GlobalWatch: A Distributed Service Grid Monitoring Platform with High Flexibility and Usability*

Sheng Di, Hai Jin, Shengli Li, Ling Chen, Chengwei Wang
Cluster and Grid Computing Lab
Huazhong University of Science and Technology, Wuhan, 430074, China
hjin@hust.edu.cn

Abstract

GlobalWatch is a distributed platform to monitor various resources of grid platforms so as to improve the flexibility and usability of grid systems. In order to enhance the flexibility, we make improvements from three aspects. 1) We take advantage of both centralized and decentralized methods in the architecture and adopt a loosely coupled architecture to get monitored data; 2) We integrate GlobalWatch with VO (Virtual Organization) concept so as to make it compatible with grid circumstances. 3) We design a XML format to make GlobalWatch flexible for users to submit and receive information. In order to enhance the usability, we not only integrate various functions into GlobalWatch client application, but also make the visual effect better than some other monitoring platforms. At last, we conclude from the experimental result that monitored nodes in GlobalWatch are indeed loosely coupled and the transmission of map data is the main bottleneck in server end.

1. Introduction

Grid computing enables coordinated resource sharing and problem solving in dynamic, multi-institutional organizations [1]. With the revolutionary impact of large-scale resource sharing and virtualization within both science and industry, grid has become a trend of distributed computing. However, some of the current grid projects (e.g. Javalin [2], DISCWorld [3], etc) have not integrated monitoring modules. As grid could be regarded as a special distributed platform, the corresponding monitoring platform is also distributed.

* This paper is supported by National Science Foundation of China under grant 60433040, ChinaGrid project from Ministry of Education, and CNGI projects under grant CNGI-04-15-7A.

There are a number of standards to evaluate the performance of a monitoring platform, such as throughput, response time, system load [4]. However, high performance is not the only issue for many middleware applications. Other issues, including flexibility, adaptability, ease of use and tools supported could be more important, depending on the specific application [4]. Yet some recent monitoring platforms have not thought enough about these issues, and this makes them not be used smoothly and conveniently.

Considering centralized parsing users' information is convenient to control and the architecture of monitoring resources should be flexible, we adopt both centralized and decentralized mrthods. As communication over the Internet has become increasingly frequent, a universal communication format is significant to the extensibility of a distributed platform. Hence, we design a format with XML (*Extensible Markup Language*) for users to submit information and get response messages. In addition, we integrate GlobalWatch with VO (*Virtual Organization*) [5] concept so as to make it compatible with grid circumstances. VO enables disparate groups of organizations and/or individuals to share resources in a controlled fashion [5]. Even though this concept is widely used in the grid community, there is still no agreement on a common definition in the literature [6].

In GlobalWatch, we organize each VO and its grid users in a hierarchical architecture. From technical angle, we adopt GIS (*Geographic Information Systems*) [7] technology in GlobalWatch. With GIS, we provide a graceful visualization for users to see information about monitored nodes, especially about locations of monitored nodes on a map. In short, the major objective of GlobalWatch is to have a distributed service grid monitoring platform which is flexible, smooth and convenient to use. GlobalWatch can be viewed as a lightweight monitoring platform.

The rest of this paper is organized as follows. Section 2 presents the related work, and section 3

Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing (APSCC'06)
0-7695-2751-5/06 $20.00 © 2006 IEEE

describes our designed architecture and key components. In section 4, we evaluate the performance of GlobalWatch with the testing data. Finally, we conclude in the last section.

2. Related Work

With the widespread use of grid technology, grid monitoring platform is more and more important. In this area, there have been some platforms to monitor distributed resources. However, they have not considered flexibility and usability well enough. We study these issues and make improvements to a certain extent.

Ganglia [8] is a scalable distributed monitoring platform designed for high performance computing systems. Ganglia presents monitoring data by using recursive languages called XML. It also uses RRDTool [9] for data storage and visualization. GlobalWatch is not as complex as Ganglia, but we provide a much better visual representation about information, especially about the locations of monitored nodes on a map. In addition, Ganglia stores historical data into monitored nodes, in contrast, we make a cache on users' computers to enhance the speed of getting historical data.

MonALISA (*Monitoring Agents in a Large Integrated Services Architecture*) [10] provides a distributed monitoring platform. It is based on a scalable DDSA (*Dynamic Distributed Services Architecture*) [11] and implemented using JINI/JAVA and WSDL/SOAP technologies [12][13]. The way to enhance the extensibility of this monitoring platform is similar to our method: loosely coupled sensor service architecture in monitoring layer. Compared with MonALISA, we have two important differences. First, we integrate the roles of producer and consumer [4] so that GlobalWatch is more convenient for users. Second, we integrate VO concept with GlobalWatch to make it compatible with grid environment.

Babylon [14] is a platform used to control the execution of object-based distributed applications. It is implemented by using Java RMI [15]. We implement the communication between users and monitored nodes by using web services [16] which can normally pass through firewalls. Although RMI is a little more efficient than web services (application-to-application), it is not as ubiquitous as web services. Moreover, in our platform, practical details like common data representations are avoided through the use of common textual representations (XML) [17].

MapCenter [18] provides a simple, autonomous and graceful visualization for monitoring. But MapCenter does not collect monitoring data with computing and storage resources. Compared with this monitoring platform, we provide cache for clients to keep a period of time of monitoring data.

3. GlobalWatch Architecture

We design two kinds of architectures in GlobalWatch: VO Architecture and Component Architecture. The difference between them is that the former is logical and virtual one, while the latter is a component-oriented framework.

3.1 VO Architecture

In order to make GlobalWatch compatible with grid environment, we design an architecture to organize VOs, grid users and nodes, shown in Figure 1. We call it VO architecture. In this architecture, grid root owns a number of VOs and each VO owns plenty of grid users. Grid users' children are monitored nodes and these nodes are the endpoints of the whole graph. A grid user may belong to different VOs, and one node may belong to different grid users.

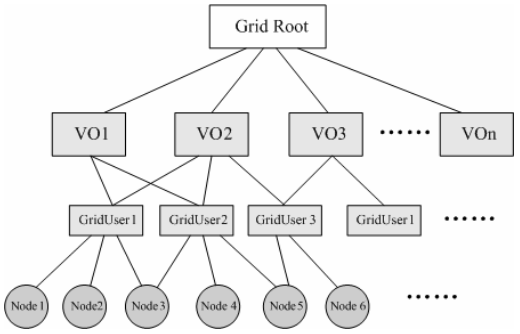

Figure 1. VO architecture of GlobalWatch

According to the VO architecture, we design a set of expressions for searching. They are:

Searching all the VOs (**1**)
Searching all the grid users that belong to a specific VO (**2**)
Searching all the nodes with a specific grid user (**3**)
Fetching a specific node's information (**4**)

All these expressions are formed on proxy server and executed on DB server. It is easy to prove that any part (including VO, grid user and node) of the VO architecture would be searched with the combination of these four expressions. In fact, each VO could be searched with (1); every grid user could be searched with (1) and (2); and each node could be searched with (1), (2) and (3). The last but important expression is used to fetch the information about a given node.

Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing (APSCC'06)
0-7695-2751-5/06 $20.00 © 2006 IEEE

3.2 Component Architecture

The component crchitecture of GlobalWatch is lightweight, shown in Figure 2.

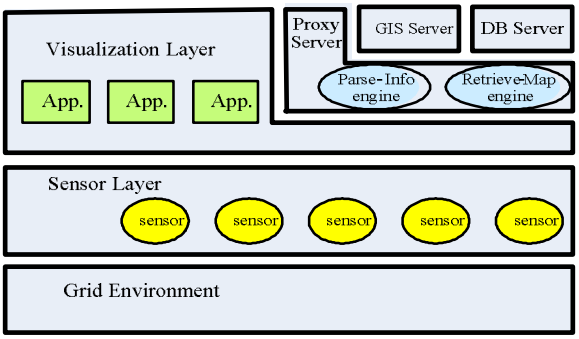

Figure 2. Component architecture of GlobalWatch

In this architecture, Sensor Layer is over grid environment and all the monitored nodes should be deployed with sensors. Each sensor adopts web services technology to receive requests and send responses. Client applications belong to Visualization Layer. Proxy server is a hinge to connect the Visualization Layer and other servers (GIS Server and DB Server). The whole running flow is shown in Figure 3.

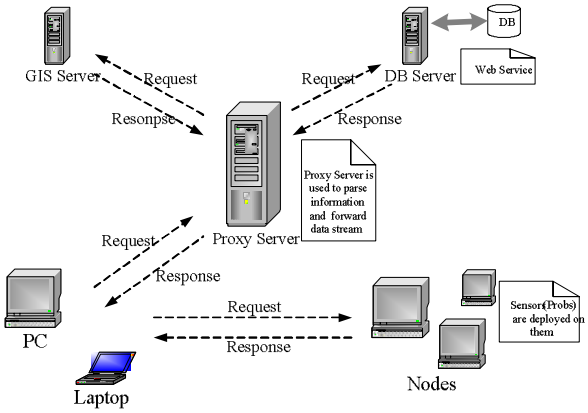

Figure 3. Communications among different modules

First, users should download client applications (or applets) from GlobalWatch website. At this time, proxy server sends out requests to GIS server to get corresponding maps and DB server to get information about monitored nodes. After proxy server receiving responses, it will parse and transmit data to users, and the users will see a map with the display of monitored nodes.

We provide some convenient functions for users. Users could use downloaded client applications (or applets) to send out some particular requests (such as adding monitored nodes, deleting nodes and modifying the information of nodes) to proxy server. Users could also use the graphic beans [19] we made to see the values of metrics (such as CPU Load, Memory Load) and analyze the status of their distributed resources. From this aspect, users are also consumers. In other words, we integrate the functions of administrating and observing into a client application for users.

After downloading client applications, it is a loosely coupled architecture in which users (clients) communicate with monitored nodes (servers/sensors). This architecture will not only reduce the load of proxy server, but also make the performance of getting monitored data unaffected by proxy server. Figure 4 presents this architecture.

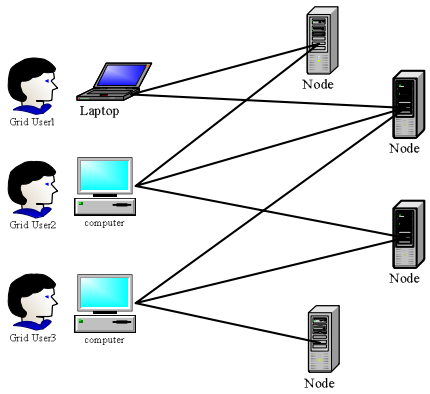

Figure 4. Loosely coupled architecture for retrieving data

Sensor Layer. A sensor is a probe used to get particular data of metrics on a monitored node. We implement the sensors by using web services which includes the operations as follows: estimating whether a target node is on line or not, getting loads of CPU, memory, etc., getting loads of processes by name, and so on. Users could download WSDL file from GlobalWatch website to get services' information. They also implement these services' operations according to their own requirements.

In order to enhance the extensibility and flexibility of GlobalWatch, all the messages transmitted between users and monitored nodes are in SOAP format (i.e. XML format).

Proxy Server. There are two kinds of data streams forwarded by proxy server: information data stream and map data stream. In order to enhance the flexibility of GlobalWatch, we make proxy server with two engines that are endowed with different responsibilities, shown in Figure 2.

- *Parse-Info Engine:* The main responsibility of Parse-Info engine is to parse nodes' information

and store them into DB server. It could be viewed as a registry [20]. We design a unified format for users to submit information. The steps of user's submission include creating tables and inserting data. Parse-Info engine parses the information users submit in XML format to an appropriate format that DB server can accept. It also parses the information transmitted back from DB server and assembles SOAP packages and forwards them to users. We use an element called *Mark* to identify submitted information whether to create a group or to create a grid user. Figure 5, 6, and 7 present the schema of creating tables, inserting data, and for users to get information, respectively. This design avoids tiresome semantic meanings which makes proxy server high extensibility.

```
<xsd:element name="Mark" type="xsd:string"/>
<xsd:element name="FieldName" type="xsd:string"/>
<xsd:element name="FieldType" type="xsd:string"/>
<xsd:element name="TableName" type="xsd:string"/>
<xsd:element name="Field">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:FieldName"/>
      <xsd:element ref="tns:FieldType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Fields">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:Field" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="CreateTable">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:Mark"/>
      <xsd:element ref="tns:TableName"/>
      <xsd:element ref="tns:Fields"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 5. Schema of creating tables

- *Retrieve-Map Engine:* Retrieve-Map engine is mainly used to communicate with GIS server. GIS server receives the requests forwarded by proxy server and performs a set of operations (such as searching map, assembling layers, parsing node points and scales) and subsequently sends responses back. When Retrieve-Map engine receives them, it will forward them to users.

Cache and Storage. In order to enhance the speed of getting historical monitoring data and displaying them, we make a cache on clients.

After downloading client applications (or applets) and starting to run them, some text files will be created on clients' computers to keep a period of time monitoring data that users are interested in.

Each file stores the monitored data of all nodes that belong to a grid user. Each file name consists of a VO's name and a grid user's name. The data streams written into the files are formed by sensors on monitored nodes.

```
<xsd:element name="FieldName" type="xsd:string"/>
<xsd:element name="FieldValue" type="xsd:string"/>
<xsd:element name="TableName" type="xsd:string"/>
<xsd:element name="Item">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:FieldName"/>
      <xsd:element ref="tns:FieldValue"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Items">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:Item" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="InsertData">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:TableName"/>
      <xsd:element ref="tns:Items"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 6. Schema of inserting data

```
<xsd:element name="FieldName" type="xsd:string"/>
<xsd:element name="FieldValue" type="xsd:string"/>
<xsd:element name="Item">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:FieldValue" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Items">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:Item" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="FieldNameList">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:FieldName" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="GetInfoResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:FieldNameList"/>
      <xsd:element ref="tns:Items"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 7. Schema of getting information

Visualization Layer. We provide both Watch Panel and Add-node Panel and they are selected alternatively by a tabbed pane, shown in Figure 8. With the Watch Panel, users can do some operations such as zooming in, zooming out, and translation on a map. On the map, the displayed points stand for monitored nodes and their colors mean different kinds of status. "Virtual

Organization" and "Grid User" are check-buttons and we list the meanings of their compound functions in Table 1. Figure 9 presents monitoring data panel. In this panel, users could click "Refresh" button to refresh data. Each small green square in the snapshot is a time slot.

In order to separate the visualization and its substrate, we adopt notification mechanism (observer pattern) and make a clear distinction among three modules (MVC): Model, Vision, and Control [21]. We design many *producers* in the client application, and each producer is used to pull data from monitored nodes. The canvas of monitored data panel is the corresponding *consumer*. Its responsibility is to receive producers' notifications and displaying data. This design of displaying is convenient for future investigators to extend the codes. For instance, if the pulling-data mode is going to be changed into pushing-data mode, we just need to extend producers.

Table 1. The displayed points on the map about VO and grid user with check buttons

VO	Grid User	What displayed on the map
not selected	selected	all the nodes of the current selected Grid User which belongs to the current VO
selected	not selected	all the Grid Users of the current selected VO
selected	selected	all the nodes of all the Grid Users which belong to the current selected VO
not selected	not selected	nothing but map

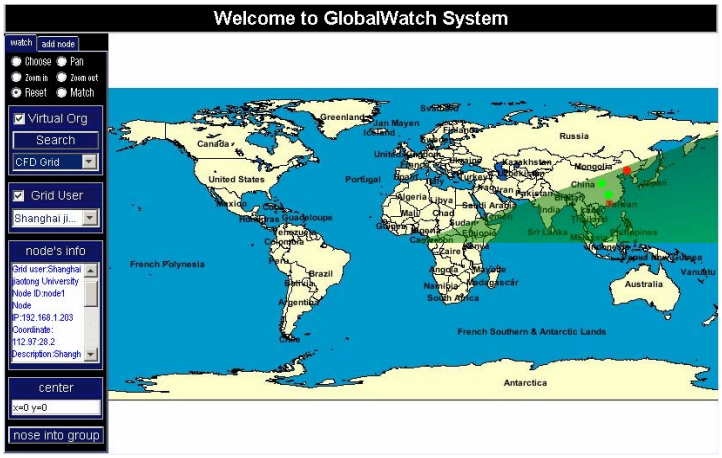

Figure 8. The snapshot of watch panel and map

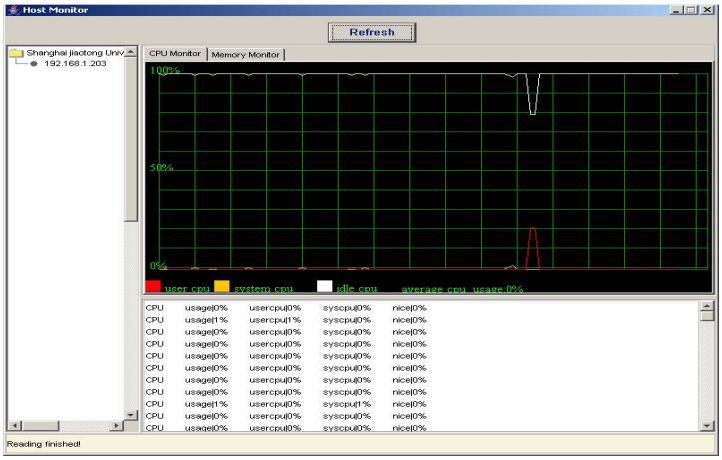

Figure 9. The snapshot of monitoring data

4. Performance Analysis

Table 2 lists the detailed configuration of the testing environment. We use the corresponding testing data to check our platform's bottleneck and prove the loosely coupled architecture mentioned above.

The cost of transmitting maps is so high that it has to be considered seriously. In GlobalWatch, proxy server is a hinge to communicate with other servers (GIS server and DB server), its status is different from the other two servers. In other words, it is essential to compare the influence to the platform's performance caused by transmitting maps and that caused by transmitting information. From Figure 10, we can see clearly the difference between them on diverse environments. When GIS server, DB server and proxy server are deployed on the same computer, we get the shortest time of transmitting data stream (including map data stream and information data stream), that is the best performance. When we put DB server on another computer, the performance is not influenced much. However, when we separate map server, we get the worst performance. Hence, the data stream of transmitting map is a significant factor to affect the performance of the server.

GlobalWatch is flexible, mainly because it is a loosely coupled architecture in which users communicate with monitored nodes. We calculate the average response time in four kinds of status (when monitored nodes die, when processes on monitored nodes die, when monitored nodes recover, and when processes on monitored nodes recover) and present it in Figure 11. Figure 11 shows no matter how the servers are deployed, the time of transmitting data between users and monitored nodes is almost the same. Hence, the performance of getting monitoring data is completely unaffected by proxy server.

5. Conclusions

In order to provide a convenient distributed service grid monitoring platform with high flexibility and usability, we have presented in this paper the design and the implementation of the GlobalWatch platform and analyzed its performance. Binding monitored nodes with a world map is a feature of GlobalWatch, though it more or less causes bottleneck on proxy server. We also integrate the role of producer (adding nodes, deleting nodes, etc.) and that of consumer (observing nodes). In addition, we integrate GlobalWatch with VO concept so as to make it compatible with grid circumstances. By analyzing the performance of GlobalWatch, we prove these monitored nodes are indeed loosely coupled.

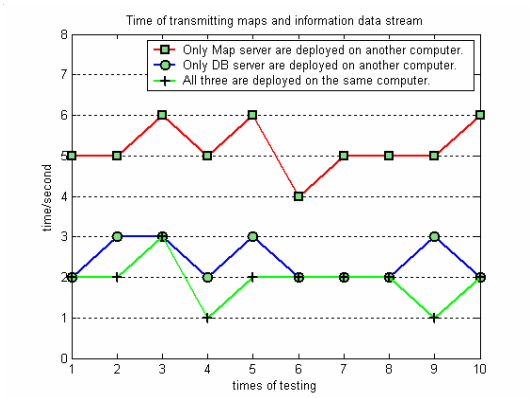

Figure 10. Time of transmitting maps and information

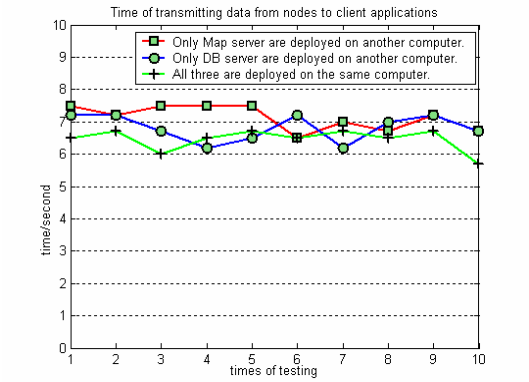

Figure 11. Time of getting monitored data

Table 2. Configuration of The Testing Environment

Servers and nodes	Configuration	Network	OS	Needed Applications
Proxy Server (1 node)	AMD2500+/512DDR333	LAN	Windows XP Or Linux	JDK1.4.2_02 Tomcat 5.0
DB Server (1 node)	AMD2500+/512DDR333	LAN	Windows XP Or Linux	JDK1.4.2_02 Mysql 3.23.57, Tomcat 5.0
GIS Server (1 node)	AMD2500+/512DDR333	LAN	Windows XP	JRun 3.1, JDK1.4.2_02 ArcIMS 4.0, Tomcat 5.0
Monitored nodes (8 nodes)	AMD1700+/256DDR333	LAN	Windows XP Or Linux	JDK1.4.2_02 WebSphere Server

Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing (APSCC'06)
0-7695-2751-5/06 $20.00 © 2006 IEEE

We expect to enhance its performance in our future wok. First, since map data stream is a significant factor causing bottleneck on the server end, we may embed GIS server into client application or make some extra proxy for GIS server. Second, there is no doubt cache will enhance the performance of GlobalWatch [22][23], so we will add cache modules into proxy server, DB server and sensors. Finally, we will try to change the current pulling data mode (active fetching data mechanism) into a pushing data mode (notification mechanism) on sensors.

References

[1] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organization", *International Journal of Supercomputer Applications* 15(3), 2001, pp.200-222.

[2] Javelin project, http://www.cs.ucsb.edu/research/javelin

[3] DISCWorld project, http://www.dhpc.adelaide.edu.au/projects/DISCWorld

[4] H. N. L. C. Keung, J. R. D. Dyson, S. A. Jarvis, and G. R. Nudd, "Performance evaluation of a grid resource monitoring and discovery service for the Grid", *IEE Proceedings: Software*, Vol.150, No.4, 2003, pp.243-251.

[5] I. Foster and C. Kesselman, *The GRID 2: Blueprint for a New Computing Infrastructure*, Second Edition, Morgan Kaufmann, 2004.

[6] S. Andreozzi, N. D. Bortoli, S. Fantinel, A. Ghiselli, G. L. Rubini, G. Tortone, M. C. Vistoli, "GridICE: a Monitoring Service for Grid Systems", *Future Generation Computer Systems Journal*, 21(4), 2005, pp.559-571.

[7] GIS Dictionary, http://support.esri.com/index.cfm?fa=knowledgebase.gisDictionary.search&search=true&searchTerm=GIS

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

[9] The RRDtool time-series data archiving system, http://people.ee.ethz.ch/~oetiker/webtools/rrdtool

[10] H. B. Newman, I. C. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu, "MonALISA: A Distributed Monitoring Service Architecture", *Proceedings of Computing in High Energy and Nuclear Physics (CHEP03)*, La Jolla, CA., 2003, pp.1-8.

[11] A Distributed Server Architecture for Dynamic Services, DRAFT 2.0, 22nd May, 2001, http://www.uscms.org/s&c/reviews/scop/200105/docs/DistributedServicesD2.pdf

[12] JINI technology, http://www.jini.org

[13] SOAP Version 1.2, Part 1: Messaging Framework, http://www.w3.org/TR/soap/

[14] M. Izatt, *Babylon: A Java-based Distributed Object Environment*, Master's thesis, York University, Canada, 2000.

[15] Java Remote Method Invocation, http://java.sun.com/j2se/1.4.2/docs/guide/rmi/index.html

[16] W3C Organizations Web Services definitions, http://www.w3.org/2002/ws/

[17] N. A. B. Gray, "Comparison of Web Services, Java-RMI, and CORBA service Implementations", *Proceedings of Australian Software Engineering Conference (AWSA)*, 2004.

[18] F. Bonnassieux, R. Harakaly, and P. Primet, "MapCenter: An Open Grid Status Visualization Tool", *Proceedings of the 15th ISCA International Conference on Parallel and Distributed Computing Systems (PDCS)*, Louisville, KY, USA, 2002.

[19] Java Beans technology, http://java.sun.com/products/javabeans/reference/

[20] B. Coghlan, A. W. Cooke, A. Datta, A. Djaoui, L. Field, S. Fisher, J. Magowan, W. Nutt1, M. Oevers, M. Soni, N. Podhorszki, J. Ryan, A. J. Wilson, and X. M. Zhu, "R-GMA: A Grid Information And Monitoring System", *Proceedings of the Tenth International Conference on Cooperative Information Systems*, 2003.

[21] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80", *Journal of Object-Oriented Programming*, 1988, pp.26-29.

[22] X. Zhang, J. Freschl, and J. Schopf, "A Performance Study of Monitoring and Information Services for Distributed Systems", *Proceedings of the 7th IEEE International Symposium on High-Performance Distributed Computing (HPDC-7)*, 2003, pp.270-281.

[23] X. Zhang and J. M. Schopf, "Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2", *Proceedings of IEEE IPCCC International Workshop on Middleware Performance (IWMP)*, 2004, pp.843-849.

Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing (APSCC'06)
0-7695-2751-5/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY