrator.

Definition 3.1.3 (*Abstract Layer for HRB*)
It forwards the requests which will be sent from DDS to HRB, to process the requests of VM resource manager.
Result(HRB) → DDS;

3.2.3 Data distribution service (DDS)

DDS is located on the top of the A-HRB. DDS is a communication middleware for real-time communications in a distributed environment and it is well suited to the system in terms of management and control of distributed hypervisors. The integrated management system for virtual resources receives the information of virtual resources and controls them in real-time using the DDS communication middleware. DDS is the leading data-centric publish/subscribe messaging standard for integrating distributed real-time applications. In the system, VRMS and A-HRB are registered as publisher-subscriber with respect to each other. VRMS is registered as a subscriber of A-HRB, receiving the information of virtual resources and monitoring them. On the contrary, A-HRB is registered as a subscriber of VRMS, receiving the control infor-

mation of virtual resources of VRMS and controlling them.

Definition 3.1.4 (*Data Distribution Service*)
It is a middleware based on publish-subscribe model which is to use in distributed environment.
DDS = {Flexible Configuration};
$\text{Request}_{\text{VRMS}} \leftrightarrow \text{Request}_{\text{A-HRB}}$;
$\text{Reply}_{\text{A-HRB}} \rightarrow$ VRMS;
Monitor → A-HRB;

3.2.4 Management system for virtual resources (VRMS)

VRMS is a server program for integrating and managing all virtual resources configuring the server system. VRMS receives the information of all the virtual machines contained in the system and store them in the memory. The stored information can be monitored through the web application provided by VRMS. The web application provides an intuitive UI to allow users to manage all the virtual resources easily. The virtual resources manager can check the information of virtual machines on a variety of devices including PCs, smart phones, tablet PCs, and etc. and control the power ON/OFF of the particular virtual machines.

Definition 3.1.5 (*Management System for Virtual Resources*)
VRMS is a server program for integrating and managing all virtual resources configuring the server system.
$\text{Request}_{\text{VM.Resource.Manager}} \rightarrow \text{DDS}_{\text{Middleware}}$;

4 Implementation

The integrated management system of virtual resources is implemented based on the architecture introduced in Sect. 3.2. It is possible to see the screen shots of monitoring and controling the information of virtual machines using the web-based GUI applications.

4.1 Hypervisor resource broker (HRB)

HRB can extract the information of virtual machines from the hypervisor, and it can control a specific machine. The system is aimed at several heterogeneous hypervisors, so HRB can extract the information of virtual machines from the heterogeneous hypervisors and control them. To support this, the libvirt library is used to implement HRB. Libvirt library goes through three processes: 'connection to hypervisor,' and 'extraction of a list of virtual machines,' followed by 'access to virtual machine.' These processes can be implemented by the same API provided by libvirt, regardless of the type of hypervisors. The usage information of virtual machine in a hypervisor consists of two types of information; the information related to the virtual hardware such as CPU and memory

```
Conn= virConnectOpen("xen:///");
Conn= virConnectOpen("qemu:///system");
```

Fig. 2 XWN/QEM Access Code using libvirt

```
// Connect hypervisor by libvirt
virConnectPtr    conn;
virDomainInfo    info;
virNodeInfo      nodeinfo;

//get node info by libvirt
virConnectListDomains(conn, ids, maxids );
virConnectListNetworks(conn, names,maxnames );
virNodeGetInfo(conn, &nodeinfo );
```

Fig. 3 Collecting the virtual machine list in an active state

are collected through the libvirt library as well as the ones on virtualized physical resources being collected also through the libvirt library.

By using the libvirt library, it is possible to obtain the system log for virtualized physical hardware, but it is not possible to get the usage of the network in real-time. Therefore, the system receives the usage of the clients' virtual network cards in order to get the network usage of the VMs. Each VM is connected to one of the client VM virtual network cards. By looking at the amount of traffic on the virtual network card, it is possible to get the network usage in the VM. In the same way, the usage of each VM is collected at a certain period of time. If the information is published in a DDS message form, the server program subscribes to it to get information. This information is obtained from rrdtool [19,20] that handle time-series data like network bandwidth, temperatures, CPU load, and etc, and also includes tools to extract **RRD** data in a graphical format.

Figure 2 shows a code connecting to the hypervisor by using the libvirt API. Since each virtual machine is managed by the hypervisor, it is necessary to connect to the hypervisor in order to import the hardware information of the virtual machine. Libvirt supports a specific driver for each hypervisor and it can access each hypervisor by using these drivers. Figure 3 shows codes designed to receive the list of virtual machines managed by a hypervisor. The codes in Fig. 3 contain a domain list, a network list, and hardware information of the virtual machines currently working inside the hypervisor.

Figure 4 shows codes designed to collect the information about each of the virtual machines. By using the list of active virtual machines with the codes in Fig. 4, the status information for each virtual machine can be collected. After getting a pointer to the virtual machine, the state information of the virtual machine is saved in the virDomainInfoPtr structure in which information of the virtual machine such as CPU, memory, and etc. is also stored. Then additional informa-

```
virDomainPtr dom = virDomainLookupByID (conn, ids[11] );

// Extract information about a domain's block device.
virDomainGetInfo(dom, &info );
char UUID[VIR_UUID_STRING_BUFLEN] = {0};
virDomainGetUUIDString( dom, UUID );
const char *name = virDomainGetName( dom );
char *ostype = virDomainGetOSType( dom );
virDomainGetAutostart(dom, &autostart );
```

Fig. 4 Collecting the information by virtual machine

```
namefp = popen( "cat /proc/net/dev | grep vnet | awk'{print &1}' ", "r" );
char *popen_start = " cat/proc/net/dev | grep";
char *popen_end= " | awk'{print &2}' ";
char *popen_end2 = " | awk'{print &10}' ";
```

Fig. 5 Collecting network interface information by the virtual machine

tion such as UUID value, domain name, OS installed on the virtual machine, and etc. is collected.

The network usage of VM cannot be monitored with libvirt. We solved this problem by using the network interface of the hypervisor. When setting up a network while creating the virtual machine, the virtual network interface is created on the hypervisor and it is connected to the network interface of the virtual machine. By monitoring this interface, one can be aware of the actual amount of data sent and received in a virtual machine. The system collects the usage information of the network interface connected to each virtual machine by using the method. Figure 5 shows the console commands to extract the information of the virtual machine's network usage in HRB. By using these commands, the network information can be collected for each virtual machine.

4.2 DDS communication module

DDS is the leading data-centric publish/subscribe messaging standard for integrating distributed real-time applications. The system sends and receives data on the DDS communication middleware. DDS is an optimized communication middleware using publish/subscribe messaging approach in distributed environment. All the publishers and subscribers should match their data types to send and receive data each other.

Figure 6 shows the implementation of various data types that are defined for the information and control of virtual resources in the system. In Table 1, more details of data types shown in Fig. 6 are provided.

The file formats of the contents in Table 1 are the extension of .IDL which is used in the DDS communication middleware. They can also be roughly classified into the system information of a virtual machine and the control

```
module Message {
    const long  NAMESIZE = 256;
    typedef string <NAMESIZE> nameType;

        struct DomInfo {
            string hypervisor_id;
            long   id;
            …;
        };

    #pragma keylist DomInfo id
    struct Sig {
        string hypervisor_id;
        …;
    };

    #pragma keylist Sig ctl_sig
};
```

Fig. 6 Implementation of data type for VM control and information in DDS environments

Table 1 Data types for getting information of heterogeneous VMs

DomInfo structure type	
string *hypervisor_id*	*// Hypervisor ID*
long *id*	*// Virtual machine ID*
string *name*	*// Virtual machine name*
string *uuid*	*// Uinique user identidfier of virtual machine*
string *ostype*	*// Operating system type number*
long *status*	*// Status number of virtual machine*
long *cpus*	*// CPU count of virtual machine*
long *cputime*	*// CPU Time of the virtual machine*
long *maxmem*	*// Max memory size of virtual machine*
long *usedmem*	*// Current memory size of virtual machine*
long *autostart*	*// Auto start data*
long *net_rev_avg*	*// Average value of receive data in network*
long *net_snd_avg*	*// Average value of send data in network*
long *net_rev_max*	*// Max value of receive data in network*
long *net_snd_max*	*// Max value of send data in network*
long *ack*	*// Ack data*

signal for controlling the virtual machine. The `struct DomInfo` is the information used to collect the status information in HRB and send it to VRMS. It includes a hypervisor ID, identifier of the virtual machine, OS type, status, the number of CPU cores, and information about CPU usage time, memory usage, and network usage, and etc.

The `structure Sig` presented at the bottom of Fig. 7 is the information used to control the virtual machine from the VRMS, and it is used in the HRB. As shown in Table 2, it includes a hypervisor ID, name of virtual machine, and control signal.

Table 2 Data type for VM control in DDS environment

Sig structure type	
string *hypervisor_id*	// Hypervisor ID
string *vm_name*	// Name of virtual machine in hypervisor
long *ctl_sig*	// Control data of virtual machine

5 Tests and evaluation

The usage information of the virtual machine collected from the hypervisors is stored inside the VRMS in real-time. The graphs are generated by statistics obtained for a specific period of time based on the information stored.

In Fig. 7, a sequence diagram which shows the process occurred inside the system to deal with the request of a virtual resource manager by using VRMS is presented. The virtual resource manager can log-in the account through the web application provided by VRMS. VRMS will show system management screen in case of successful log-in. A virtual resource manager requests the information of virtual resource through the management screen. The request is transferred to the middleware of DDS communication via VRMS and it goes to A-HRB where a certain virtual resource is located. A-HRB can receive the information of a specific virtual resource through HRB and then transfers it to VRMS by DDS. VRMS finally completes the sequence by showing the information of a particular virtual resource to the virtual resource manager.

Figure 8 shows the management screen for resources usage information of heterogeneous hypervisors that VRMS provides on a web browser screen. The left-hand side (red-dashed line) of the figure includes the manager of the virtual machine currently connected, the hypervisors of the system, and the virtual machines running on each hypervisor. The manager can select the virtual machine that he wants to check and then monitor it through the graph on the right-hand side (black-dashed line) of the figure. The graph shows the usage of CPU, memory, and network which all have been collected from the virtual machines. The monitoring screen is updated at a regular time interval which can be changed by the manager. The manager can monitor the information of the entire virtual resources of the server system, and he can control or manage them conveniently.

6 Conclusions

In this paper, we presented an integrated management system that can provide resource usage information of heterogeneous virtual resources and control them. The two main focuses of the system are to abstract various virtual resources and to reconfigure them flexibly. For this, the system uses a libvirt-based virtualization API and DDS. Specifically, libvirt, the virtualization API, was used as the underlying technology to extract virtual machine information from multiple heterogeneous hypervisors and to control them. It also allows users to manage heterogeneous hypervisors by supporting most of hypervisors such as VMware, XEN, KVM, and etc. In addition, with the DDS communications middleware, the system can deliver the status information of the virtual resources in real-time and the control information of a system manager for virtual machines on the heterogeneous hypervisors.

The system includes an abstraction module of A-HRB which can support various virtualization APIs so that it can be easily extended to support a new hypervisor or a new virtualization API. In addition, the system provides a web-based GUI interface so that users can easily use it as shown in Sect. 4. This allows the resource manager to use web

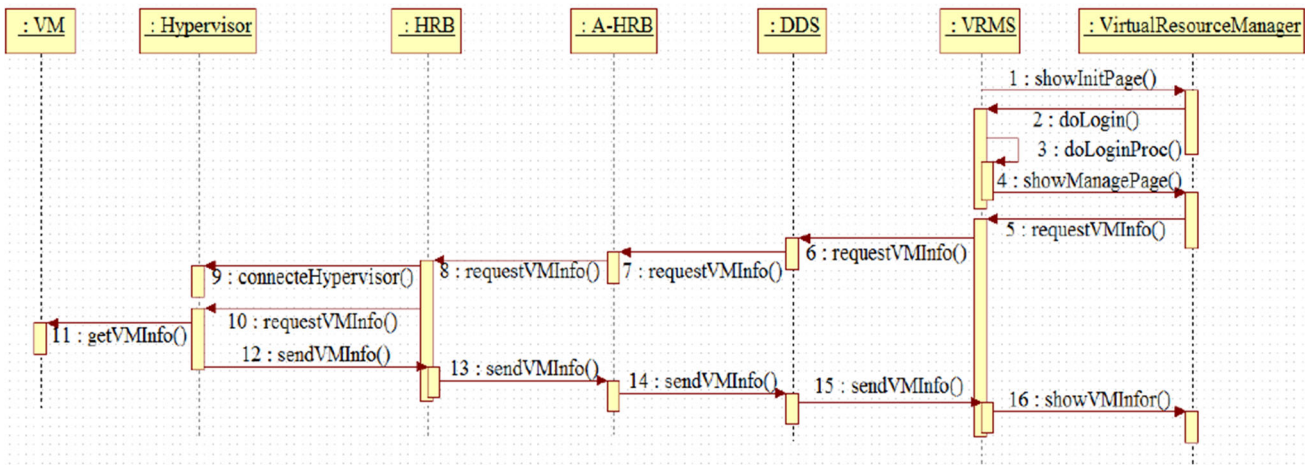

Fig. 7 The process to get resource usage information of heterogeneous virtual machines

Fig. 8 Resource usage information from heterogeneous virtual machines

applications to easily manage various heterogeneous virtual resources.

Acknowledgments This work was supported by the Industrial Convergence Core Technology Development Program (No. 10048474) funded by the Ministry of Trade, Industry & Energy (MOTIE), Korea.

References

1. Paul, I., Yalamanchili, S., John, L.K.: Performance impact of virtual machine placement in a datacenter. In: IEEE 31st International Performance Computing and Communications Conference (IPCCC) (Dec. 2012), pp. 424–431 (2012)
2. Suryanarayana, V., Balasubramanya, K. M., Pendse, R.:Cache isolation and thin provisioning of hypervisor caches. In: IEEE 37th Conference on Local Computer Networks (LCN) (Oct. 2012), pp. 240–243 (2012)
3. Kovari, A., Dukan, P.: KVM & OpenVZ virtualization based IaaS open source cloud virtualization platforms: OpenNode, Proxmox VE. In: IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics (SISY) (Subotica, Sep. 2012), pp. 335–339 (2012)
4. Yuchao, Z., Bo, D., Fuyang, P.: An adaptive Qos-aware cloud. In: Proceedings of International Conference in Cloud Computing Technologies, Applications and Management (ICCCTAM), (2012)
5. Alarifi, S.S., Wolthusen, S.D.: Detecting anomalies in IaaS environments through virtual machine host system call analysis. In: International Conference for Internet Technology and Secured Transactions, 10–12 Dec 2012, pp. 211–218 (2012)
6. Arsene, A., Lopez-Pacheco, D., Urvoy-Keller, G.: Understanding the network level performance of virtualization solutions. In: IEEE 1st International Conference on Cloud Networking (CLOUDNET), 28–30 Nov 2012, pp. 1–5 (2012). doi:10.1109/CloudNet.2012.6483645
7. Young-Woo, J., Jin-Mee, K., Seung-Jo, B., Kwang-Won, K., Young-Chun, W., Sang-Wook, K.: Standard-based virtual infrastructure resource management for distributed and heterogeneous servers. Adv. Commun. Technol. ICACT 2009, 2233–2237 (2009)
8. Eun-Ha, S., Yang, T., Young-Sik, J.: Hierarchical resource management model on web grid service architecture. J. Supercomput. **46**(3), 257–275 (2008)
9. Audrey M., Thierry D., Emmanuel L., Michelle S.: A CIM-based framework to manage monitoring adaptability, network and service management, inter. work. on systems virtualization management, pp. 261–265 (2012)
10. Yubin W., Bowen J., Zhengwei Q.: IO QoS: A New Disk I/O Scheduler Module with QoS Guarantee for Cloud Platform. In: International Symposium on Information Science and Engineering (ISISE), 14–16 Dec 2012, pp. 441–444 (2012). doi:10.1109/ISISE.2012.105

11. Noguero, A., Calvo, I.: A time-triggered data distribution service for FTT-CORBA. In: IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA), 17–21 Sept 2012, pp. 1–8 (2012). doi:10.1109/ETFA.2012.6489552
12. Gusev, M., Ristov, S.: Superlinear speedup in Windows Azure cloud. In: IEEE 1st International Conference on Cloud Networking (CLOUDNET), 28–30 Nov 2012, pp. 173–175 (2012). doi:10.1109/CloudNet.2012.6483679
13. Kudryavtsev, A., Koshelev, V., Avetisyan, A.: Modern HPC cluster virtualization using KVM and palacios. In: International Conference on High Performance Computing (HiPC), 18–22 Dec 2012, pp. 1–9 (2012). doi:10.1109/HiPC.2012.6507495
14. Sotomayor, B., Montero, R.S., Llorente, I.M., Foster, I.: Virtual infrastructure management in private and hybrid clouds. IEEE Internet Comput. **13**(5), 14–22 (2009)
15. Gusev, M., Ristov, S., Donevski, A.: Security Vulnerabilities from Inside and Outside the Eucalyptus Cloud. In: BCI '13 Proceedings of the 6th Balkan Conference in Informatics, pp. 95–101 (2013)
16. CloudStack: Open Source Cloud Computing Software. http://www.cloudstack.org. Accessed May 2012
17. Bist, M., Wariya, M., Agarwal, A.: Comparing delta, open stack and Xen Cloud Platforms: a survey on open source IaaS. In: IEEE 3rd International Advance Computing Conference (IACC) (2013)
18. OpenStack Cloud Software. http://www.openstack.org. Accessed May 2012
19. Wu, M., Zhang, Z., Li, Y.: Application Research of Radoop Resource Monitoring System Based on Ganglia and Nagios. In: 4th IEEE International Conference on Software Engineering and Service Science (ICSESS) (2013)
20. RRDtool. http://nmis.sourceforge.net/. Accessed May 2012

Yongseong Cho received the M.S. degree in School of Computer Science and Engineering, Soongsil University, Seoul, Korea, in 2013. He is currently a Ph.D. candidate in School of Computer Science and Engineering, Soongsil University, Seoul, Korea.

Jongsun Choi received the B.S. degree in School of Computer Science and Engineering from Soongsil University, Seoul, Korea, in 2000 and the M.S. degree and Ph.D. degree in Dept. of Computer Graduate School from soongsil university, Seoul, Korea, in 2002 and 2010 respectively. He has worked at Yuhan university as a full-time lecture professor (2008-2009) and at Intelligent Robot Research Center of Soongsil University as a research professor (2010-2011) and at Seoil university as a full-time lecture professor (2012). He is currently an assistant professor of School of Computer Science and Engineering at Soongsil University, Seoul, Korea.

Jaeyoung Choi received the B.S. degree in Department of Control and Instrumentational Engineering, from Seoul National University, Seoul, Korea, in 1984, the M.S. degree in Department of Electrical Engineering, University of Southern California in 1986, and the Ph.D. degree in School of Electrical Engineering from Cornell University in 1991. He has previously worked at Oak Ridge National Laboratory (1992-1994) and University of Tennessee, Knoxville (1994-1995) as a postdoctoral research associate and a research assistant professor, respectively, where he had been involved with the ScaLAPACK project. He is currently a professor of School of Computer Science and Engineering at Soongsil University, Seoul, Korea.

Myungho Lee received his B.S. in Computer Science and Statistics from Seoul National University, Korea, M.S. in Computer Science, Ph.D. in Computer Engineering from University of Southern California, USA. He was a Staff Engineer in the Scalable Systems Group at Sun Microsystems, Inc, Sunnyvale, California, USA. He is currently an Full Professor in the Dept of Computer Engineering at Myongji University. His research interests are in High Performance Computing: architecture, compiler, and applications, with special interest in GPU computing and cloud computing.