Journal of Information Processing Vol.20 No.2 366–377 (Apr. 2012)

[DOI: 10.2197/ipsjjip.20.366]

**Regular Paper**

# Proposal of Grid Monitoring System with Fault Tolerance

Abu Elenin Sherihan[1,a)] Masato Kitakami[1,b)]

Received: May 18, 2011, Accepted: November 7, 2011

***Abstract:*** A Grid monitoring system is differentiated from a general monitoring system in that it must be scalable across wide-area networks, include a large number of heterogeneous resources, and be integrated with the other Grid middleware in terms of naming and security issues. A Grid Monitoring is the act of collecting information concerning the characteristics and status of resources of interest. The Grid Monitoring Architecture (GMA) specification sets out the requirements and constraints of any implementation. It is based on simple Consumer/Producer architecture with an integrated system registry and distinguishes transmission of monitoring data and data discovery logically. There are many systems that implement GMA but all have some drawbacks such as, difficult installation, single point of failure, or loss of message control. So we design a simple model after we analyze the requirements of Grid monitoring and information service. We propose a grid monitoring system based on GMA. The proposed Grid monitoring system consists of producers, registry, consumers, and failover registry. The registry is used to match the consumer with one or more producers, so it is the main monitoring tool. The failover registry is used to recover any failure in the main registry. The structure of a proposed grid monitoring system depends on java Servlet and SQL query language. This makes the system more flexible and scalable. We try to solve some problems of the previous works in a Grid monitoring system such as, lack of data flow and single point of failure in R-GMA, and difficulty of installing in MDS4. Firstly, we solve the problem of single point of failure by adding failover registry to the system. It can recover any failure in Registry node. Secondly, we take into consideration the system components to be easy to install/maintain. The proposed system is combination of few systems and frequency of update is low. Thirdly, load balancing should be added to the system to overcome the message overloaded. We evaluate the performance of the system by measuring the response time, utilization, and throughput. The result with load balancing is better than that without load balancing in all evaluation results. Finally, we make a comparison between the proposed system and the other three monitoring systems. We also make a comparison between the four types of load balancing algorithms.

***Keywords:*** Grid system, monitoring system, GMA, fault tolerance, load balancing

## 1. Introduction

The growing popularity of the Internet and the availability of powerful computers and high-speed networks as low-cost commodity components are changing the way we do computing and use computers today [1]. The interest in coupling geographically distributed (computational) resources is also growing for solving large-scale problems, leading to what is popularly known as Grid computing. In this environment, the security problem is a hot topic in Grid research due to the dynamics and uncertainty of Grid system. The Grid security issues can be categorized into three main categories: architecture issues, infrastructure issues, and management issues [8].

The Grid management is important as the Grid is heterogeneous in nature and may consist of multiple entities, components, users, domains, policies, and stake holders. The different management issues that Grid administrators are worried about are credential management, trust management, and monitoring issues.

The ability to monitor and manage distributed computing components is critical for enabling high performance distributed computing. Monitoring data is needed to determine the source of performance problems and to tune the system for better performance [9]. Fault detection and recovery mechanisms need monitoring data to determine if a server is down, and whether to restart the server or redirect service requests elsewhere. A performance prediction service might use monitoring data as inputs for a prediction model, which would in turn be used by a scheduler to determine which resources to use.

Monitoring is the act of collecting information concerning the characteristics and status of resources of interest. Monitoring is also crucial in a variety of cases such as scheduling, data replication, accounting, performance analysis and optimization of distributed systems or individual applications, self-tuning applications, and many more [8]. The functions of monitoring are correctness checking, performance enhancement, dependability or fault tolerance, performance evaluation, debugging and testing, control or management, and security.

In the Grid, resources are changeful, and the performances of resources are also fluctuating [2]. Therefore, an efficient management of resources and applications running on the Grid has become key, difficulty, and challenge. Moreover, for parallel ap-

[1] Graduate School of Advanced Integrarion Science, Chiba University, Chiba 263–8522, Japan
[a)] sherihan@graduate.chiba-u.jp
[b)] kitakami@faculty.chiba-u.jp

plications on the Grid, an achievement of highly performance is highly dependent upon the effectively coordination of their components. In order to implement robust, available and high performance Grid environment, right and effective monitoring on Grid is a key. A well-designed Grid monitoring system should be able to measure, record, archive, and then publish the performance of Grid resources, status information of Grid applications.

Most existing monitoring systems work with network or cluster systems. There are several research systems implementing the Grid Monitoring Architecture (GMA) [6]: Autopilot, R-GMA, MDS, etc. Autopilot [11] is a framework for enabling applications to dynamically adapt for changing environments. It aims to facilitate end-users in the development of application. R-GMA [12] combines grid monitoring and information services based on the relational model. Although R-GMA is robust, it has three drawbacks: lack of data flow, loss of control message, and single point of failure. MDS is introduced in the next section in details. It has many problems such as too difficult to install.

In this paper, we focus on monitoring management in Grid computing. The proposed Grid monitoring system is also based on the GMA [6]. GMA is the basis of most monitoring system. The goal of GMA is to provide a minimal specification that will support required functionality and allow interoperability.

We design a simple Grid monitoring system. The proposed system components are producers, registry, consumers, and failover registry. The goals of this system are to provide a way for consumers to obtain information about Grid resources as quickly as possible, and to recover any faults in the system. There is no direct relationship between producer and consumer. The monitoring tool is registry. It manages and controls the relationship between all producers and consumers existing in the system. All objects in the system are producers which have the reply of queries, consumers who need some data, Registry which manages the queries and replies, and failover registry. Failover registry is responsible for taking place of main registry in case of failure. In the proposed Grid monitoring system, we observe that there may be overloaded in Registry if the number of requests is large, so load balancing should be added to the proposed Grid monitoring system in order to get better performance.

Load balancing is not a new concept in the server or network space. Load balancing is a technique applied in the parallel system that is used to reach an optimal system condition, which is workloads are evenly distributed amongst computers, and as its implication will decrease the execution time of programs. Load balancing is dividing the amount of work that a computer has to do between two or more computers so that more work gets done in the same amount of time and, in general, all users get served faster. Load balancing can be implemented with hardware, software, or a combination of both.

Load balancing can be static or dynamic [14]. The static load balancing algorithms are Round Robin algorithm, Randomized algorithm, Central Manager algorithm, and Threshold algorithm. Dynamic load balancing algorithms are Central Queue algorithm, and Local Queue algorithm. The type of algorithm to implement is determined by the type of parallel applications to solve. The most suitable algorithm to our system is Central Manager algorithm.

We try to solve some problems of the previous works in Grid monitoring system such as, lack of data flow and a single point of failure in R-GMA, and difficulty of installing in MDS4. We solve the problem of a single point of failure by adding failover registry in the proposed system. Failover registry is responsible for replacing Registry in case of failure. The second problem that we solve is a lack of data flow. We integrate load balancing with the proposed system. Load balancer at Registry divides the load between Registry and failover registry. The third one is difficult to install. We take into consideration the system components to be easy to install/maintain. The proposed system is combination of few systems and frequency of update is low.

This paper is organized as follows. In Section 2, we show some previous work in monitoring management. In Section 3, we explain the details of our monitoring system and how it is work in Grid system. The performance evaluation and comparison with previous work are shown in Section 4. In Section 5, there is a complete comparison between four types of load balancing algorithm. Finally, the conclusions of this paper and the future work are given in Section 6.

## 2. Related Work

There are a few tools developed by different research communities, which feature in monitoring in the Grid. Most of these monitoring systems are under development. Here we just mention some works of Grid monitoring systems.

The Monitoring and Discovery System (MDS) [3] of the Globus Toolkit (GT) is a suite of components for monitoring and discovering Grid resources and services. The latest version of the MDS system is MDS4. MDS4 builds on query, subscription, and notification protocols and interfaces defined by the Web Service Resource Framework (WSRF) and WS-Notification families of specifications and implemented by the GT4 Web Services Core. MDS4 provides two higher-level services: an Index service, which collects and publishes aggregated information about information sources by GIIS (Grid Index Information Service), and a Trigger service, which collects resource information and performs actions when certain conditions are triggered. These services are built upon a common Aggregation Framework infrastructure that provides common interfaces and mechanisms for working with data sources. The advantages of MDS4 are flexibility and scalability. The disadvantages of MDS4 are it can work only with web service, and it is too difficult to install [10].

Ganglia monitoring system [4] is a scalable distributed monitoring system. It provides scalable monitoring of distributed systems in the architectural design space including large-scale clusters in a machine room, computational Grids consisting of federations of clusters, and, most recently, has even seen application on an open, shared planetary-scale application testbed called PlanetLab. The system is based on a hierarchical design targeted at federations of clusters. It relies on a multicast-based listen/announce protocol to monitor state within clusters and uses a tree of point-to-point connections amongst representative cluster nodes to federate clusters and aggregate their state. The advantages of Ganglia are new nodes can be added easily, it is portable,

it has a fine web interface, and it is widely used and can be easily integrated with other monitoring systems. The disadvantages of Ganglia [16] are Database is not scalable, and there is heavy network if there are big numbers of nodes.

Hawkeye monitoring system [5] provides a simple and lightweight way for system administrators to monitor and manage distributed systems. Hawkeye is designed by Condor group and mainly used for monitoring Condor pools. Hawkeyes architecture comprises of four major components: Hawkeye pool, Hawkeye manager, Hawkeye monitoring agent, and Hawkeye module. The advantages of Hawkeye are Ref. [16] Multiplatform system, and possible custom-made sensors. The disadvantages of Hawkeye are poor front-end, and under development.

## 3. Proposed Grid Monitoring System

### 3.1 Overview

Most of Grid monitoring and information service have shortcomings and not easy to use [10]. First, they are too large and too difficult to install, to configure and to deploy. For example, to use MDS4 you need configure third party monitoring tools such as Ganglia or Hawkeye. Second, some functions they provide may be limited to some specific projects; these functions may be useless to another project and may reduce the performance of this project. Finally, some protocols these tools rely on also have defects.

The requirements of Grid monitoring are performance enhancement, dependability, performance evaluation, and management. We design a simpler model after we analyze most of the requirements of Grid monitoring and information service, and implement it. We try to solve some problems of the previous works in a Grid monitoring system such as, flow of data and single point of failure in R-GMA, and difficulty of installing in MDS4. We also have two goals of the proposed system: management and dependability (failure recovery). The proposed Grid Monitoring System is based on the Grid Monitoring Architecture (GMA) [6] as shown in **Fig. 1**.

In order to satisfy the requirement of Grid monitoring, Global Grid Forum (GGF) recommend Grid Monitoring Architecture (GMA) as a Grid monitoring mechanism [6]. The GMA specification sets out the requirements and constraints of any implementation. It is based on simple Consumer/Producer architecture with an integrated system registry and distinguishes transmission of monitoring data and data discovery logically. In GMA, all of monitoring data are events which are based on timestamp for storing and transferring such as, CPU usages, memory usage, thread status and error information. The Grid Monitoring Architecture consists of three types of components: Directory Service (Registry), Producer and Consumer.

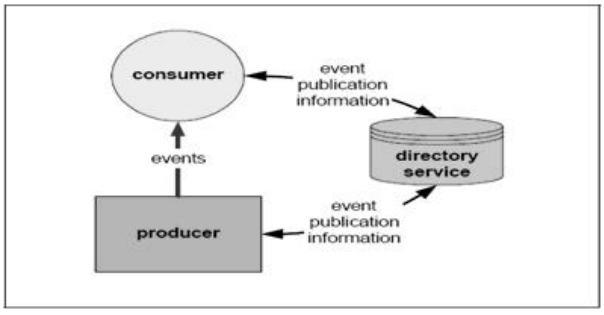


**Fig. 1** GMA components.

The architecture of the proposed Grid monitoring system and the Communications between the Producer and the Consumer is shown in **Fig. 2**. The proposed Grid monitoring system consists of producers (P), registry, consumers (C), and failover registry. The main aim of the proposed system is to provide a way for consumers to obtain information about Grid resources as quickly as possible. It also provides fault tolerance system supported by failover registry. In Fig. 2, the solid line is the normal communication between consumer and registry. The dotted line is the replacement communication in case of registry failure. The structure of the proposed Grid monitoring system depends on java Servlet and SQL query language. Java servlets are more efficient, easier to use, more powerful, more portable, and cheaper than traditional Common Gateway Interface (CGI). Structured Query Language (SQL) is a database computer language designed for managing data in relational database management systems (RDBMS), and originally based upon relational algebra. Users are offered all the flexibility that SQL query language brings. So the system's components are easy to use in any platform and it can solve the problem of single point of failure and flow of data.

### 3.2 Components of Proposed Grid Monitoring System

Producers are the Grid services which register themselves in registry, describe the type and structure of information by SQL CREATE TABLE and SQL INSERT TABLE, and reply to the query of consumer as shown in **Fig. 3**. So the producers in our Grid monitoring system are the source of data. Each producer has an interface and a Servlet. Producer interface communicates with producer Servlet to build the database. The functions that are supported by the producer are creating tables, inserting data into tables, deleting data from tables, and updating data in tables.

Registry acts as a discovery Grid service to find relevant producers matching the query of a consumer. Registry schema consists of four layers: register layer, data layer, service layer, and republish layer. Register layer is responsible for registering all producers and consumers in the system. Data layer as shown in

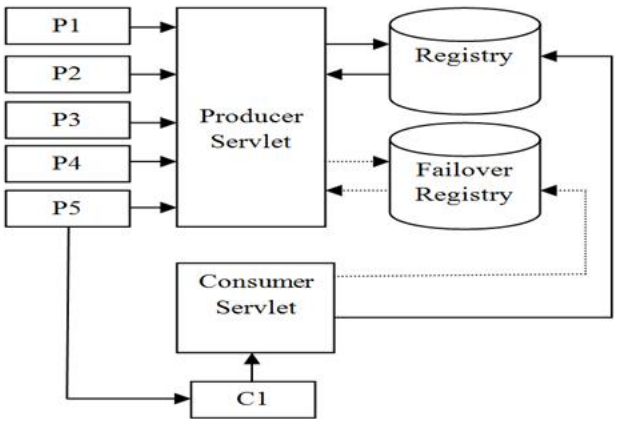


**Fig. 2** Proposed Grid monitoring system.

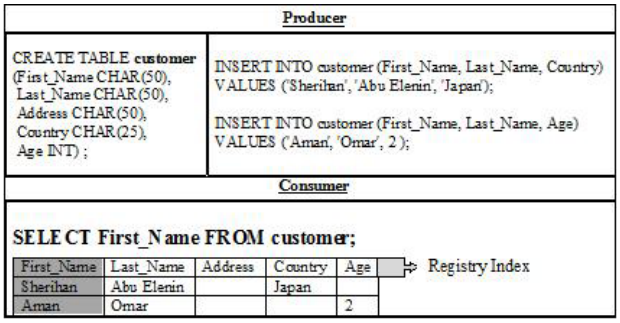


**Fig. 3** SQL example.

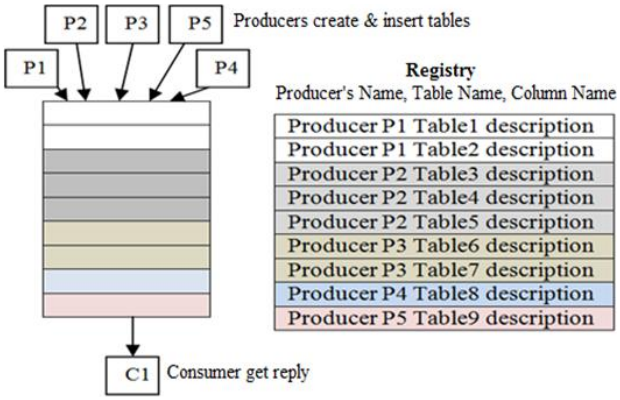


**Fig. 4** Data layer in Registry schema.

**Fig. 4** contains the description of the database exist in all producers. Every Producer enters the index of his tables in Registry schema, and the consumer can get the reply from these indexes. As the example in Fig. 4, the Registry index contains the table name and description of it. For example, if the table "customer" in Fig. 3 exists in Producer1, then data layer in Registry contains Producer1 has 'customer table' with the description First_Name, Last_Name, Address, Country, Age. Service layer and republish layer take request and get the reply, respectively. The functions that are supported by the registry are registering both producers and consumers, adding entry from producers, updating entries from producers, removing entries from producers, and searching about a suitable producer for consumer. The overall purpose of the registry is to match the consumer with one or more Producers. This is achieved by that Producers publish information about themselves, and consumers search through the registry until they find the relevant match and the two communicate directly with each other. The registry is not responsible for the storage of database, but only the index of it. Registry selects the producer that has the reply of consumer's request from Registry schema. For example, as in Fig. 4, if the consumer requests data that exists in P1 Table2 and P3 Table6, Registry firstly sends to P1 because P1 firstly exists in the schema. If it doesn't reply, then Registry sends to P3 in order to send the reply, and so on.

Failover registry is a backup version of all layers in registry. It acts like registry in the situation of failure of registry. It also has all the functions of registry. The proposed Grid monitoring is the only system that has failover registry. Other Grid monitoring systems like R-GMA has a drawback in single point of failure in the directory server. However, Ganglia, MDS, or Hawkeye haven't any solution for failure.

Consumers can be software agents or users who query the Registry to find out what type of information is available and locate the producers that provide such information. The function of consumer is sending request to registry to find data by SQL SELECT statement in browser interface.

### 3.3 The Overall System

Our Grid system is divided into Grid domains (GDs). GD consists of application domain (AD), resource domain (RD), client domain (CD), and Trust Manager (TM). TM's operations consist of Trust Locating, Trust Computing, and Trust Updating. This system was proposed and tested in Ref. [7]. We add another operation to TM. This operation is Registry to manage the relationship between producers and consumers.

Every domain can have any number of producers and consumers. But it has one TM with Registry; this makes management, and one failover registry node; this makes failure recovery. The domain can have any number of nodes that is an intersection with other domains or not.

### 3.4 The Stages of Proposed Grid Monitoring System

(1) Every producer in every domain sends to registry in Trust Manager (TM) to register itself;
(2) TM registers all producers and makes backup of every registration in failover registry that exists in every domain;
(3) Producer Servlet of every domain creates tables and sends the description of the tables (name of columns) to registry in TM;
(4) TM sends a copy of this description to failover registry;
(5) Every producer inserts data in the tables (rows);
(6) Consumer sends to registry in TM to register itself. If the consumer finds registry in TM fails, he/she can send the request to register to failover registry;
(7) Consumer requests registry about some data as SQL SELECT statement. For example:
$SELECT * FROM\ customer\ WHERE\ age > 20$;
(8) Registry (R in TM or failover R) looks up in its data layer indexes and finds the suitable producer who has requested data;
(9) Registry sends to this producer to reply to this consumer's request;
(10) This producer sends the reply to consumer (transfer data).

After analyzing the stages of the proposed Grid monitoring system, we observe that there may be overloaded in Registry if the number of requests is large. We find that message overloaded is the only drawback of the proposed system. So Load Balancing (LB) should be added to the proposed Grid monitoring system to get better performance. It is important in order to get optimal resource utilization, maximize throughput, minimize response time, and avoid the overload. If we add LB in the proposed system, we will get the third requirements of Grid monitoring; performance enhancement.

### 3.5 Central Manager Load Balancing Algorithm

Central manager algorithm is one type of a static load balancing [14]. In a static load balancing algorithm, performance of pro-

**Table 1** Comparison between Grid monitoring systems.

| | MDS2 | R-GMA | Hawkeye | Proposed System |
|---|---|---|---|---|
| Information Collector | Information Provider | Producer | Module | Producer |
| Aggregate Information Server | GIIS (Grid Index Information Service) | None | Manager | None |
| Directory Server | GIIS | Directory Server | Manager | Registry |
| Based on GMA | Yes | Yes | No | Yes |
| Failure Detection | No | No | No | Yes |
| Load Balancer | No | No | No | Yes |

cessors is measured before program execution. The jobs distribution in the parallel system is performed by Registry. Producers then will execute the assigned jobs that are assigned by Registry.

In this algorithm, in each step, central processor will choose a slave processor to be assigned a job. The chosen slave processor is the processor having the least load [13]. The central processor is able to gather all slave processors load information, there of the choosing based on this algorithm are possible to be performed.

The characteristics of Central Manager algorithm are: it has a fault tolerant, it has more forecasting accuracy, there is no overload rejection, it has largely stability, it depends on centralized mode, it is cooperative, there is no process migration, and it has less of resource utilization. Most of these characteristics are suitable with the proposed Grid monitoring system such as, fault tolerant, large stability, centralized, cooperative, and less of the resource utilization. We should make some updates to this algorithm to be suitable to the proposed system. First, there is overloaded at Registry, so we add one condition to the algorithm. If TM finds that number of requests at Registry is more than or equal to 7, then all requests that exceed 7 will be transformed to failover registry. The amount of work will be divided to Registry and failover registry. We choose the number of requests =7, because we have two domains; one with 8 nodes and the other with 7 nodes. So we decide the smallest number of nodes in domain 2 to be suitable with the system. This number is a system example, so if you work with different number of nodes, you should choose the smallest number of nodes in all domains. Second, from the definition of Central Manager algorithm, we don't have master or slaves. To execute the algorithm, the proposed system doesn't depend on master/slaves but it depends on producer/consumer and main monitoring tool Registry. So the algorithm will be executed at producers and Registry not at the master and some slaves as shown in **Fig. 5**. This flowchart can be executed at Registry or failover registry. So load balancing will split the loads at Registry; or failover registry and producers, and make redundancy in the system. This means the overloaded will be split twice at Registry and failover Registry. In Fig. 5, TM decides the load of every producer. If it finds the producer has smallest load, it will send the request to it. But if it finds the producer hasn't small load, the registry will find another producer. Here the codes of Load balancing (LB) of the system:

```
Registry LB code:
IF (Number of requests >7) Then goto Failover
    Registry.
Failover Registry LB code:
IF(Number of requests >7) Then goto Registry.
```

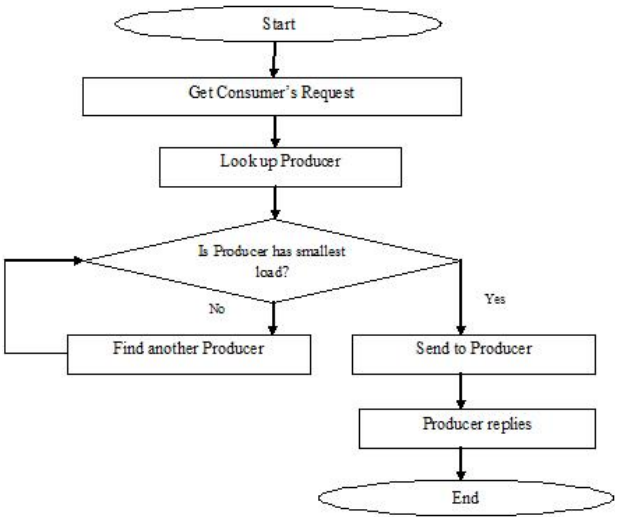


**Fig. 5** The flowchart of modified central manager algorithm.

```
Producers LB code:
IF (producer's load == smallest load)
    Then Get request Else goto the next producer
```

### 3.6 Comparison between Grid Monitoring Systems

We compare the three types of Grid monitoring systems that work in the same manner the proposed system works as shown in **Table 1**. Firstly, MDS and R-GMA share many similarities since both of them provide solutions for Grid information system and driven by basic properties of the Grid environment. They also based on GMA. A Grid information system should enable a large number of users to request the information services and share the information of large-scaled, distributed resources concurrently and efficiently. MDS addressed this requirement by constructing a hierarchical architecture. R-GMA tackles this problem by storing the information about the information producers and the information consumers in a central (can be distributed) repository and making them visible to each other.

MDS and R-GMA are, however, also quite different from each other in many aspects. The differences stem from their design patterns, their architectures, and their underlying technologies. MDS contains an aggregated service component (GIIS) that makes the construction of a hierarchical architecture much easier. On the other hand, R-GMA follows a simple, but flat, architecture design. Although Registry used in R-GMA as the Directory Service can be distributed, there are no concrete protocols to specify how to distribute multiple Registries and what kind of interarchitecture should be built in these Registries to guarantee consistency [15]. Hence, in most cases there is only one Registry,

which indicates the R-GMA system is highly centralized at the point of the Registry. So, Registry can be a central point of failure in R-GMA systems. The components of R-GMA are easy to install. Because the number of nodes in the Grid system always is very large, the administrations need the system to be an easy to install and an easy to maintain. If it isn't easy to install, it will take a long time to build and also a long time to maintain. However, MDS is difficult to install. For example, MDS4 needs to configure the third party monitoring tools.

Secondly, the proposed Grid monitoring system and R-GMA share many similarities since both of them are based on GMA with Java Servlet. The Producer in both systems communicates with a Servlet called Producer Servlet, which registers the table name and the identity and values of any fixed attributes to the RDBMS in the Registry. The Registry's RDBMS holds the information about the Producers. The differences between them are that R-GMA has a single point of failure (Directory service). The proposed Grid monitoring system doesn't have this problem because of the existence of failover registry. Secondly, R-GMA has overloaded at Directory service as described above. The proposed Grid monitoring system doesn't have this problem because it integrates with load balancing system. Thirdly, it also is easy to install the components of the system. To build the system, you need only four steps as described in Section 4.1.

Thirdly, the architecture of Hawkeye is completely different. The architecture of Hawkeye comprises four major components: Pool, Manager, Monitoring Agent, and Module. The components are organized in a four-level hierarchical structure (Pool, Manager, Monitoring Agent, and Module). The main use case that Hawkeye was built to address is that of being able to offer warnings (e.g., high CPU load, low disk space, or resource failure). It also allows for easier software maintenance within a Pool. It has a Manager as an aggregate information server like MDS. The components of Hawkeye are easy to install unlike MDS.

## 4. Evaluation Results

When we work with Grid monitoring system, we found many problems in the previous work such as difficult to install, database is not scalable, lack of data flow, loss of control message, or a single point of failure. We solve most of these problems in the proposed Grid monitoring system. The first problem is a single point of failure. This problem is solved by adding failover registry to recover any failure in Registry. The second problem is difficult to install. We make the system platform is easy as possible as discuss in the following subsections. This is an advantage for the administrations and the developers not for the users. The third problem is overloaded. This problem is solved by adding load balancer at Registry to divide the load between Registry and failover registry. To evaluate the performance of the proposed Grid monitoring system, we pay more attention to the following parameters: response time, utilization, and throughput. In the end, we make a complete comparison between the proposed Grid monitoring system and the conventional systems.

### 4.1 Experimental Platform

Our Grid platform consists of as shown in **Fig. 6**: 1) Hardware Components: Nodes: 5 PCs (Intel Pentium4 2.2 GHz processor, Intel RAM 256 MB) and 10 PCs (Intel Atom 1.66 GHz processor, Intel RAM 2 GB), and Interconnection Network: Gigabit Ethernet 1,000 Mbps. 2) Grid Middleware: Globus Toolkit 4.2.1 (GT4). 3) Software Components: Operating System in all nodes: Linux Fedora 10, and Tools: Programs written in Java, Apache Ant for Java- based build tool, and Microsoft SQL server 2008.

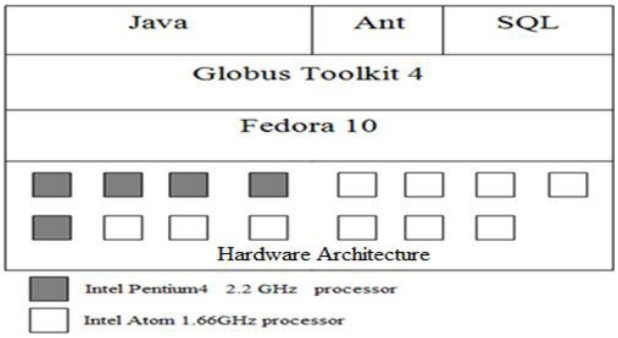


**Fig. 6** Experimental platform.

The proposed system uses hundreds of databases that exist in Chiba University. Every producer has tens of databases about students, staffs, published papers, laboratory contents. The consumer can send any query after entering the system by his username and password. This system depends on trust management besides monitoring management. Trust management is built for security and published in Ref. [7]. Grid monitoring management helps the system for enhancing the performance and recovering failure. There is transparency in the system. Consumers don't know from where the data exist. There is also scalability. Any node can be added or shrunk from the system.

### 4.2 Installation Evaluation

The proposed Grid monitoring system is easy to install and to maintain. This is important for the administrations and the developers of the system for two cases. Firstly, the recovery time should be reduced in the case of node failure. Secondly, when new nodes should be added to the system, they should be easy to install. The steps to build our Grid system are:

Step 1: Installing Fedora 10
Step 2: Installing Java
Step 3: Installing Apache Ant
Step 4: Installing GT4
- Step 4.1: Checking the system
- Step 4.2: Building the Toolkit
- Step 4.3: Setting up security
- Step 4.4: Creating a MyProxy server
- Step 4.5: Setting up GridFTP (File Transfer Protocol)
- Step 4.6: Starting the web services container
- Step 4.7: Configuring RFT (Reliable File Transfer)
- Step 4.8: Setting up GRAM4 (Grid Resource Allocation & Management)

This platform is heterogeneous because it has different hardware. But the software is homogeneous in all nodes. Every node has Linux Fedora 10 and Globus Toolkit 4 and programming interface (Java, Ant, and SQL). This platform is easy to install and to

use. The interface is web based and the user can use it easily.

MDS4 also installs Globus Toolkit 4 but with additional services and with configuring third party monitoring tools. For example, MDS4 needs to Visualize Index Service with WebMDS. It also deploys the Trigger Service to notify interested parties about certain configured changes in the status of the resources that an Index Service is monitoring. After Registering a WSRF service to an Index Service, MDS4 registers any Grid resource via Information Providers. Therefore, MDS4 is not easy to install. It has many components to install. It needs strong administration skills to install/maintain. The steps to build MDS4 [3] are:

Step 1: Installing Linux
Step 2: Installing Java
Step 3: Installing Apache Ant
Step 4: Install SSLeay
Step 5: Install OpenLDAP
Step 6: Installing the Network Time Protocol (NTP)
Step 7: Installing GT4 (Using globus-setup for MDS)
  Step 7.1: Checking the system
  Step 7.2: Building the Toolkit
  Step 7.3: Managing the Configuration Files
  Step 7.4: Setting up security
  Step 7.5: Creating a MyProxy server
  Step 7.6: Setting up GridFTP (File Transfer Protocol)
  Step 7.7: Starting the webservices container
  Step 7.8: Configuring RFT (Reliable File Transfer)
  Step 7.9: Syntax of the interface
    Step 7.9.1: Configuring the Aggregator Sources
    Step 7.9.2: Configuring the Aggregator Sink
  Step 7.10: Development Logging in Java WS Core
    Step 7.10.1: Configuring server side developer logs
    Step 7.10.2: Configuring client side developer logs
Step 8: Setting up the sources of information about:
- Hawkeye
- Ganglia
- WS GRAM
- Reliable File Transfer Service (RFT)
- Community Authorization Service (CAS)

### 4.3 Domains Structure in the Experimental

The system consists of two domains (Domain1 and Domain2). Domain1 consists of 8 nodes: G1, G2, G3, G4, G5, G6, G7, and G8. Domain2 consists of 7 nodes: G1, G2, G3, G4, G5, G6 and G7. Trust Manager (TM) with registry (R1) of Domain1 is existed in G1 and there is back up version called failover registry existed in G8. In Domain2, Trust Manger (TM) with registry (R2) is existed in G1 and there is back up version called failover registry existed in G6. We have two nodes that exist in the two domains; G5 and G7. In the system, always every node is called with its domain name such as G5: Domain2 or G5:Domain1.

Failover registry is an important tool to recover the failure. For example in Domain2, if G1:Domain2 failed, the request will go to G6:Domain2 (failover Registry), and it works all functions of Registry. The algorithm will be:

IF (G1:Domain2 == Fail) THEN Goto (G6:Domain2) ELSE Get the request from consumer.

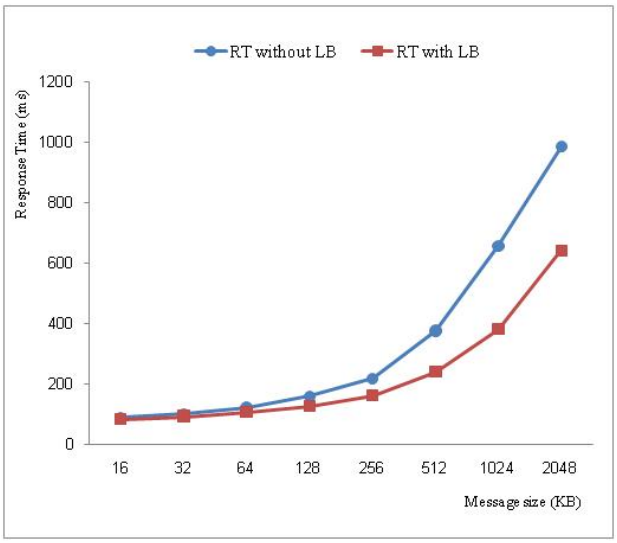


**Fig. 7** Response time of proposed Grid monitoring system.

Finally, failover registry has two functions. One is failure recovery of main Registry in every domain. The other is distributing the query load when Registry has more queries in the same time as discussed in Section 4.

### 4.4 Response Time (RT)

Response time is the average amount of time from the point a consumer sends out a request until the consumer gets the response. We measure response time depending on message size with the fixed number of requests; 15 requests. We measure response time twice; one without load balancing (i.e., there may be overloaded) and one with load balancing. The result is shown in **Fig. 7**.

In case of no load balancing in the system, the result means that when the message size is less than or equal 512 KB, response time is small and increases very simply. But when the message size is more than 512 KB, response time is big and increases very largely. So the performance is always good when the message size is less than or equal 512 KB. Low response time is considered better than high response time. In case of load balancing system, we note that the results are less than of no loading balancing system. This is because the loads of tasks on Registry are divided and the queries are served by Registry and failover registry. So in the same time, many consumers can get the replies and the response time is reduced.

All results that are less than or equal to 512 KB are slightly less than the results of no load balancing. But when the message size is more than 512 KB, response time is largely less than of that without load balancing. So the result of using load balancing is better than that of no load balancing; especially when message size is more than 512 KB.

### 4.5 Utilization

Utilization is the ratio of time a system is busy (i.e., working for us); divided by the time it is available. Utilization is a useful measure in evaluating performance. To get the optimal resource utilization, all the system needs to be always busy; 100% run. Some nodes shouldn't be busy and the other nodes are idle. For all

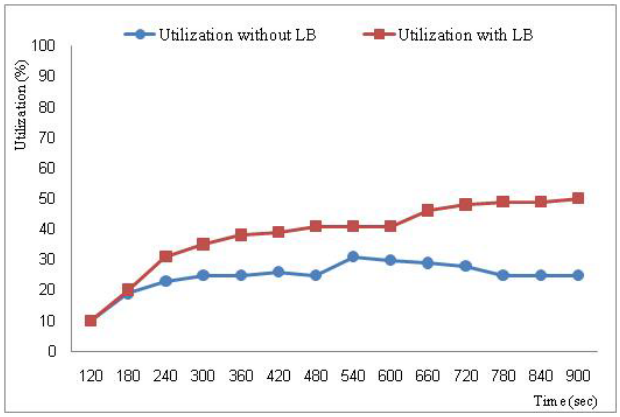


**Fig. 8** Utilization of proposed Grid monitoring system.

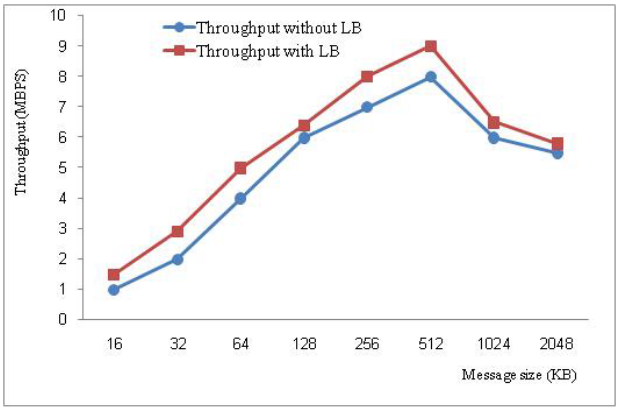


**Fig. 9** Throughput of proposed Grid monitoring system.

the results discussed here, the number of requests is 15 requests. The utilization of producers is measured over time slice; every 60 seconds as shown in **Fig. 8**. We measure utilization twice; one without load balancing (i.e., there may be overloaded) and one with load balancing.

In case of no load balancing in the system, the utilization is an interval from 10 to 31%. In the beginning of executing consumers' requests, Registry only works as monitoring tool. So the system becomes busy slightly. Registry takes one request, finds the suitable producer, and sends to it to transfer data to consumer and so on. So the utilization is increased when the number of requests is increased.

In case of load balancing system, the utilization is an interval from 10 to 50%. In the beginning of executing consumers' requests, Registry and failover registry work as monitoring tools. The requests will be executed faster so the utilization is increased. High utilization is considered better than low utilization.

In the summary, when the system works with LB, all requests will be worked in the same time; in parallelism, so the system will be always busy through time and utilization is increased. When the system works without LB, there may be overloaded and through the time, all these requests can't be replied.

### 4.6 Throughput

Throughput is the amount of data transferred in one direction over a link divided by the time taken to transfer it, usually expressed in bits or bytes per second. People are often concerned about measuring the maximum data throughput rate of a communications link. A typical method of performing a measurement is to transfer a large file and measure the time taken to do so. The throughput is then calculated by dividing the file size by the time to get the throughput in megabits, kilobits, or bits per second. We measure the throughput as a function of data (message size) in Mega Bytes Per Second (MBPS) as shown in **Fig. 9**.

In case of no load balancing in the system, the total information that flows over a given time is not high. When message size is less than or equal 512 KB, the throughput is increased. But when message size is more than 512 KB, the throughput is decreased, so it is bad.

In case of load balancing system, we note that the results are higher than of no loading balancing system, and it is good. High throughput means highly data flow. This is because the loaded on Registry is split into Registry and failover registry. So in the same time, many consumers can get the replies. For example, one consumer can send many requests and all served by Registry and failover registry. So the transferred data will be high more than of no load balancing. When message size is less than or equal 512 KB, the throughput is also increased. But when message size is more than 512 KB, the throughput is also decreased. So we recommended using message size with less than 512 KB when working with the proposed Grid monitoring system to get high performance.

In the summary, when the system works with LB, all requests will be worked in the same time; in parallelism, so the transfer of data will be increased. When the system works without LB, there may be overloaded and not all requests are reserved so the transfer of data will be decreased.

### 4.7 Comparison with Previous Work of Grid Monitoring

Besides the performance study of MDS, R-GMA, and Hawkeye that exist in Ref. [15], we also make a comparison between them and the proposed Grid monitoring system. The comparison is based on their architectures, technologies, and performance. The comparison depends on the scalability of Directory Server with respect to the number of concurrent users. In particular, we compare the performance of the MDS GIIS, the R-GMA Directory server, Hawkeye Manager, and Proposed system Registry.

Since we could employ at most fifteen machines, it was too impossible for us to actually implement hundreds of users. Instead, we used multiple user processes running on each machine. For example, to simulate the traffic generated by 300 users in the real world, we ran 20 traffic-generating processes on each of the fifteen machines. **Figures 10**, **11**, and **12** show the performance results of the four systems.

The response time of MDS2, R-GMA, Hawkeye, and proposed Grid monitoring system are shown in Fig. 10. The response times of four systems are the same when the number of the users is 1 and 10. This causes because the number of users is less than the number of nodes in the system. The response time of Hawkeye and proposed system is the same until 300 users. The results show that R-GMA introduces the highest response time. The highly in response time may be due to the flow of data at the directory server. The results also show that MDS2 introduces the smallest

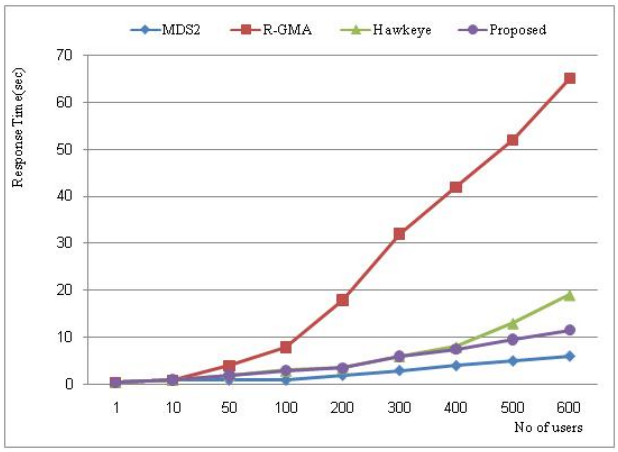


**Fig. 10** Response time of four Grid monitoring systems.

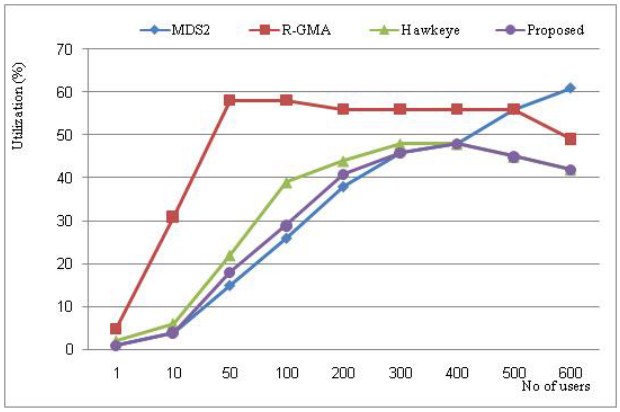


**Fig. 11** Utilization of four Grid monitoring systems.

response time. The response time remains relatively small (less than 6 seconds) even as the number of users increases (up to 600). So, MDS2 is the best in the response time.

Our analysis shows that the response time of MDS2, R-GMA, Hawkeye, and proposed system is increased when the number of users is increased. Low response time is considered better than highly response time. So, MDS2 is the best, and the proposed Grid monitoring system follows it. In the proposed Grid monitoring system, the response time remains small (less than 11.5 seconds) even as the number of users increases (up to 600). R-GMA is the worst in the response time (65 sec at 600 users).

The utilization of MDS2, R-GMA, Hawkeye, and proposed Grid monitoring system are shown in Fig. 11. After 10 users, the utilization on R-GMA rapidly increases, reaching the maximum average of 58% at about 50 users and 100 users. At 200, 300, 400, and 500 users, the utilization of R-GMA becomes stable (56%). At 500 users, the utilization begins to decrease gradually. We believe this occurs because the flow of queries between consumers (users) and producers becomes high after 15 users.

The utilization of the proposed Grid monitoring system equals the utilization of MDS2 at 1 user, 10 users, 300 users, and 400 users. The utilization on the proposed Grid monitoring system rapidly increases, reaching the maximum average of 48% at about 400 users. At 500 and 600 users, the utilization begins to decrease gradually. This occurs because the load balancing disturbs the requests from consumers twice; one at Registry and the other at

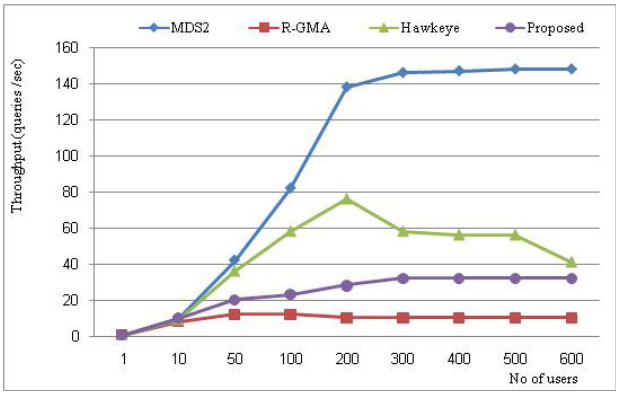


**Fig. 12** Throughput of four Grid monitoring systems.

producers. When the number of users is increased, the utilization is increased until it reaches 400 users. This is the capacity of load balancing system. So if the number of users is exceeded 400 users, the utilization will be decreased.

The utilization of MDS2 is increased as the number of users is also increased. The utilization of Hawkeye is increased until the number of users is 300. At 300, and 400 users, the utilization of Hawkeye becomes stable (48%). At 500 users, the utilization begins to decrease gradually as the proposed Grid monitoring system.

The throughput of MDS2, R-GMA, Hawkeye, and proposed Grid monitoring system are shown in Fig. 12. For MDS, the throughput goes up until the number of users is 500 users. After this, the throughput becomes stable. The throughput of R-GMA is increased until the number of users is 100. And it becomes stable after 200 users. For Hawkeye, the throughput is increased until 200 users. At 300 users, the throughput begins to decrease gradually. The throughput of the proposed Grid monitoring system is increased until 300 users. After this, the throughput becomes stable. This may be due to the fact that the proposed system is based on Java, so must spawn additional threads to handle the user queries. The R-GMA presents the lowest throughput and MDS2 presents the highest throughput.

### 4.8 Summary of the Evaluation Comparison

After the evaluation of the four Grid monitoring systems, we should make a complete comparison between them. A complete evaluation comparison is shown in **Table 2**. MDS2 consists of information provider as information collector, GIIS as directory server, and GRIS as information server. It is based on Grid Monitoring Architecture (GMA). It has some problems such as, single point of failure, and the system organization. MDS is combination of many systems, difficult to install, and frequency of update is high. The results for MDS2 have a good performance when compared with the other systems. It presents a good scalability with respect to the number of users. It has the lowest response time and the highest throughput. Because MDS2 has a hierarchical structure, it supports the dynamic cleaning of dead resources by using a soft-state protocol. In the response time evaluation, it introduces the smallest response time over all the numbers of users. This means that MDS2 can work with large number of users with small response time. In the utilization evaluation, the

**Table 2** Evaluation results of the four Grid monitoring systems.

| | MDS2 | R-GMA | Hawkeye | Proposed System |
|---|---|---|---|---|
| Installation easiness | Difficult | Easy | Easy | Easy |
| The ratio of Response Time to MDS2 | 1 | 9.5 | 2.4 | 1.8 |
| The ratio of Utilization to MDS2 | 1 | 1.4 | 1 | 0.9 |
| The ratio of Throughput to MDS2 | 1 | 10.4 | 2.2 | 3.1 |

utilization is increased when the number of users is increased. It can work with more than 600 users with high utilization. In the throughput evaluation, the throughput is highly increased until the number of users is 300. After 300 users, it is slightly increased.

Hawkeye consists of module as information collector, manager as directory server, and agent as information server. It has some problems such as, single point of failure, and loss of control message. In general, the Hawkeye Agent scaled well with increasing numbers of users. The main cost was communication, when a client connects to the Agent, a new connection must be established because Hawkeye does not allow socket caching from previous queries. Hawkeye has stable load due to the fact that the Agent is single threaded, so no matter how many concurrent users query the Agent, only one can successfully connect to the Agent at a time. In the response time evaluation, it introduces small response time until number of users is 400. After 400 users, it largely increased and the difference between it and MDS2 became high. In the utilization evaluation, the utilization is increased until the number of users is 400. After 400 users, the utilization is decreased. In the throughput evaluation, the throughput is increased until the number of users is 200. After 200 users, the throughput is decreased. From all the evaluation results, the number of users that is suitable to Hawkeye is 200 users. The ratio of Hawkeye to MDS2 in the response time, utilization, and throughput are 2.4, 1, and 2.2 times, respectively.

R-GMA consists of producer as information collector, directory server, producer servlet as information server, and consumer. It has some problems such as, flow of data, loss of control message, and single point of failure. The results for R-GMA have poor performance when compared with the other systems. In all evaluation results, it presents a bad scalability with respect to the number of users. It has the highest response time, the highest utilization and the lowest throughput. The main cause is the query preparation and sending. To retrieve data from each Producer, the client-side sends a new connection request to the Producer, which then builds a new database connection to query the data from that Producer. Creating a new database connection to query each Producer is an expensive operation, especially when a large number of connections exist. The ratio of R-GMA to MDS2 in the response time, utilization, and throughput are 9.5, 1.4, and 10.4 times, respectively.

In summary, the proposed Grid monitoring system introduces a good performance. It is closed to MDS2 in the performance evaluations, such as system utilization and response time. It has a low response time, and a high utilization. Only MDS2 is better than it. The difference of response time between it and MDS2 is very small until 400 users. After 400 users, the difference becomes slightly big but still small. The difference of utilization between it and MDS2 is tiny small until 400 users. After 400 users, the utilization of it is decreased and that of MDS2 is increased. The throughput is increased until 400 users, and after 400 users, it is constant. This may be due to the fact that the proposed system is based on Java, so must spawn additional threads to handle the user queries. To get high performance, the number of users should be less than 400. Of course if the number of nodes in the system increased, the system can serve more than 400 users. The ratio of the proposed system to MDS2 in the response time, utilization, and throughput are 1.8, 0.9, and 3.1 times, respectively.

## 5. Comparison with Previous Work of Load Balancing

We compare the four types of load balancing algorithms which are integrated with the proposed Grid monitoring system. These four algorithms are: Round Robin algorithm, Randomized algorithm, Modified Central Manager algorithm, and Threshold algorithm. The performance of them evaluate by measuring response time and throughput as shown in **Figs. 13** and **14**.

Firstly, we measure response time as a function of the number of users as shown in Fig. 13. When the number of users is one, all algorithms show the same response time. This is because the system is not worked in parallel. When the number of users is 10, the difference between response times from the four algorithms is slightly small. This is because the number of users is less than the number of nodes in the system. In the remaining results, Randomized algorithm introduces the biggest response time and Central Manager algorithm gives the smallest response time. So Central Manager algorithm is also the best algorithm. Threshold and Round Robin algorithms give mediate results. We observe that the response time in all four algorithms until 300 users is small. When the number of users is more than 300, the response time, in all algorithms, is largely increased; especially in Randomized and Round Robin algorithms. If the number of nodes in the system is increased, the system can serve many numbers of users. So we recommended working in this proposed system with the number of users less than 300.

Secondly, we measure the throughput as a function of the number of users as shown in Fig. 14. When the number of users is one, all algorithms show the same throughput. This is because the system is not worked in parallel. When the number of users is 10, Central Manager and Threshold algorithms give the same result, and Randomized and Round Robin give the same throughput. In Central Manager algorithm, throughput is increased with the number of users until 300 users. It is constant after 300 users. In Threshold algorithm, throughput is increased with the num-

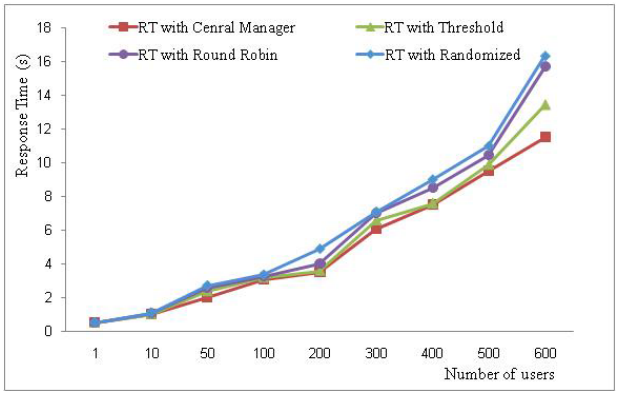


**Fig. 13** Response Time for 4 load balancing algorithms.

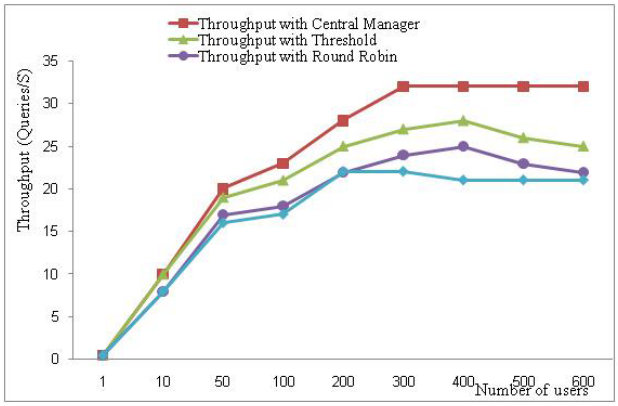


**Fig. 14** Throughput for 4 load balancing algorithms.

ber of users until 400 users. It is decreased after 400 users. In Round Robin algorithm, throughput is increased with the number of users until 400 users. It is decreased after 400 users. In Randomized algorithm, throughput is increased with number of users until 300 users. It is decreased after 400 users. These show the capacity of each algorithm in the number of users. In the summary, Random and Round Robin algorithms have some properties such as, forecasting accuracy, stability, decentralized system, less resource utilization, and static algorithm. Threshold algorithm is different from them in that it has cooperative system. The modified Central Manager algorithm is the best in the evaluation. The most characteristic is that it has fault tolerance. It also has other characteristics such as, no overload rejection, more forecasting accuracy, etc. The performance of the four types of static load balancing is evaluated by measuring the response time, and throughput. Modified Central Manager algorithm is the best. It has introduced good performance. Randomized algorithm has introduced bad results.

## 6. Conclusions and Future Work

The monitoring system in a distributed system is a new topic. Previous works over monitoring system are interested in cluster computing, network, or P2P systems. In Grid systems, most the monitoring system is under development and isn't executed in real projects. In the proposed Grid monitoring system, we focus in the system management by controlling the relationship between the producers, consumers, and registry, and its fault tolerance by adding failover registry in every domain. The overloaded is a big problem in the system, so load balancing should be added. The performance of the proposed Grid monitoring system is evaluated twice by measuring the response time, utilization, and throughput depending on message size of query. The proposed system with load balancing is better than that of without load balancing in all results. We have made a comparison between the proposed Grid monitoring system and other three monitoring systems depending on the number of users; MDS2, R-GMA, and Hawkeye. MDS2 is the best in the performance evaluation, and the proposed system is the second best. Hawkeye introduces mediated results, and R-GMA is the worst in the most results. We also compare the four types of load balancing algorithms. The modified Central Manager algorithm successes in reducing response time and increasing throughput.

The proposed Grid monitoring system solved three problems of the previous Grid monitoring systems. For future work, we should solve the remaining problems of the previous monitoring systems such as loss of control message, heavy network if there are big numbers of nodes, and poor front-end.

**Acknowledgments** The authors are grateful to Prof. Hideo Ito for discussions and advice.

### Reference

[1] Buyya, R., Chapin, S. and DiNucci, D.: Architectural Models for Resource Management in the Grid, *GRID 2000 1st IEEE/ACM International Workshop*, Bangalore, India (2000).
[2] Yao, Y., Fang, B., Zhang, H. and Wang, W.: PGMS: A P2P-Based Grid Monitoring System, *3rd International Conference of Grid and Cooperative Computing* (*GCC 2004*) China (2004).
[3] Globus Toolkit, available from ⟨http://www.globus.org/⟩.
[4] Massie, M.L., Chun, B.N. and Culler, D.E.: The ganglia distributed monitoring system: Design, implementation, and experience, *Parallel Computing*, Vol.30, pp.817–840 (Aug. 2004).
[5] Hawkeye, available from ⟨http://www.cs.wisc.edu/condor/hawkeye/⟩.
[6] Tierney, B., Aydt, R., Gunter, D., et al.: A Grid Monitoring Architecture (2004), available from ⟨http://www-didc.lbl.gov/GGF-PERF/GMAWG/papers/GWD-GP-16-2.pdf⟩.
[7] Elenin, S.A. and Kitakami, M.: Trust Management of Grid System Embedded with Resource Management System, *IEICE Trans. Inf. Syst.*, Vol.E94-D, No.1, pp.42–50 (2011).
[8] Chakrabarti, A.: *Grid Computing Security*, 1 edition, pp.33–45, Springer (2007).
[9] Tierney, B., Crowley, B., Gunter, D., Holding, M., Lee, J. and Thompson, M.: A Monitoring Sensor Management System for Grid Environments, *Cluster Computing*, Vol.4, No.1, pp.19–28 (Mar. 2001).
[10] Pei, W., Chen, Z., Feng, C. and Wang, Z.: Design and Implementation of a Plain Grid Monitoring and Information Service, *5th IEEE International Symposium on Network Computing and Applications* (*NCA'06*), pp.277–284 (2006).
[11] Ribler, R.L., Vetter, J.S., Simitci, H. and Reed, D.A.: Autopilot, adaptive control of distributed applications, *Proc. 7th IEEE Symposium on High-Performance Distributed Computing*, pp.172–179 (1998).
[12] Bhatti, P., Duncan, A., Fisher, S.M., Jiang, M., Kuseju, A.O., Paventhan, A. and Wilson, A.J.: Building a robust distributed system: Some lessons from R-GMA, *International Conference on Computing in High Energy and Nuclear Physics* (*CHEP '07*), Victoria, Canada (Sep. 2007).
[13] Rahmawan, H. and Gondokaryono, Y.S.: The Simulation of Static Load Balancing Algorithms, *International Conference on Electrical Engineering and Informatics*, Malaysia (2009).
[14] Sharma, S., Singh, S. and Sharma, M.: Performance Analysis of Load Balancing Algorithms, *Academy of Science, Engineering and Technology*, No.38, pp.269–272 (Feb. 2008).
[15] Zhang, X., Freschl, J.L. and Schopf, J.M.: Scalability Analysis of Three Monitoring and Information Systems: MDS2, R-GMA, and Hawkeye, *Journal of Parallel and Distributed Computing*, Vol.67, pp.883–902 (2007).
[16] Škiljan, Z. and Radić, B.: Monitoring Systems: Concepts and Tools, *The Six CARNET Users Conference* (2004).

**Sherihan Abu Elenin** received her B.Sc. from the faculty of computers and information, Mansoura University, Egypt in 2002. She is currently pursuing a doctoral degree at Chiba University, Chiba, Japan. Her scientific interests include parallel processing, distributed computers, cluster systems, and grid computing. She is a student member of IEEE.

**Masato Kitakami** received his B.E. degree in electrical and electronic engineering, M.E. degree in computer science, and Dr. Eng. degree all from Tokyo Institute of Technology, Tokyo, Japan in 1991, 1993, and 1996, respectively. He joined Department of Electrical and Electronic Engineering, Tokyo Institute of Technology in April 1996 and moved to Department of Information and Image Sciences, Chiba University in December 1999. From April 2001 to March 2003, he is with VLSI Design and Education Canter, the University of Tokyo. He has been with Graduate School of Advanced Integration Science, Chiba University since April 2007 and is now associate professor. Dr. Kitakami received the Young Engineer Award from an IEICE in 1999. His research interests include error control coding, dependable paralell/distributed systems, error control in data compression. He is a member of IEICE and IEEE.