Intelligent Information Management, 2010, 2, 398-405
doi:10.4236/iim.2010.27049 Published Online July 2010 (http://www.SciRP.org/journal/iim)

A Unified Monitoring Framework for Distributed Environment

Xiaoguang Wang[1], Hui Wang[2],Yongbin Wang[1]
[1]*Computer School*, *Communication University of China*, *Beijing*, *China*
[2]*Information Engineering School*, *Communication University of China*, *Beijing*, *China*
E-mail: *xgwong*0815@*gmail.com*
Received April 12, 2010; *revised May* 27, 2010; *accepted June* 28, 2010

Abstract

Distributed computing is a field of computer science that studies distributed systems. With the increasing computing capacity of computers it is widely used to solve large problems. Monitoring system is one of the key components in distributed computing. Although there have been varieties of monitoring systems developed by different organizations, it is still a great challenge to monitor a heterogeneous distributed environment in a unified and transparent way. In this paper, we present a unified monitoring framework for distributed environment (UMFDE) with heterogeneous monitoring systems, and then propose a comprehensive method based on the Enterprise Service Bus (ESB) to integrate the monitoring systems in the environment as a unified monitoring system. A representative case study is given to show the feasibility of this framework.

Keywords: Distributed Computing, Unified Monitoring, Enterprise Service Bus

1. Introduction

Distributed computing is a field of computer science that studies distributed systems that consist of multiple autonomous computers or clusters that communicate through a network.

Distributed systems have been in the mainstream of high performance computing with the increasing computing power of computers. They have been primarily used for solving comprehensive application problems such as parallel rendering, weather modeling, nuclear simulations, and data mining.

A scalable and unified resource monitoring system is one of the key components for the effective utilization of the massive computing power in distributed environment. Due to the considerable diversity of resources in the distributed environment, the monitoring of the environment is much more difficult than that of a group of homogenous computers. There have been a variety of monitoring systems developed by different organizations, such as ganglia, Relational Grid Monitoring Architecture (R-GMA). Although some of them can be used in distributed environment, there are still some constraints. For example, we need to replace the existing monitoring systems and deploy the same monitoring system in all the clusters or computer sites. To address such challenges, [1] proposes an integrated framework (UGMF) which is compatible with other monitoring systems in the grid environment, however, there are still some problems not addressed, such as message transformation, content-based routing and flexibility.

In this paper, we present a loosely-coupled, scalable and non-intrusive monitoring framework for distributed environment with heterogeneous monitoring systems. Here we use an integrative method based on Enterprise Service Bus (ESB) to integrate the existing monitoring systems into a unified monitoring framework without redeploying or modifying the systems themselves. The proposed architecture is capable of providing clients with standard-based interfaces, and implementing uniform access to different monitoring data sources through ESB in a transparent fashion. Besides that, we implement a representative case study in a distributed environment to verify the feasibility and scalability of the proposed framework.

The rest of the paper is organized as follows. In Section 2, we introduce several monitoring systems and present an overview of the Enterprise Service Bus. In Section 3, the proposed unified monitoring architecture is discussed in detail. A representative case study based on the proposed architecture is introduced in Section 4. The final conclusion is presented in Section 5.

This research is supported by China National High Technology Program 863 (No.2008AA01A318-0)

2. Related Work

2.1. Monitoring Systems

Resource monitoring is one of the fundamental components in distributed systems. Monitoring data plays a key role in various tasks of distributed computing such as fault detection, performance analysis, performance tuning, performance prediction, and task scheduling. Resources monitored in distributed environment include the CPU load, the memory usage, the network status, the disk usage, and etc.

In this section we present a brief introduction of several existing monitoring systems for distributed systems.

2.1.1. Grid Monitoring Architecture (GMA)

The Grid Monitoring Architecture (GMA) [2] is developed by the Global Grid Forum Performance Working Group. The goal of the architecture is to provide a minimal specification that will support required functionality and allow interoperability.

The Grid Monitoring Architecture consists of three types of components, as shown in **Figure 1**:

- Directory Service: The directory service stores the information about producers and consumers, and accepts requests, supports information publication and discovery.
- Producer: A producer is any component that uses the producer interface to send monitoring data to a consumer.
- Consumer: A consumer is any component that uses the consumer interface to receive monitoring data from a producer.

The GMA architecture supports three interactions for transferring data between producers and consumers: publication/subscription, query/response, and notification. In any case, the communication of control messages and transfer of performance data occur directly between each consumer/producer pair without further involvement of the directory service.

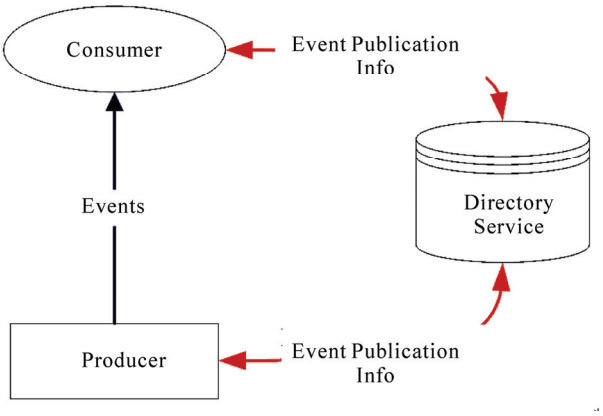

Figure 1. Components of the grid monitoring architecture.

2.1.2. Ganglia

The Ganglia [3] is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters, and relied on a multicast-based listen/announce protocol to monitor state within clusters and uses a hierarchy of point-to-point connections amongst representative cluster nodes to federate clusters and aggregate their state. It has been used to interconnect clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes.

Compared with the Grid Monitoring Architecture, there is not a directory service for Ganglia to store and discover the information of producers and consumers.

Literature [4] presents a structure for monitoring a large set of computational clusters over wide-area networks using Ganglia. It illustrates methods for scaling a monitor network comprised of many clusters while keeping processing requirements low.

2.1.3. Windows HPC Server

Windows HPC Server 2008 [5] (HPCS) is the Microsoft high performance computing (HPC) platform built on Windows Server 2008 64-bit technology. HPCS can efficiently scale to a large number of processing cores and computers. HPCS includes a new, integrated management console that integrates the new network configuration wizard, template-based provisioning based on the Windows Server 2008 Windows Deployment Services technology, a new scheduler, cluster health monitoring at a glance along with built-in diagnostics, and a faster Microsoft Message Passing Interface (MS-MPI) that includes new NetworkDirect support.

HPC Server 2008 provides administrators and users with several tools to effectively monitor IT resources in the cluster, including the Node Management section in the Administration Console, System Center Operations Manager and Microsoft HPC Class Library.

2.2. Enterprise Service Bus

Enterprise Service Bus [6] (ESB) is a software infrastructure that can be used to connect heterogeneous applications and IT resources. It brings flow-related concepts such as transformation and routing to a Service-Oriented Architecture (SOA). It mediates incompatibilities among various applications, coordinates their interactions, and makes them broadly available as services for additional uses.

Despite various definitions used by different users, there are common components in the architecture of an ESB as below:

- Message transformation: To transform messages from one format to another based on open standards like XSLT and XPath.
- Message routing: To determine the destination of an

incoming message based on user-defined rules and logic.

- Security: Mechanisms such as authentication, authorization, and encryption should be provided by an ESB to ensure the integrity and privacy of both the ESB runtime and the messages.
- Management: A management environment is necessary for the configuration of the ESB to be reliable and also to monitor the runtime execution of the message flows in the ESB.
- Transport Management: To convert incoming transport protocols to different outgoing transport protocols.
- Message Broker: ESB acts as a broker between service consumers and service providers.

3. Overview of the Architecture

An overview of the proposed architecture (UMFDE) is shown in **Figure 2**. It is a four-tier framework that consists of the Service Provision Layer, the Service Interface Layer, the Integration Layer and the Application Layer.

Generally, there are two ways to monitor a distributed environment with a number of heterogeneous computers and clusters in the internet. One way is to deploy a new monitoring system to substitute all of the existing systems. This way is simple but impossible in most cases, because owners of the existing systems are in general not the same one. The other way is to integrate the existing systems by their exposed service interfaces without replacing the existing systems. Developers can reuse the monitoring services or develop new services.

The proposed architecture uses the integration method to achieve the goal of monitoring a distributed system.

3.1. Service Provision Layer

Service Provision Layer is comprised of a group of clusters or sites with their monitoring systems. Most of existing monitoring systems provide some kind of services that can be used by the Service Interface Layer.

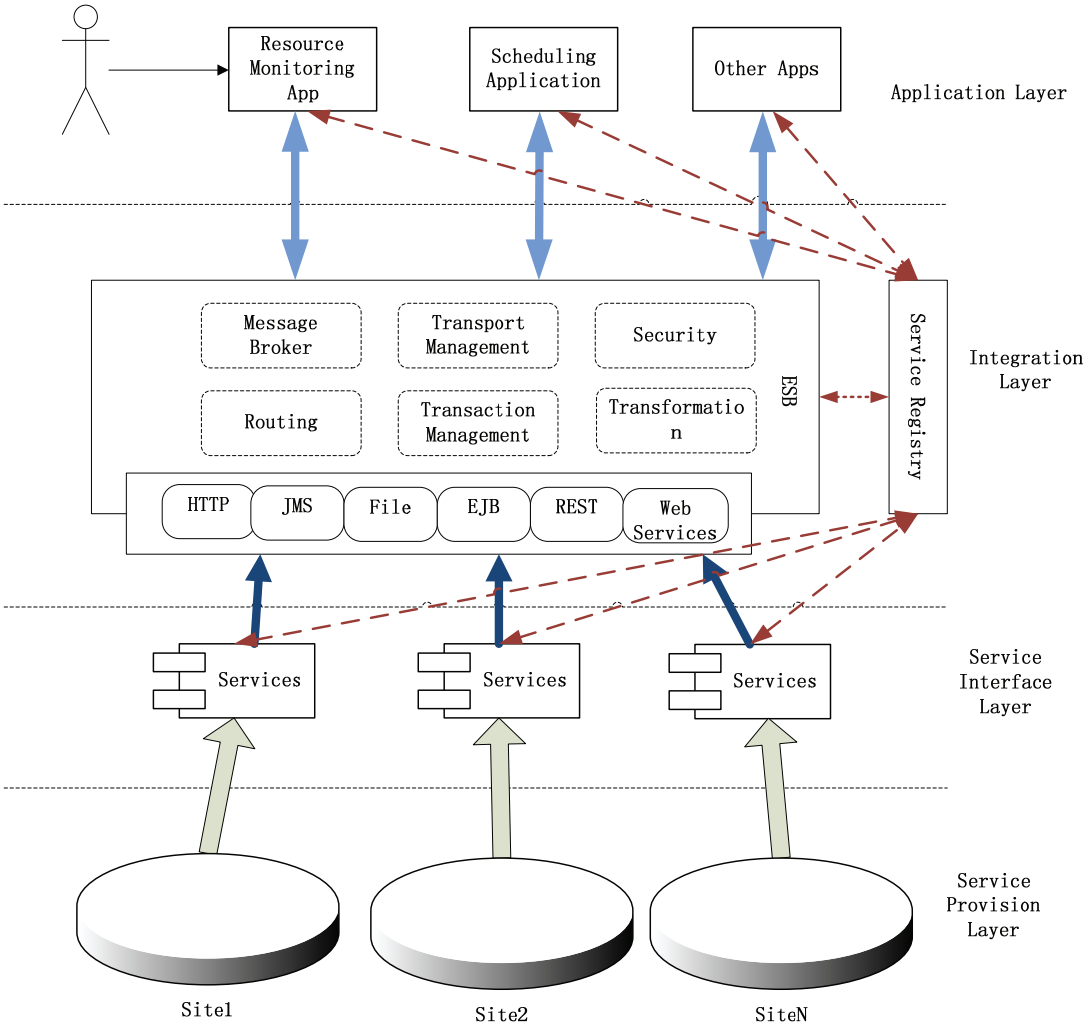

Figure 2. Architecture of the proposed monitoring framework.

3.2. Service Interface Layer

This layer is in charge of the encapsulation of the functionalities of existing monitoring systems into services, which can be connected to components and used in the Integration Layer. In this layer we may develop new monitoring services for clusters or sites without monitoring system. Since services may be offered by different systems, they may be developed based upon various protocols, such as HTTP, JMS, TCP, and SMTP.

3.3. Integration Layer

The aim of this layer is to integrate and thus provide reusability of original monitoring services defined in the Service Interface Layer, and to provide consumers with composite monitoring services. The Integration Layer is made up of two components: a Service Registry, an ESB.

Service Registry is a platform for publishing and discovering information about producers and consumers of the services, including the encapsulated services in the Service Interface Layer, the composite services in the Integration Layer, and consumer services in the Application Layer. However, it will not be further involved in the interactions between the publishers and the consumers.

ESB provides a reliable and scalable infrastructure that connects services of heterogeneous monitoring systems, mediates their incompatibilities, transforms their responses, and makes them broadly available as uniform services for different users. It is responsible for routing requests, invoking original services in Service Integration Layer, integrating and returning results and other fundamental functionalities including content-based routing, message transformation and location transparency. It also enables disparate applications and services to communicate using different protocols through transport protocol conversion.

The main features of the UMFDE are provided by the Integration Layer.

3.4. Application Layer

This layer consists of a variety of applications utilizing the monitoring data provided by the services in the Integration Layer instead of gathering raw data directly from the existing monitoring systems. These applications contain resource management applications, scheduling applications and so on.

The proposed framework supports two interaction types between producers and consumers, namely publication/subscription, and query/response.

4. Implementation

In this section, we discuss the implementation of a case study using the architecture proposed in last section. We will begin with the description of the scenario, and then continue into the details of implementation.

4.1. Overview

The context of the case study is that the client application sends a request to the UMFDE for the response of "the status of computers the client has access to in this environment".

The distributed environment used in this case study is comprised of two clusters, and a group of computers, *site A*. One cluster is Rocks cluster with monitoring system *ganglia*, the other one is Windows HPC Cluster. Site A has no monitoring system.

Here we choose the open source Mule ESB which is widely used by many enterprises to implement the case study. Key elements of Mule ESB are explained below [7-9]:

- Service component: A service component is nothing more than a Java object that is hosted and contained by Mule. These components become services within the Mule instance. Components contain the business logic. Each service is configured using an inbound router collection, a component, and an outbound router collection.
- Endpoint: An endpoint is responsible for the receiving or sending of messages via associated transport.
- Router: Based on the Message Router pattern [8], routers exist to be able to decouple individual processing steps when messages are received from, or are dispatched to, endpoints.
- Transformer: Transformers are used to convert data from one format to another, such as type transformation, message transformation, and so on.

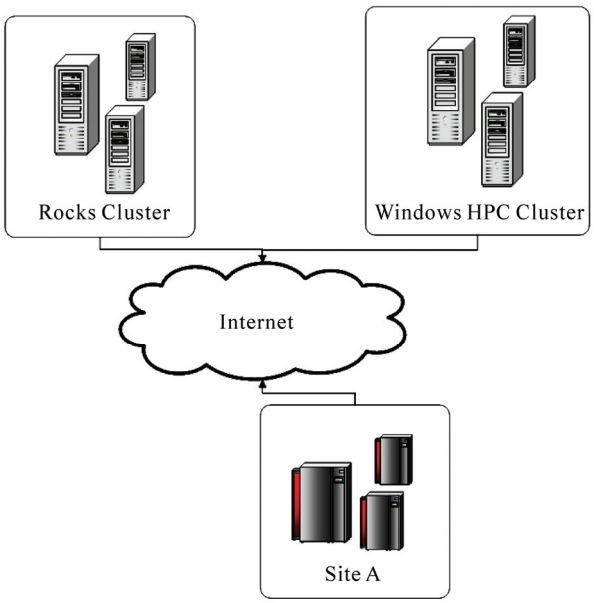

Figure 3. The distributed environment in the case study.

Besides that, we use Mule Service Registry to provide a full complement of registry and repository features, including service and artifact management, governance, lifecycle and dependency management.

The main services used in the case study include:

- *Monitor Broker Service:* Receives requests from and returns responses to client applications. It is hosted by the mule ESB.
- *Authorization Service:* As an external web service, it provides an interface for the Monitor Broker Service to obtain a list of clusters or sites whose monitoring data the current user is authorized to access.
- *Original Monitor Services*: Three services are developed to act as brokers to monitor two clusters and site A.

4.2. Implementation Details

4.2.1. Service Interface Layer

We develop web services for the monitoring systems of the two clusters. We design and develop a monitoring web service for Site A from scratch because there is no monitoring system in Site A. These services will return the status of its machines.

All of the three monitoring services will be published to the Mule Service Registry.

The response of web service of ganglia contains a list of machine's status which includes Id, Hostname, Ip, Mem_free, Cpu_num, cpu_speed, cpu_idle.

The response of web service of Windows HPC Cluster also contains a list of machine's status, but the status has different format and information. The status mainly includes Id, Name, NumberOfCores, CpuSpeed, State, Memory-Size, OnlineTime, NodeGroups, and Reachable. And the web service for Site A has similar response format as these computers are also built on windows platform.

It is obviously that the formats of responses are different, so we propose a unified response format to unify and integrate all of the response messages. We list key properties of the unified format as follows:

OsName: Name of the operating system.
Cluster name: Name of cluster the node belongs to.
ClusterType: Rocks or Windows HPC, Site A.
CpuNum: Number of cpu.
CpuSpeed: Speed of cpu.
CpuLoad: The current cpu load.
MemorySize: The total size of memory.
MemoryFreePercent: The percent of free memory.

4.2.2. Integration Layer

The message flow of the case study is divided into two parts: a request flow for handling the request message, and a response flow for dealing with the response message.

4.2.2.1. Request Flow

As shown in **Figure 4**, at first, client application calls the *MonitorBroker* Service in Mule ESB.

The *MonitorBroker* Service uses a component *msglogger* to log the request message into log files. Here we define a class to implement the log service. Then, the request message is forwarded to *AuthService*.

In the service *AuthService*, we define a Reflection-MessageBuilder component which will try and set the response of the authorization service as a bean property on the request message using reflection. The remaining part of *AuthService* is a filtering router which contains two outbound endpoints.

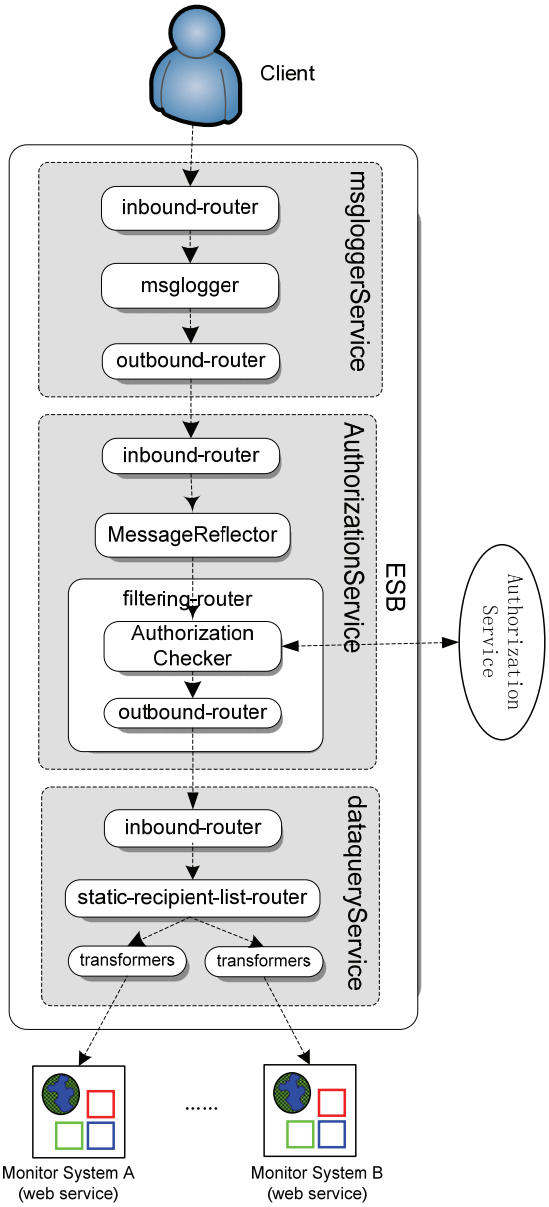

Figure 4. An overview of the Mule configuration of the request flow.

The first outbound endpoint invokes the external authorization service. After receiving the response, the *ReflectionMessageBuilder* will set the list of authorized clusters as a property called *recipients* on the request message. Then the second outbound endpoint will send the updated request message to the *DataQueryService*.

The *static-recipient-list-router* in the *DataQueryService* will apply a transformer to the request message, and then invoke external monitor services whose inbound endpoint is defined in the property *recipients* of the request message.

4.2.2.2. Response Flow

In the response flow, the monitor broker service will aggregate the response messages from different monitor services. And because the response messages are based on different formats, the broker service will transform them with the unified format.

The procedure of the response flow is described as following:

- The monitoring services process requests from the Mule ESB broker service, and return responses to the broker service.
- In the broker service, *transferService* serivce component will record the reponse activity, and then transform reponse messages using the unified format. After that, reponse messages will be forwarded to *aggregationService* component.
- *AggregationService* Service mainly contains a custom asynchronous reply router component, *result Aggregator*. The *resultAggregator* will retrieve the reponse message, and aggregate them into one reponse.
- Finnaly, the broker service will return the reponse to the client application.

4.2.2.3. Application Layer

We build a web application based on ASP.NET for users to monitor the status of the distributed environment. This client application is responsible for controlling user access and users' requests. It also provides multiple views of the status of the environment for users, such as the status of a computer and some cluster.

For this case study, we build a web form to collect users' requests, call the broker service in Mule ESB and display the results.

4.2.2.4. Sequence Graph

The sequence graph as shown in **Figure 6** is described in detail below:

1) The client sends a request to the monitor broker service hosted by the ESB to get the status of computers in the distributed environment;

2) The monitor broker service receives the request, and retrieves the authorized clusters of the client through the third party service *authorizationService*;

3) After receiving the response from *authorization-Service*, the broker service individually calls the monitoring services of the authorized sites;

4) The broker service asynchronously receives the responses from monitoring services, and then integrates them into one response message;

5) ESB sends the integrated message to the client.

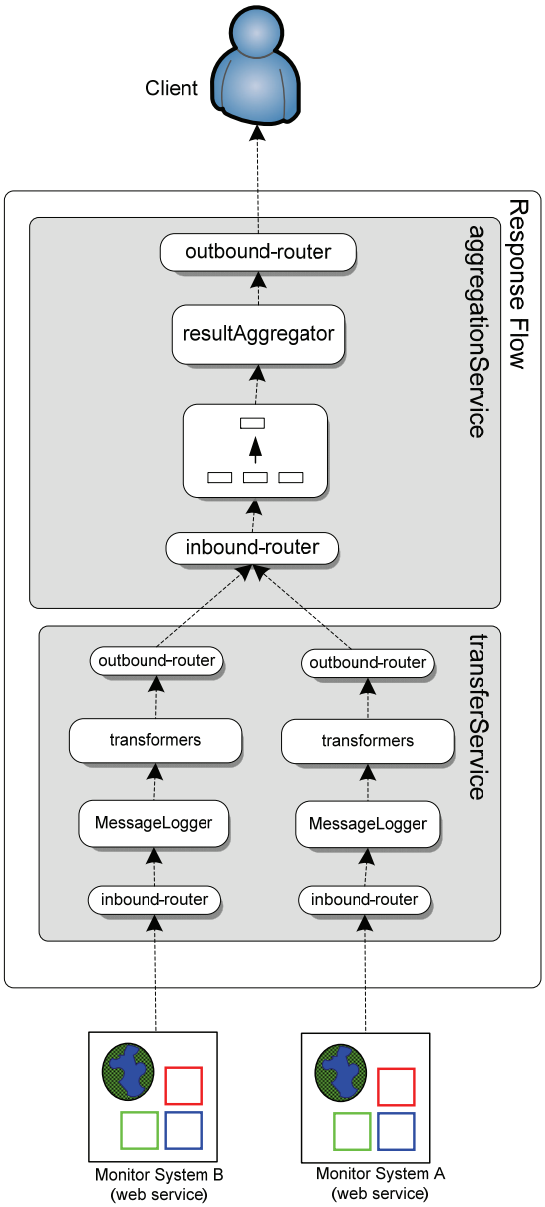

Figure 5. An overview of the Mule configuration of the response flow.

4.3. Summary

As shown in the case study, the proposed architecture is loose-coupled and scalable. It is advantageous in senses of:

- Compared with traditional unified monitoring tech-

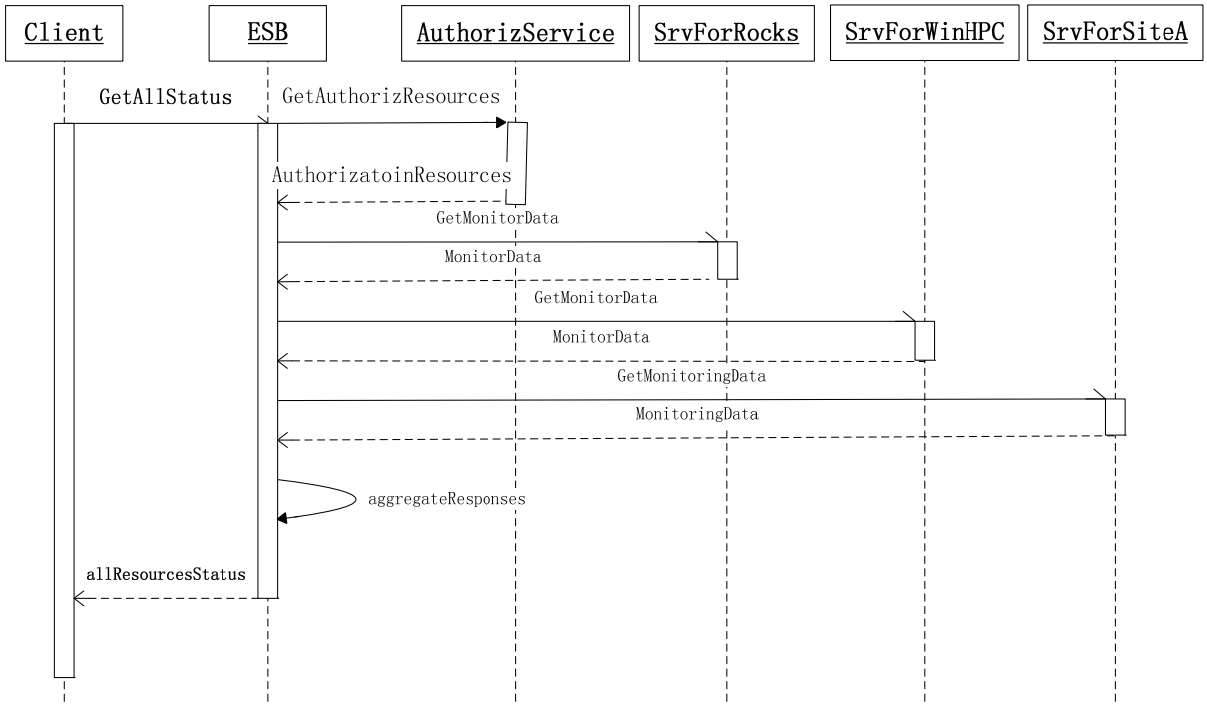

Figure 6. The sequence graph.

nologies, the integration method based on enterprise service bus makes it easier to integrate the existing monitoring systems.

- Reducing the cost of software development and maintenance.
- Rapid response to changing requirements. For example, if requirement changes, we just need modify the broker service hosted by ESB.
- Improving system flexibility, scalability, availability and robustness.

5. Conclusions

This paper proposes a unified monitoring framework for distributed environment that contains clusters or computer sites with heterogeneous monitoring systems. This framework provides a loosely-coupled, scalable and non-intrusive way to monitor a largely distributed environment.

We use an integration method based on Enterprise Service Bus to monitor computers which belong to different clusters in a unified and transparent way. For clusters with monitoring systems, we can reuse or develop services based on the legacy systems. For clusters without monitoring systems, we need develop new monitoring systems and services. By this approach, it is not required to redeploy or modify the legacy monitoring systems. The proposed framework provides standard-based interfaces to clients, and uniform access to different monitoring data sources through ESB in a transparent fashion.

The implemented case study has shown the feasibility and scalability of the proposed framework.

6. References

[1] K. Shen, S. Yang, M. Tian and P. Liu, "Towards a Uniform Monitoring Framework Supporting Interoperability in Grid," *Proceedings of the Fifth International Conference on Grid and Cooperative Computing*, Changsha, China, 2006, pp. 50-53.

[2] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski and M. Swany, "A Grid Monitoring Architecture," Technical Report GWD-Perf-16-1, GGF, 2002.

[3] M. Massie, B. Chun and D. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, Vol. 30, No. 7, 2004, pp. 817-840.

[4] F. Sacerdoti, M. Katz, M. Massie and D. Culler, "Wide Area Cluster Monitoring with Ganglia," *IEEE International Conference on Cluster Computing*, Hong Kong, 2003, pp. 289-298.

[5] C. Russel and S. Crawford, "Windows Server 2008 Administrator's Companion," Microsoft Press, 2008.

[6] X. G. Wang, H. Wang and Y. B. Wang, "A Monitoring

Framework for Multi-Cluster Environment Using Enterprise Service Bus," *International Conference on Management and Service Science*, Beijing, China, 2009, pp. 1-4.

[7] T. Rademakers and J. Dirksen, "Open-Source ESBs in Action," Manning Publications, 2008.

[8] G. Hohpe and B. Woolf, "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions," Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA, 2003.

[9] P. Delia and A. Borg, "Mule 2: A Developers Guide," 1st Edition, Apress, 2008.

