

A Performance Study of Two Grid Information Systems: Measurement, Analysis and Comparison

By
Xuehai Zhang

Department of Computer Science
University of Chicago

Advisor: Ian Foster
Department of Computer Science
University of Chicago

In partial fulfillment towards Master in Science,
University of Chicago

A Performance Study of Two Grid Information Systems: Measurement, Analysis, and Comparison

Xuehai Zhang
Department of Computer Science
University of Chicago
Chicago, IL 60637

hai@cs.uchicago.edu

ABSTRACT

Performance of the information service has a critical effect on the functionalities and performance of other software components and services in a Grid. Very few results have been published, however, that quantitatively study the performance of the Grid information systems.

In our work, we studied the performance of two well-accepted Grid information systems, the Monitoring and Discovery Service (MDS) and the Relational-Grid Monitoring Architecture (R-GMA), in a quantitative way. For each of them, we designed a set of experiments addressing different performance topics and ran these tests against the system deployed in a small-scaled testbed. By analyzing the performance data collected in these experiments, we not only obtained a quantitative view of the behavior of both MDS and R-GMA in different situations but also gained a good understanding of the performance limitations and their causes. In addition to the individual performance study, we also made a performance comparison between the two systems based on the experiments addressing similar performance topics in both of them.

Our work can be a good reference for the developers of the MDS and R-GMA and for the researchers working on the future information systems in the Grid.

Keywords: *Grid, Grid Information Systems, Performance Study.*

1. Introduction

The Grid [1] has emerged as a new infrastructure to support distributed computing and large-scaled resource sharing. One important component for Grid computing is the information system [2]. It provides the information services to support the discovery and monitoring of the resources in the Grid. The information collected by a Grid information system is used in a variety of Grid activities, for example, Grid applications use it to plan and adapt their behaviors.

Studying the performance of the Grid information systems is important because it greatly affects the performance and the behaviors of other dependent Grid components. Yet very few results have been published that quantitatively study the performance of the current information systems in the Grid. The goal of our work is to achieve a quantitative knowledge about the performance of the current information systems and their performance limitations; such knowledge will help in performance tuning, as well as in design and implementation of future information systems in the Grid.

In this paper, we report on a joint-investigation of two information systems the Monitoring and Discovery Service (MDS) [2][3] of the Globus Project™ and the Relational-Monitoring Architecture (R-GMA) [4][5] used in the European Datagrid. R-GMA is an implementation of the Grid Monitoring Architecture (GMA) defined by Global Grid Forum (GGF) [6]. The selection of MDS and R-GMA is based on two considerations. One is that both systems are in current use in Grid projects; the other consideration is that their different architectures and underlying technologies may have different effects on their performance.

In our work, we first present a list of topics for each of the MDS and the R-GMA addressing their scalability performance and the overhead of their host environments. For MDS, the topics include the effect of too many users on the performance of the Grid Resource Information Service (GRIS) [2] and the Grid Index Information Service (GIIS) [2], and the impact on the performance of a GIIS of choice of data caching. For R-GMA, we are interested in the scalability and efficiency of the Registry [4] (Directory Service [7]) with respect to Consumers [7] and Producers [7]. We believe the by addressing these topics we will gain valuable indexes to evaluate the performance of both MDS and R-GMA. We also combine these performance topics with consideration of user access patterns observed in the Grid. In our study, we focus on a highly used user-access pattern, that is, a large number of Grid users requesting the same service from an information system at the same time.

To address these topics, we designed a set of experiments for each system and then carried out these experiments against the system deployed in a small-scaled testbed at Argonne National Laboratory. We analyzed the performance data we collected in each experiment and obtained not only a quantitative view of both MDS and R-GMA's performance under different situations but also insight into the factors limiting the performance. This information provides constructive suggestions for performance improvements for both systems. We include here a performance comparison between MDS and R-GMA based on the experiments we have done.

The rest of the paper is organized as follows. In Section 2, we provide a general background about the Grid, the MDS, and the R-GMA. Section 3 describes the performance topics we are interested in for both systems. In Section 4, we present the experimental environments and the performance metrics used in the later experiments. Section 5 discusses the details of the experiments we ran for both MDS and R-GMA. For each experiment, the experimental configuration is presented and the performance results are discussed. Section 6 presents a comparison between MDS and R-GMA. Section 7 summarizes our conclusions and briefly discusses future work.

2. Background

In this section, we first introduce the Grid and the Grid information services. We then provide the background of the MDS and the R-GMA information systems.

2.1 The Grid and Grid Information Services

Computational approaches to problem solving have been employed in nearly all scientific fields. Several challenging problems remain, however, such as the NCSA's simulation of a rotating star model, which exceed the available capacity at any single computing site today. In order to provide the computer power needed for very large scientific problems, the concept of the computational Grid was developed. A computational Grid is a hardware and software infrastructure to provide dependable, consistent, pervasive, and inexpensive access to high-end computational resources. Examples of the software infrastructure to make Grid computing realities include the Globus Toolkit™ [8] and the Condor system [9].

To achieve the goals of Grid computing and resource sharing, a Grid defines and provides a set of services to handle a variety of tasks, such as obtaining information about Grid components, locating and scheduling resources, communicating, accessing code and data, monitoring Grid performance, authenticating users and resources, and ensuring the privacy of communications. Among these services, the Grid information services are of primary importance. The Grid integrates numerous distributed resources, services, computations, users, and so forth as the participants to form a virtual organization (VO) [10]. Each participant, however, has little or no knowledge of the other participants in the same VO. The lack of such information presents a big obstacle to the functionalities and performance of most VO participants because each highly depends on one or more other participants. The Grid information services are designed to solve this problem. They enable the participants of a Grid to be aware of the existence and characteristics of the others, and they provide mechanisms to discover, monitor, and retrieve information from all the Grid participants.

2.2 The Globus Project and MDS

The Globus Project is a multi-institutional research and development project that seeks to enable the construction of computational Grids providing pervasive, dependable, and consistent access to high-performance computational resource despite geographical distribution of both resources and users. The Globus Toolkit™ is a software package that provides a set of services that can be used either independently or together to develop useful Grid application and programming tools. The core services include the following:

- The *Globus Resource Allocation Manager (GRAM)* provides resource allocation and process creation, monitoring, and management services.
- The *GridFTP* defines a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks.
- The *Grid Security Infrastructure (GSI)* provides a single-sign-on, run-anywhere authentication service.
- The *Monitoring and Discovery Service (MDS)* is part of an extensible Grid information service that combines data discovery mechanisms with the Lightweight Directory Access Protocol (LDAP).

As the Grid information system in the Globus Project, MDS uses an extensible framework for managing static and dynamic information about the status of a computational Grid and all its components: networks, compute nodes, storage systems, instruments, and so on. MDS also employs the Lightweight Directory Access Protocol [11][12][13] as a uniform interface and backend to such information.

The architecture of MDS, which is depicted in Figure 1, comprises two basic entities: a standard, configurable information provider framework called a Grid Resource Information Service (GRIS) and a specialized aggregate directory service called a Grid Index Information Service (GIIS). A GRIS provides a uniform means of querying resources on a computational Grid for their current configuration, capabilities, and status. A GRIS is a distributed information service that can answer queries about a particular resource by directing the query to an information provider deployed as part of the Globus services on a Grid resource. A GRIS can be configured to register itself with a GIIS so that the GIIS can pass on information about the machine to others. In MDS, the machine receiving the registration data is denoted as “registrar”; the “registrant” represents the machine sending the registration data.

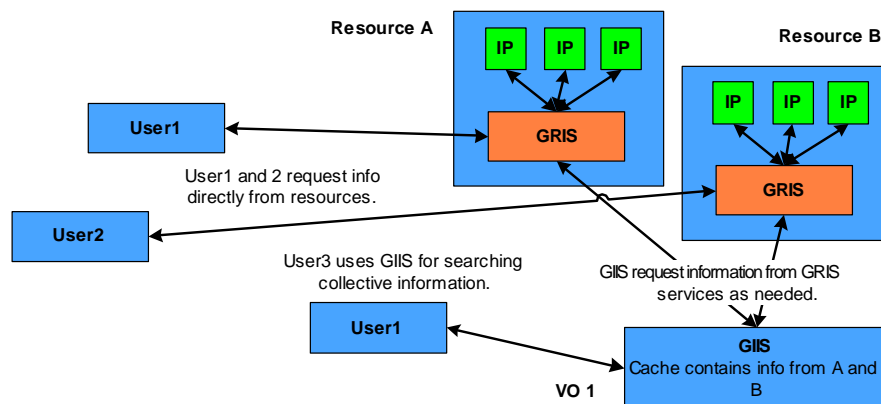


Figure 1: MDS Architecture Overview. The figure depicts a two-level MDS hierarchy, where the GRIS at resource A and B register to the GIIS of VO 1. User can query GIIS to discover collective information (such as User3) and/or query GRIS to obtain information about the individual resource (such as User1 and User2).

A GIIS provides a means of combining arbitrary GRIS services to present a coherent system image that can be explored or searched by Grid applications. A GIIS thus provides a mechanism for identifying resources of particular interest. While a typical configuration is to have a GIIS act as a server to one or more GRIS clients, a GIIS can be a client, a server, or both, depending on its hierarchical relationship to other host machines.

2.3 GMA and R-GMA

Grid Monitoring Architecture (GMA) is an architecture of monitoring components that specifically addresses the characteristics of Grid platforms. It is proposed by the Grid Performance Working Group [6] of the Global Grid Forum. The goal of GMA is to provide a minimal specification that will support required functionality and allow interoperability.

The GMA as shown in Figure 2 consists of three components: Consumers, Producers, and a Directory Service, or Registry. Producers register themselves with the Directory Service, which may itself be distributed, and describe the type and structure of information they want to make available to the Grid. Consumers can query the Directory Service to find out what type of information is available and to locate Producers that provide such information. Consumers can also register themselves to the Directory Service by publishing the types of information they consume. Thus a Producer can find the Consumers who are interested in the data it publishes. Once the information is known, the Consumer can contact the Producer directly to obtain the relevant data. However, GMA does not constrain any of the protocols nor the underlying data model.

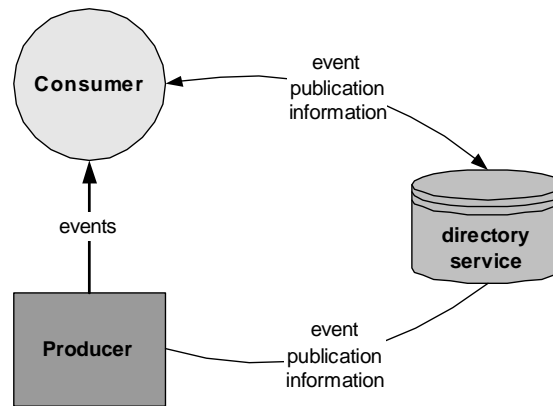


Figure 2: Grid Monitoring Architecture Components

Relational-GMA (R-GMA) is a GMA implementation in Java using a Relational Data Model [14]. It is developed by European DataGrid and used not only as a monitoring system but also as the basis for the information system of the DataGrid. R-GMA strictly follows the GMA proposal but uses the term “Registry” instead of the term “Directory Service”. The implementation of R-GMA is based on Relational Database Management System (RDBMS) and Java Servlet technologies [15][16][17]. Figure 3 illustrates the R-GMA components. To publish its existence and the information it generates, a Producer simply announces a table name and the row(s) of a table. Behind the scenes, the Producer communicates with a servlet called ProducerServlet, 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 registered table name, the identity, and the values of those fixed attributes) and the descriptions of the Producers’ tables. Consumers can issue SQL queries against a set of supported tables (the names and attributes of which can be consulted). Behind the scenes, the query goes to a servlet named ConsumerServlet that analyzes the query and uses the registry to find suitable Producers of information. Then new queries to retrieve the information are generated automatically and sent to discovered producers, and the ConsumerServlet acting on behalf of the Consumer processes the results. The user servlets (ProducerServlet, ConsumerServlet, and so on) do not have to run on the same machine where the Producer or Consumer is.

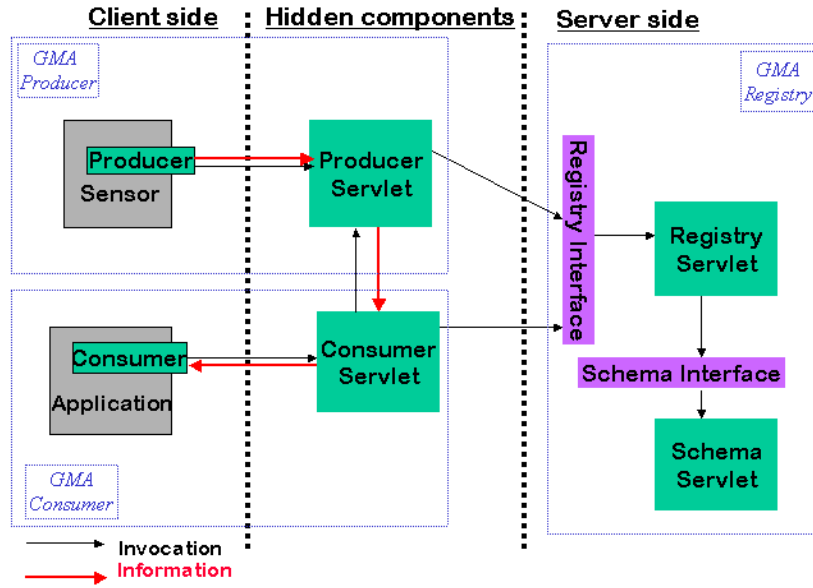


Figure 3: R-GMA Components

3. Performance Topics for MDS and R-GMA

In this section, we discuss the performance topics we investigated for both MDS and R-GMA. The topics for each system address the system's scalability performance while taking into consideration the system overhead and user access patterns. A primary user access pattern we considered is that of a large number of users accessing a Grid information system at the same time and requesting the same services provided by that system. This pattern is interesting because it is highly used and has a big effect on the performance of the system.

In MDS, the performance topics we are interested in include the following:

- How does the performance of a GRIS or a GIIS scale with the number of the concurrent users? Can we suggest an upper bound to be used when a system is configured?
- How does a GRIS or GIIS's throughput, average response time or the overhead of its host behave when there are a large number of concurrent users or a very high request rate? Does there exist a threshold for either the throughput or the average response time for a GRIS or GIIS?
- How much does the caching mechanism used in the GIIS affect its performance?
- How many information providers can a GRIS server accept without dramatic performance degradation?
- What is the maximum number of lower-level GRIS servers that a GIIS server can support without dramatic performance degradation? How does this scale?

In R-GMA, we are interested in the following performance topics:

- How many Producers or Consumers can a Registry hold? Does there exist an upper bound?
- How does the performance of the Registry scale with the Producers or Consumers sending registration requests concurrently?
- How does the performance of the ProducerServlet scale with the number of the concurrent Producers sending registration requests through it to a same Registry server?

- How does the performance of the ConsumerServlet server scale with the number of the concurrent Consumers sending requests through it to locate Producers and consume information data?

4. Experimental Setup

In this section, we describe the details of the experimental testbeds where we explored the performance topics we outlined in Section 3. We include a description of the hardware and software platforms, the deployment of the MDS and R-GMA, the client load generation, and the performance metrics we used for the experiments.

4.1 Hardware and Software Platforms

An effective study of the performance of MDS and R-GMA must simulate the real world of the Grid. The server side of the system needs to be run in a dedicated environment that is different from where the clients are run, in order to prevent the overhead at the client side from being the limiting factor of the system server's performance.

To fulfill these requirements, we set up a testbed at Argonne (ANL) and deployed both MDS and R-GMA on it. The servers of MDS and R-GMA were run in dedicated mode on the testbed. The clients (or users) we called, simulated by the processes generated by client programs, were run in a cluster of nodes at the University of Chicago (UC). A 100 Mbps network connects the ANL testbed and the UC nodes.

4.1.1 Testbed at ANL

We built the testbed for the servers of both MDS and R-GMA using the Lucky nodes at ANL. These nodes are a cluster of seven dual-processor Linux boxes with hostnames $lucky\{0,1,3,\dots,7\}.mcs.anl.gov$ and running kernel 2.4.10 (we will use $lucky_n$ to denote the node $lucky_n.mcs.anl.gov$ in the rest of the paper). The $lucky_2$ was out of service during our experiments. Each Lucky node is equipped with two 1133 MHz Intel PIII CPUs (with a 512 KB cache per CPU) and 512 MB main memory. The interconnect among the Lucky nodes is 100 Mbps Ethernet.

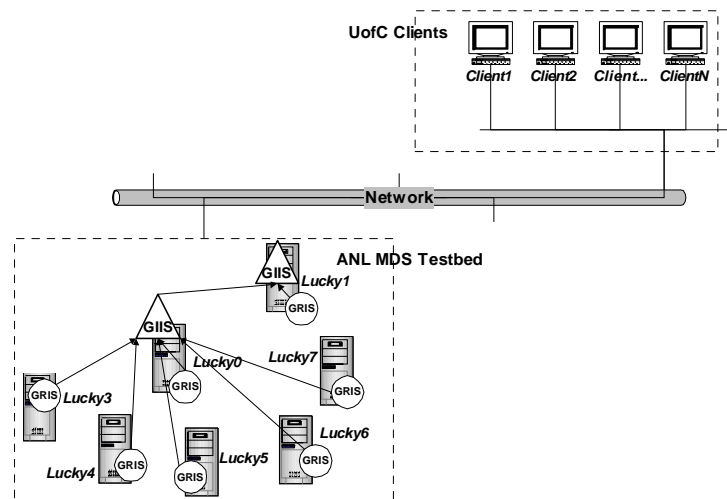


Figure 4: MDS Test-bed Architecture

We adopted the following strategy to set up and configure the MDS server testbed. We employed all seven Lucky nodes. On each node we installed and ran MDS 2.1, part of the Globus Toolkit 2.0. The testbed was configured with each machine running a GRIS to collect the information generated by the information providers for that machine. Two machines, *lucky0* and *lucky1*, were configured to run a GIIS besides the GRIS and act as the directory index servers for the test-bed. The GIIS on *lucky0* was configured to accept the registrations from the GRIS running on *lucky0* and *lucky3* through *lucky7*. The GIIS on *lucky1* was configured to accept the registration from the GIIS on *lucky0*. The hierarchy of the MDS testbed is depicted in the lower part of Figure 4.

The testbed for the R-GMA experiments was constructed in the following manner. We chose the latest version (no version number provided) of the R-GMA obtained from the DataGrid and installed it in the Lucky nodes. *Lucky1* was configured to be the Registry server for the testbed. Thus, an RDBMS (a MySQL database) was also installed and run on *lucky1*. We configured the rest of the Lucky nodes to act as the user servlet and made them communicate with the Registry at *lucky1*, Producers, and Consumers.

4.1.2 Client Environment at University of Chicago

As we have discussed, it is very important that client machines do not operate in a heavy load and in a different environment from the servers while evaluating the server side's performance of a Grid information system. To this end, we selected twenty high-performance Linux boxes at the University of Chicago as the client nodes used for both MDS and R-GMA experiments. Fifteen machines were equipped with a 1208 MHz uniprocessor and 248 MB RAM, while the rest of the machines had a slightly slower CPU (but at least 756 MHz), also with 248 MB RAM. In each experiment, we balanced the client machines' load by distributing the client processes to as many client machines as possible.

4.2 Client Traffic Generation and Constraints

To provide a complete and accurate understanding of the server performance, we designed and implemented customized benchmarking techniques for both MDS and R-GMA. Since we could employ at most twenty client machines, it was impossible for us to actually implement hundreds of clients pinging the GRIS or GIIS in MDS experiments or a server running ConsumerServlet in R-GMA experiments. Instead, we used multiple client processes running on each machine. For example, to simulate the traffic generated by 1,000 users in the real world, we ran 50 traffic-generating processes on each of the twenty client machines.

This approach was limited to at most X "clients" per machine because of the CPU, memory, and network contention on the client side. Another potential pitfall brought by this traffic-generating scheme is that the total request rate generated by the client processes never exceeds the capacity of the server, leaving the effect of request bursts on server performance unevaluated. The reason lies in the limitation caused by some peculiarities of the underlying TCP protocol that is used in both MDS and R-GMA [18].

The client traffic-generating codes for both MDS and R-GMA can be accessed from <http://people.cs.uchicago.edu/~hai/lucky/src/>.

4.3 Performance Metrics

In our experiments designed for both MDS and R-GMA, we used *throughput* and *response time* as our primary metrics. We defined *throughput* as the average number of requests (or queries) processed by a server during a unit of time, typically a second. In MDS, the requests include the MDS queries sent from a client to a GRIS or a GIIS. In R-GMA, the requests include the registration requests sent from a Producer or a Consumer to a Registry (through a ProducerServlet), the queries sent from a Consumer to a ConsumerServlet server, and so on. We defined *response time* as the average amount of time (in second) required for a server to handle a request sent from a client. Response time consists different components in each system. In MDS, response time includes the time to build the connection to the MDS server (GRIS or GIIS), the time required for the MDS server to process the query, and the data transfer time to return the query results to the client. In R-GMA, the response time to handle a query send from a Consumer includes the time to connect to the ConsumerServlet, the time for the ConsumerServlet (on behalf of the Consumer) to connect to the Registry and locate the Producers, the time for the ConsumerServlet to connect to each discovered Producer and retrieve the information, and the data transfer time for the

ConsumerServlet to send the information back the Consumer. Since our performance study focuses on the server-side performance of MDS and R-GMA, *throughput* and *response time* are two very important indexes to show the performance.

In our experiments, we also used metrics to measure the server's overhead performance, including the CPU states, average loads and memory usage. In particular, metric *load1* represents the average number of processes ready to run during the last one minute (average system load during the past 1 minutes); *CPU-User* is the percentage of CPU time used in user mode; and *CPU-System* represents the percentage of CPU time used in system mode. In our study, we combined *CPU-System* and *CPU-User* and used the *CPU-Load* metric to describe the comprehensive CPU performance at the server side.

5. Experimental Results and Analysis

In this section, we first give an overview of the experiments for both MDS and R-GMA. Then we discuss the details of the experiments for both systems. For each experiment, we describe the topic it was designed to address and the configuration of the experiment. Then we present and analyze the performance results, pointing out performance limitations and investigating possible causes.

5.1 Overview of the Experiments

The following experiments for MDS and R-GMA were run on the testbeds described in Section 4.1. In general, each experiment ran for 10 minutes, and the values reported are the average over all the values recorded during that time span. We used Ganglia [19] to collect and log the performance data (wrapped in XML [20]) at five-second intervals. In addition, all requests to the server in both MDS and R-GMA occurred with a certain *wait period* fixed to each experiment. Here we use *wait period* to denote the time interval between two continuous requests sent from a same client in both MDS and R-GMA.

The experiments for MDS consisted of four different experiments. The first experiment studied the effect of large number of the users on the performance of a GRIS. The second experiment investigated how much of performance gain is seen if a GIIS enables data caching. The third one explored the effect of too many information providers on the performance of a GRIS. The fourth experiment studied the effect of too many registered GRIS on the performance of a GIIS. In MDS experiments, clients were configured to send requests or queries using *grid-info-search* command, which is an MDS command to send queries to a GRIS or a GIIS and display the results on stdout.

The R-GMA experiments comprised four parts, too. The first experiment explored the effect of large number of concurrent Producers sending registration requests on the performance of a Registry. The second investigated how well the Registry scales with the number of concurrent Consumers sending registration requests. The third explored the effect of too many Producers on the performance of the host of the ProducerServlet. The fourth investigated the effect of too many Consumers on the performance of the ConsumerServlet and its host. The Producer we used in R-GMA experiments was the CircularBufferProducer [4], which writes information to a circular buffer where it can be picked up by a Consumer. The mode of transfer of information from the Producer to the Consumer in these experiments was the pull model, in which the user asks the Consumer to execute its query and the Consumer asks the Producer to process. We used the term *ProducerServlet server* to denote the machine running ProducerServlet and *ConsumerServlet server* to represent the machine running ConsumerServlet. *Registry server* was used to denote the server providing registry service. In our experiments, the registry server did not run any user servlets.

5.2 MDS Performance

We present here the MDS performance results for each experiment.

5.2.1 Effect of Large Numbers of Users

The purpose of our first experiment for MDS was to study the scalability of a GRIS with respect to the number of users concurrently accessing it. We ran a GRIS at *lucky7* and configured each client process that simulated an independent user sending queries to the GRIS for 10 minutes through UC's client nodes. The wait period of the queries was set to be 0.5 second and 2 seconds, respectively. *Lucky7* had only 8 core GRIS information providers running with default configuration; for example, the memory information was collected by memory information provider every minute (cachetime=60 sec).

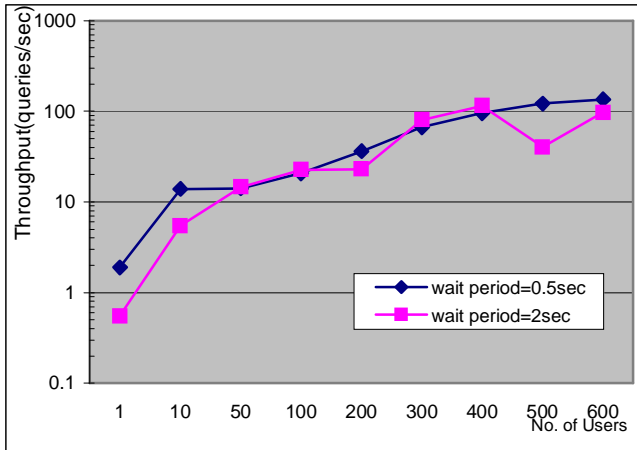


Figure 5: MDS Experiment1: GRIS Throughput vs. No. of Concurrent Users

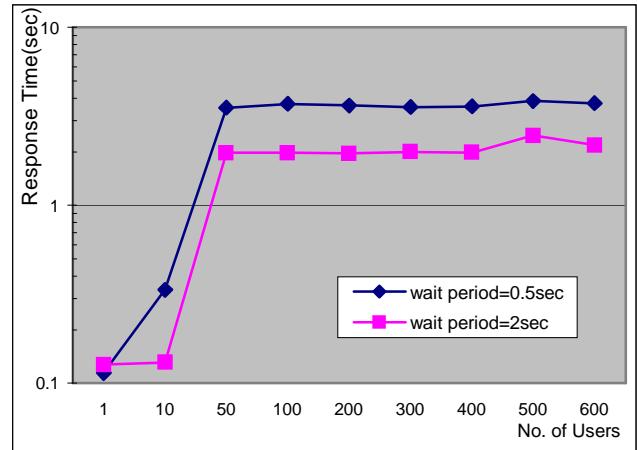


Figure 6: MDS Experiment1: GRIS Response Time vs. No. of Concurrent Users

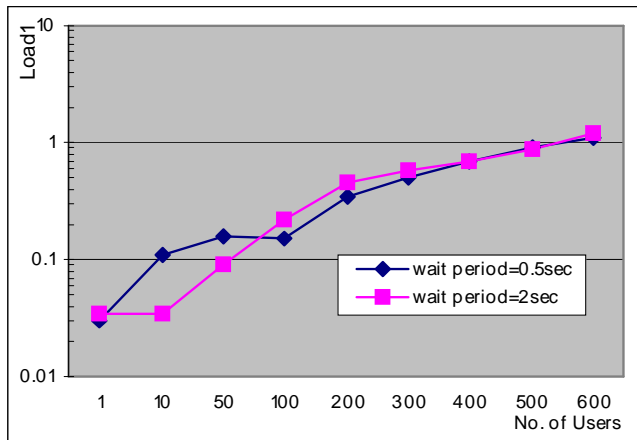


Figure 7: MDS Experiment 1: GRIS Server Load1 vs. No. of Concurrent Users

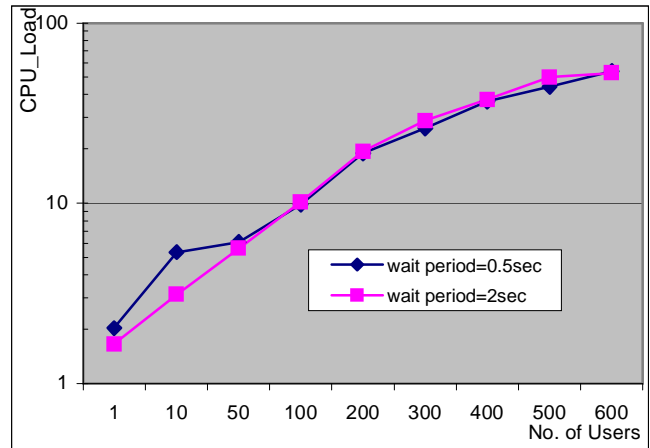


Figure 8: MDS Experiment1: GRIS Server CPU_Load vs. No. of Concurrent Users

The performance results are shown in Figure 5 - Figure 8 (y axis uses logarithmic scale). Figure 5 and Figure 6 plot the GRIS's throughput and response time with respect to the number of the users. The throughput of the GRIS is positively affected by the number of concurrent users. It goes up quickly to approximate 70 queries per second when the number of the users increases from 1 to 300; then the increasing rate of the throughput slows down greatly, and finally it reaches a saturation point at around 110 queries per second. This indicates that the GRIS meets its capacity when the concurrent users are over 400. After that, the throughput remains relatively stable, and adding more users has little effect on it. An investigation of the underlying OpenLDAP server shows that OpenLDAP uses multithreading scheme to handle the client requests. This explains why GRIS can achieve a high throughput when the number of users increases. However, there exists an upper bound for the number of threads that OpenLDAP can create, and it is the reason why the throughput of the GRIS cannot exceed an upper bound (110 queries/sec in our experiment) no matter how many concurrent users it serves. Figure 6 shows that the response time has a very small value (less than 0.2 second) when the users number less than 10, but increases quickly to approximate 4 seconds when the user number is 50 and remains stable after that.

Figure 7 and Figure 8 illustrate the load1 and CPU_Load of the host node of the GRIS - *lucky7*. We observed both of them have a positive relationship with the number of users. At the maximum number of 600 users in the plot, the total CPU load is as high as approximate 60%.

The above observation leads us to conclude that a GRIS presents a very good scalability with regard to the number of concurrent users. It can serve 400 concurrent users with a high throughput (110 queries per second) while keeping response time around 4 seconds that is acceptable to the users. A remaining concern is that the server running the GRIS may experience a high load with the increasing of the users.

5.2.2 Performance Gains of Data Caching

The MDS enables caching of data at both the GRIS (caching the lower-level data from information providers) and GIIS (caching the data from the lower level registrants) to increase the performance of retrieving information. Our second experiment evaluated how much performance gain is seen at registrar side in a quantitative manner. We carried out two tests. One measured the performance of the registrar that always had the requested data in cache; that is, the data in the registrar's cache never expires, and whenever queries are received by the registrar, the cached data is returned. The other to measured the performance of the registrar that did not cache any data; that is, whenever queries are received by the registrar, the registrar requests the data from the next lower level of the hierarchy.

The experiments were set up as follows. We used a GIIS server on *lucky0* as the registrar, which was configured to have five lower-level, GRIS registrants (the hierarchy was described in Figure 4). We configured the *cachettl* parameter, which is used in MDS to represent the recommendation to the registrar about how long to maintain in cache the GRIS information provided by the registran, to a very large value to simulate data always in cache and to zero to simulate no data in cache. Each of the client processes was configured to send queries to the GIIS on *lucky0* to ask for the memory or CPU information in registrant *lucky6* for 10 minutes with the wait period as 1 second.

With data caching enabled and disabled, the GIIS in our study presented very different behaviors. The difference can be observed apparently in the following plots. Figure 9 shows the difference about GIIS throughput, and Figure 10 illustrates the response time. When the GIIS always has data in cache, its throughput increases near linearly with the number of the users (up to 104 queries/second at the user number 120) while the response time remains stable around a very small value, 0.15 second. If the GIIS has no data in cache, the throughput remains very small (not exceeding 3 queries/second) and has no clear relationship with the number of the users. With an increase in the number of users, however, the response time keeps growing (up to 40 seconds with 120 users) and shows an approximately linear relationship with the users. The performance difference about the server running GIIS is also shown in Figure 11 and Figure 12. Figure 11 illustrates the load1 overhead of the server in both cases, while Figure 12 shows the absolute difference of the CPU_Load of the server in both cases. From Figure 12, we see that there is a big gap (nearly 45%) in CPU_Load of the server depending on whether the GIIS has the data in cache or not. The CPU_Load difference remains constant when the number of users increases. Hence, without data in cache, the server running the GIIS experiences a much higher load to handle the users' queries.

These observations indicate that the caching mechanism plays an important role in improving the performance in MDS at the GIIS level. With data caching we can gain a much better performance. A new concern arises, however, about what kind of data should be cached and how long these data should be kept in the cache. Different applications have different requirements about the efficiency of retrieving the data and the data itself. Hence, the caching mechanism must be customized to different applications.

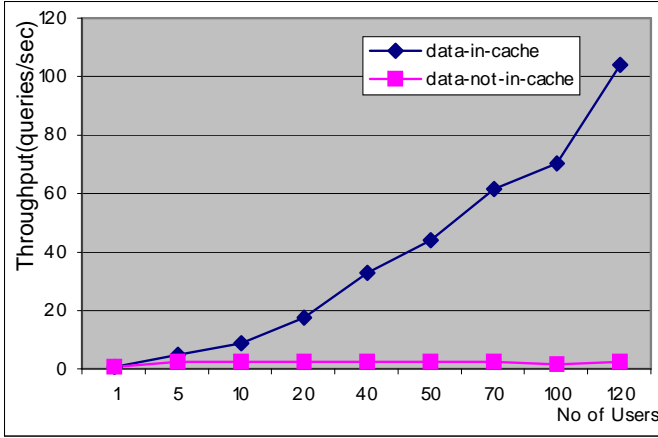


Figure 9: MDS Experiment2: GIIS Throughput vs. No. of Concurrent Users

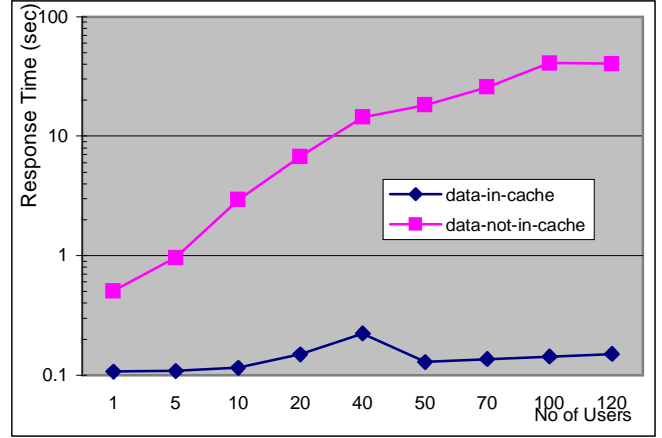


Figure 10: MDS Experiment2: GIIS Response Time vs. No. of Concurrent Users

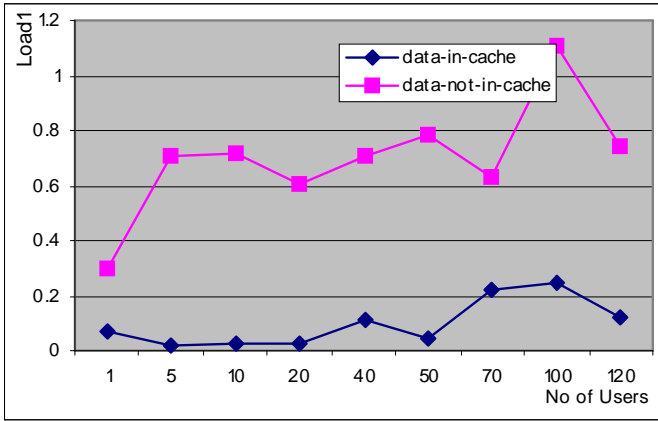


Figure 11: MDS Experiment2: GIIS Server Load1 vs. No. of Concurrent Users

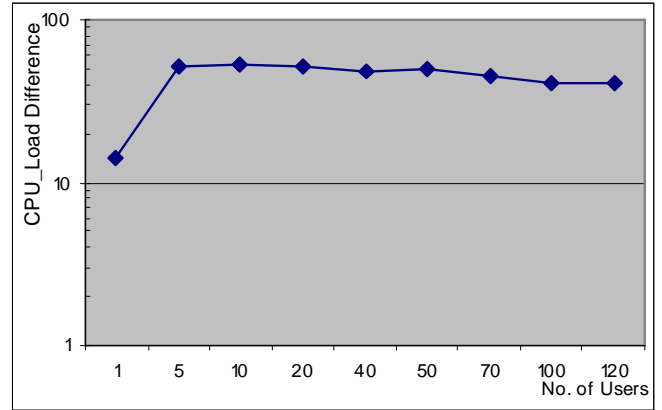


Figure 12: MDS Experiment2: GIIS Server CPU_Load vs. No. of Concurrent Users

5.2.3 Effect of Too Many Information Providers

Currently MDS version 2 provides collections of default information providers for different platforms. The core providers include

- *grid-info-cpu* reports CPU and load information with the default *cachetime* 60 seconds
- *grid-info-fs* reports filesystem information with the default *cachetime* 900 seconds
- *grid-info-mem* reports random access memory (RAM) and virtual memory (VM) information with the default *cachetime* 60 seconds
- *grid-info-net* reports Net Interface Card (NIC) and the network information with the default *cachetime* 900 seconds
- *grid-info-os* reports operating system information with the default *cachetime* 12 hours
- *grid-info-platform* reports architecture information with the default *cachetime* 60 seconds
- *grid-info-merged* merges all host information

Although the default information providers meet the needs for most Grid activities, we can expect that, as more new services and applications are added into the Grid system, we will need to design and add new information providers into the service of MDS. Thus, it is important to study how increasing numbers of information providers affects the performance of the GRIS that these information providers register to.

In this experiment, we modified the memory information provider provided in the core MDS providers and added multiple copies of the new version into the default information providers to simulate an increasing number of information providers. The caching time of the new memory information has a time to live of 1 minute. Four tests were carried out with the information provider number 8 (the default value), 20 more, 60 more, and 90 more, respectively. We configured each of the client processes to send queries to the GRIS at *lucky7* where the new information providers were installed and in service. Each client process ran 10 minutes, with a wait period be 0.5 second. Every query sent from each client process was configured to request for the information from all the registered information providers in the GRIS at *lucky7*.

Our experiment results indicate that the GRIS's performance degrades significantly with respect to the increasing number of the information providers. As shown in Figure 13 and Figure 14, the more information providers, the bigger the loss of performance of the GRIS. The degradation rate of the response time is much faster than the throughput when the user number gets bigger. The load1 and CPU_Load results depicted in Figure 15 and Figure 16 provide more evidence that as the number of the information providers grows, the GRIS server load increases as well. Because the observed performance constraint, we recommend that each GRIS be responsible for no more than 30-50 information providers if there is a large number of concurrent users. This recommendation is also based on the fact that all the information providers are involved in most of user's queries. If each query contacts only a small fraction of the information providers, the maximum number a GRIS holds can be more than 50.

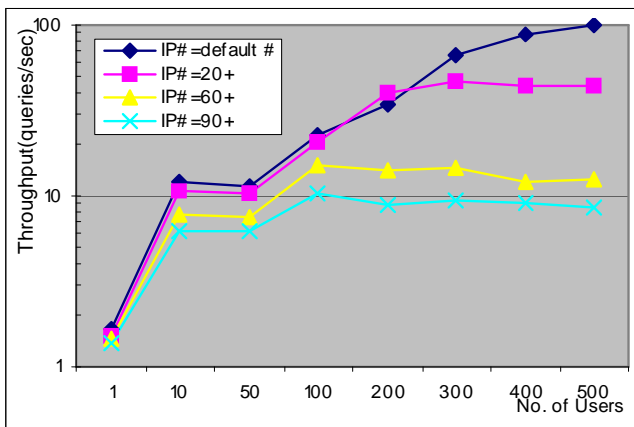


Figure 13: MDS Experiment 3: GRIS Throughput vs. No. of Concurrent Users

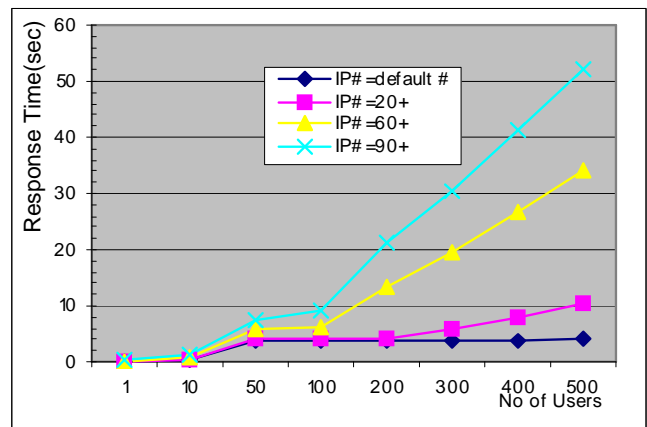


Figure 14: MDS Experiment 3: GRIS Response Time vs. No. of Concurrent Users

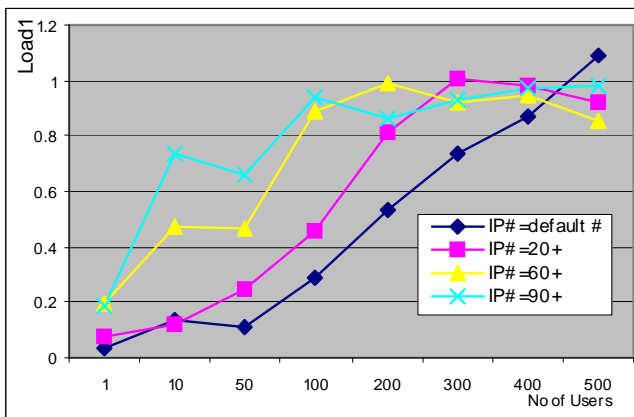


Figure 15: MDS Experiment 3: GRIS Server Load1 vs. No. of Concurrent Users

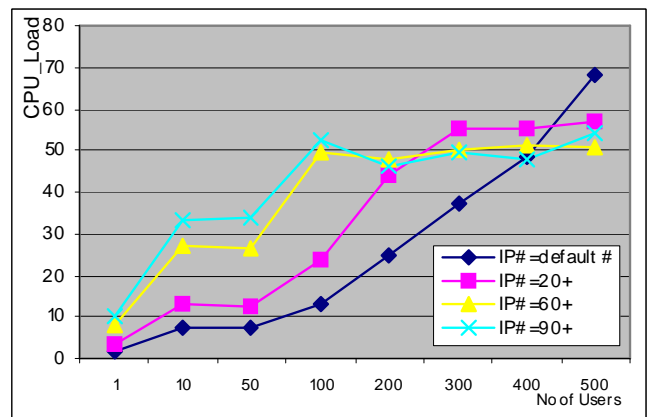


Figure 16: MDS Experiment 3: GRIS Server CPU_Load vs. No. of Concurrent Users

5.2.4 Effect of Too Many GRIS

In our third experiment, we observed that having too many information providers greatly affects the performance of a GRIS. Thus a natural question is, what is effect of too many lower-level GRIS as the registrants on the performance of an upper-level GIIS as the registrar? The answer to this question is also very important because it helps to understand how the registrar scales with the registrants in MDS.

Our first task in this experiment was to simulate a large number of lower-level GRIS. Since the Lucky testbed contains only 7 machines, we simulated additional GRIS by multiple (typically, 10) GRIS instances at each node. Each node (except *lucky0*) registered to the GIIS running at *lucky0* and sent its registration information in 10-second intervals. In this experiment, we configured the GIIS to let it have 5, 50, and 100 registered GRIS respectively. The client processes running at machines at UC sent queries to GIIS at *lucky0* to query the information from all the lower-level GRIS for 10 minutes with a 0.5 second wait period. Data caching was enabled at both the GIIS and each GRIS. The GIIS was configured to cache each GRIS's data for 30 seconds.

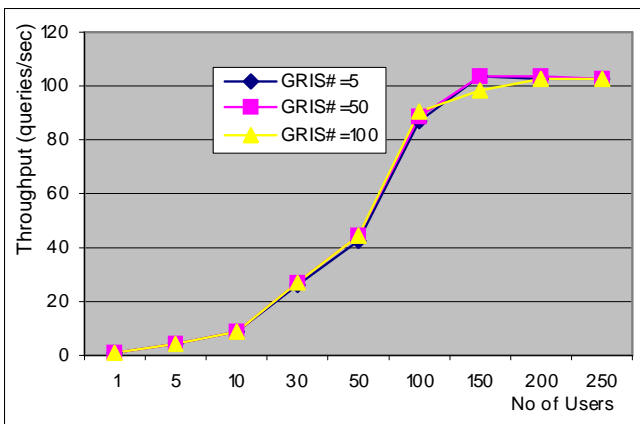


Figure 17: MDS Experiment 4: GIIS Throughput vs. No. of Concurrent Users

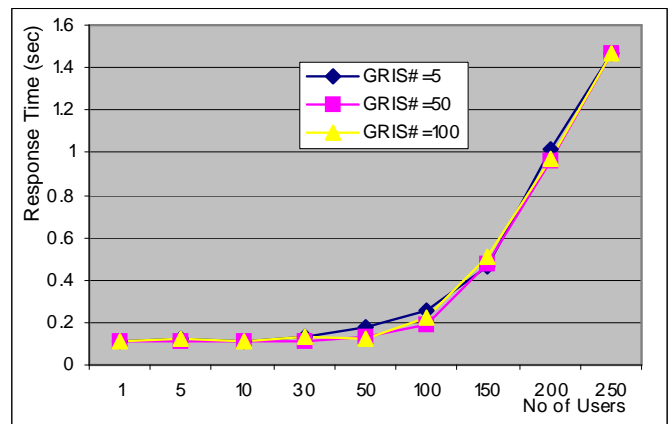


Figure 18: MDS Experiment 4: GIIS Response Time vs. No. of Concurrent Users

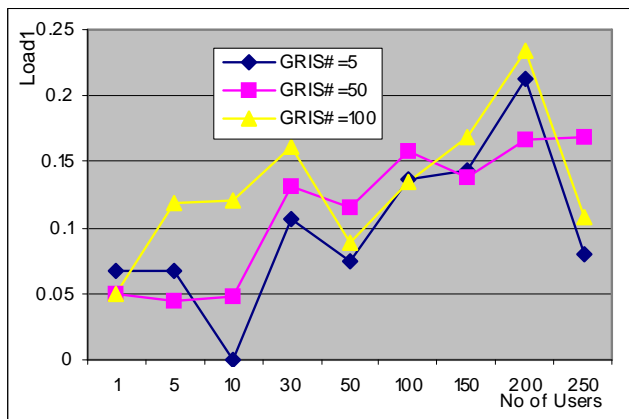


Figure 19: MDS Experiment 4: GIIS Server Load1 vs. No. of Concurrent Users

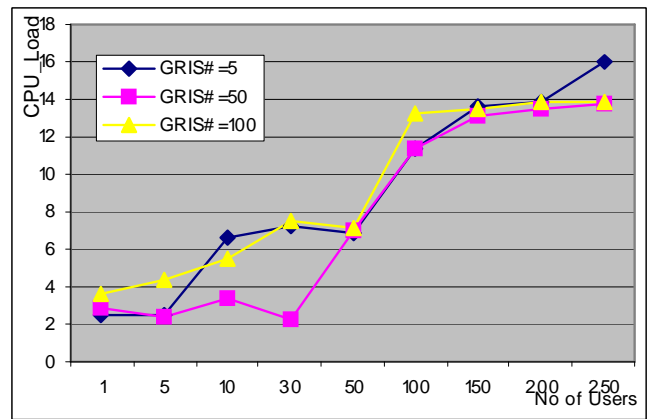


Figure 20: MDS Experiment 4: GIIS Server CPU_Load vs. No. of Concurrent Users

Our performance results show the GIIS has very good scalability with respect to the number of the lower-level GRIS. When the GRIS number is increased from 5 to 100 through 50, the GIIS's throughput (Figure 17), response time (Figure 18), the load1 average (Figure 19), and CPU_Load (Figure 20) present the similar patterns; and their values at a certain number of

users vary only slightly. What we found in this experiment indicates that in the real world, the directory server in a virtual organization can be responsible for over 100 registrants while still keeping very good performance.

In this experiment, the number of GRIS we simulated was no larger than 100. We did not go beyond this number because a big number means a large number of GRIS instances running at each node and the overhead brought by the resources contention among these instances can become a bottleneck for the overall performance. More machines have to be employed to simulate a very big GRIS number.

5.3 R-GMA Performance

We next conducted four experiments to evaluate R-GMA performance. Our focus again was on scalability.

5.3.1 Effect of Concurrent Registering Producers on Registry

Our first experiment of R-GMA studied how the performance of the Registry scales with the concurrent Producers sending the registration requests through ProducerServlet. In this experiment, we ran the Registry on *lucky1* and made the client processes, each of which simulated a CircularBufferProducer. Each Producer sent registration requests to the Registry with a 0.5 wait period between requests. We measured this behavior for 10 minutes. We ran a ProducerServlet on each of the rest of the Lucky nodes.

Since the goal of this experiment was to test the behavior of the Registry, we had to make sure ProducerServlet not be the bottleneck of Registry’s performance. We adjusted the number of Producers communicating with each ProducerServlet to balance the load.

The experiment results are shown in Figures 21 and 22. Figure 21 shows that the throughput of the Registry is likely to be positively affected by the number of Producers sending concurrent requests. However, the increasing rate of the throughput is small. Figure 22 illustrates that the Registry’s response time remains fairly stable (around 1.5 seconds) even the Producer’s number goes up to 400.

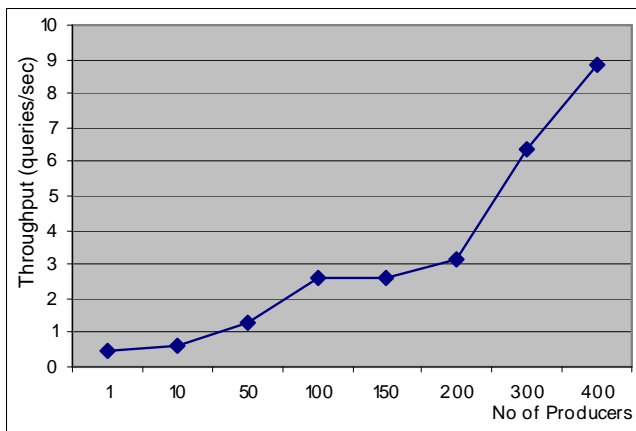


Figure 21: R-GMA Experiment1: Registry Throughput vs. No. of Concurrent Producers

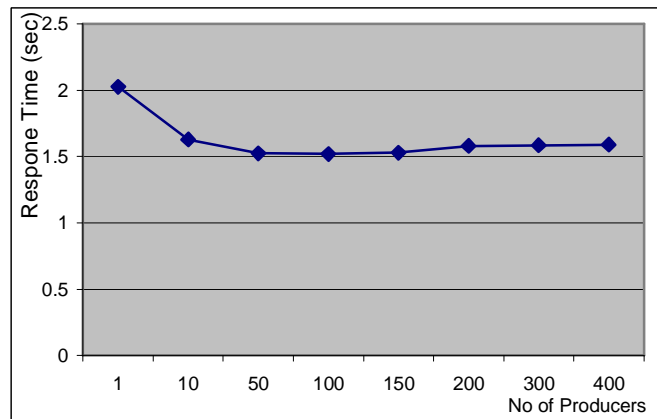


Figure 22: R-GMA Experiment1: Registry Response Time vs. No. of Concurrent Producers

On the other hand, the Registry server’s load (depicted in Figure 23 and Figure 24) shows a different performance pattern from the response time. We observed a large increase in both load1 and CPU load with 200 Producers sending registration requests concurrently. We believe this increase occurs because either the network (Registry also relies on a servlet called RegistryServlet to handle the connections from the ProducerServlet) or the functionality of publishing the Producer’s information to the Registry database or both can no longer handle the traffic from the Producers. Thus, the Registry server is overloaded by an increasing number of unserved registration requests from the Producers.

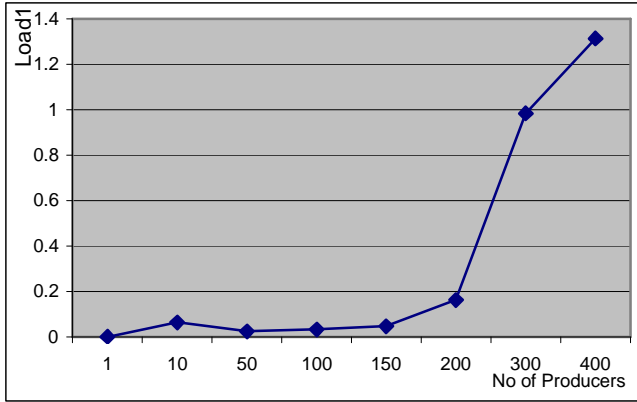


Figure 23: R-GMA Experiment1: Registry Server Load vs. No. of Concurrent Producers

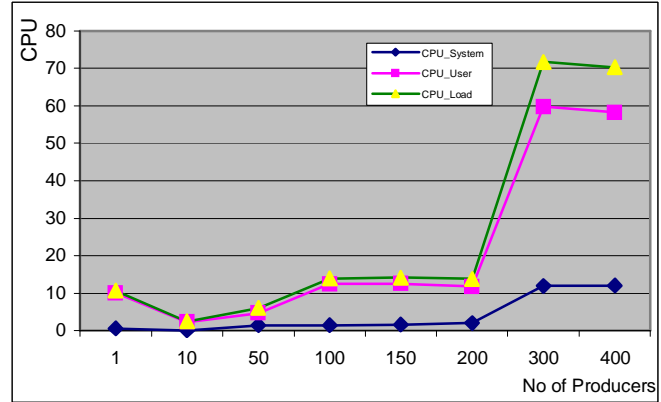


Figure 24: R-GMA Experiment1: Registry Server CPU Load vs. No. of Concurrent Producers

5.3.2 Effect of Concurrent Registering Consumers on Registry

Our second experiment studied the effect of the concurrent Consumers sending registration requests on the performance of the targeted Registry. The purpose of this experiment was to quantitatively determine how many concurrent Consumers a Registry can accept without great performance degradation. This is analogous to the MDS experiment, which showed the GRIS or GIIS's performance, scaled with the number of the users. We will discuss their similarity in Section 6.

The Registry was running at *lucky1*. Each client process (running at UC nodes) simulated a Consumer and sent registration requests to the Registry for 10 minutes with a 0.5 second wait period. We adopted the load-balancing scheme described in Experiment 1 to direct each request from a Consumer to a ConsumerServlet server with the lowest load. Usually a Consumer sends the registration request to Registry and discovers the information about the corresponding Producers from the Registry; then it contacts the discovered Producers for data. In this experiment, the Consumers were customized to send only the registration requests and perform the discovery of the Producers. They did not further contact the discovered Producers for the data.

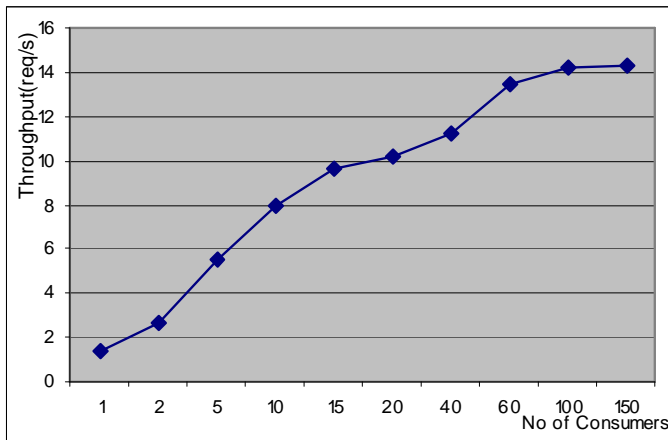


Figure 25: R-GMA Experiment2: Registry Throughput vs. Concurrent Consumers

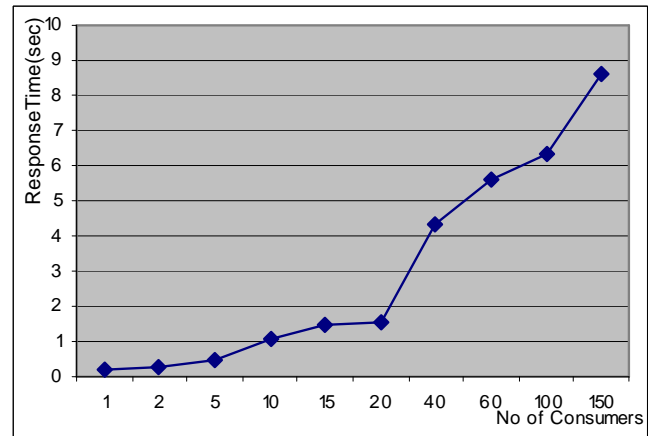


Figure 26: R-GMA Experiment2: Registry Response Time vs. Concurrent Consumers

As we observed from Figure 25, the Registry throughput shows a near linear relationship with the number of concurrent Consumers although the increasing rate of the throughput is relatively small. The Registry response time results (shown in

Figure 26) presents that the Registry’s response time keeps small while Consumers are under 20, after that, it increases faster and reaches 9 seconds at 150 Consumers. However, when we looked through the Registry server’s load performance in Figure 27 and Figure 28, we observed the server experiences a high load even the number of Consumers is small. CPU_Load grows much faster and reaches 50% at the 10 concurrent Consumers.

The performance results indicate that the Registry shows a good scalability with the increasing of the concurrent Consumers but the server hosting the Registry can experiences a high load at a small number of Consumers. The performance constraints may come from the following sources: the network (RegistryServlet in the Registry server to handle the connections from the ConsumerServlets), the functionality to publish the Consumer’s existence information to the Registry’s database, and the functionality to discover the corresponding Producers in the database.

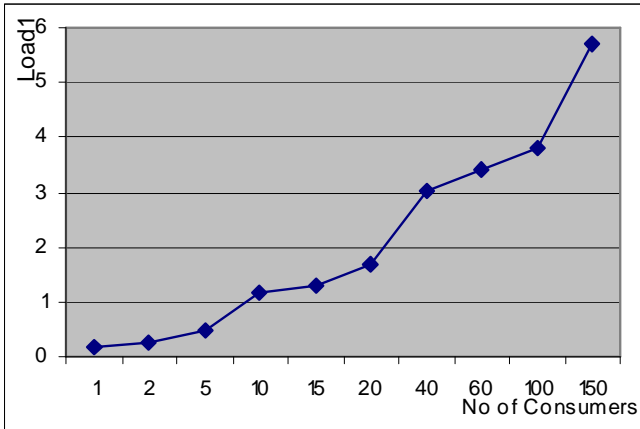


Figure 27: R-GMA Experiment2: Registry Server vs. Load1 Concurrent Consumers

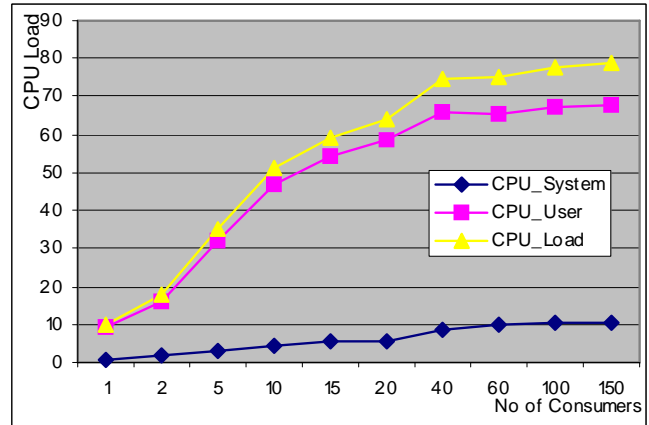


Figure 28: R-GMA Experiment2: Registry Server CPU_Load vs. Concurrent Consumers

5.3.3 Effect of Producers on ProducerServlet

In this experiment, we studied the scalability of the ProducerServlet with respect to the number of concurrent Producers (still used CircularBufferProducer) sending connection requests to it. The ProducerServlet also handles the transfer and the storage of the information data produced by the Producers. This experiment is analogous to the experiment in MDS that studied the effect of too many information providers on the performance of a GRIS, if we treat the ProducerServlet similar as the GRIS. It is necessary to know how many Producers a ProducerServlet can hold without big performance loss so that we can adjust the ProducerServlet’s incoming connections from the Producers to a reasonable value to keep the server at a low load.

We used the *lucky3* as the ProducerServlet server. The Registry was run on *lucky1*. Producers simulated by the client processes were run at the UC client nodes; each was configured to publish its existence and the information data to *lucky1* Registry through the *lucky3* ProducerServlet for 10 minutes. In this time span, we also ran a small number of Consumers (8) on *lucky4* to consume the information generated by all the Producers through the ConsumerServlet running on the same node; otherwise, the Producer will be destroyed if no Consumer consumes the data it produces before its expiration time. In this experiment, we explored the performance of the ProducerServlet with 10, 50, and 100 registered Producers respectively.

The performance results are shown in Figure 29 - Figure 32. With the increasing number of registered Producers, the performance of the ProducerServlet degrades at a certain number of accessing Consumers. Both the throughput and response time can be unacceptable when the ProducerServlet is responsible for a large number of Producers while it is accessed by a large number of Consumers. At the same time, the server hosting the ProducerServlet also experiences a higher load.

This observation leads us to conclude that the ProducerServlet can achieve a good scalability in different configurations. If it is responsible for a small number of Producers, it can present a good performance even the number of accessing Consumers is big. If the number of registered Producers is very big (around 100), it can only maintain a good performance only on condition that the number of concurrent Consumers is small.

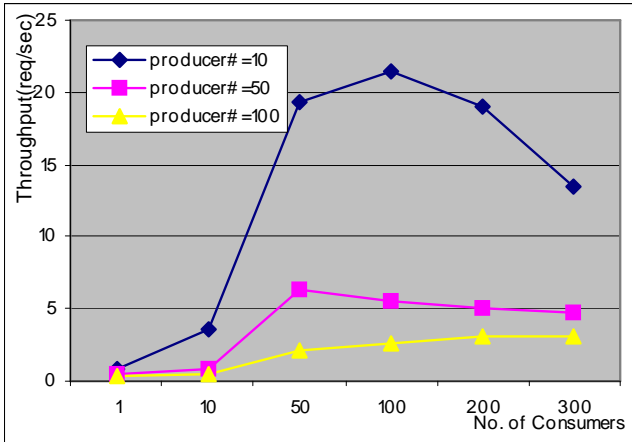


Figure 29: R-GMA Experiment3: ProducerServlet Throughput vs. Producers

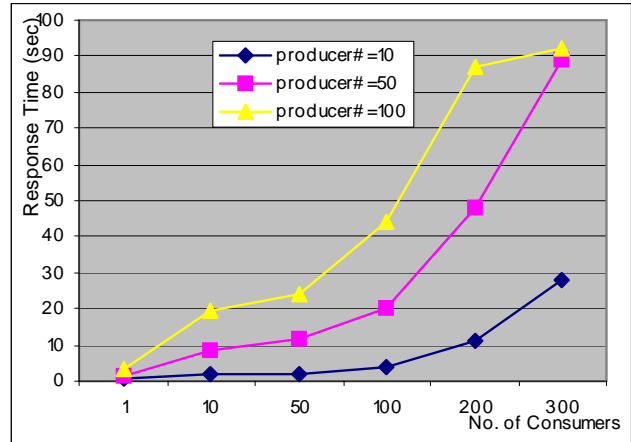


Figure 30: R-GMA Experiment3: ProducerServlet Response Time vs. Producers

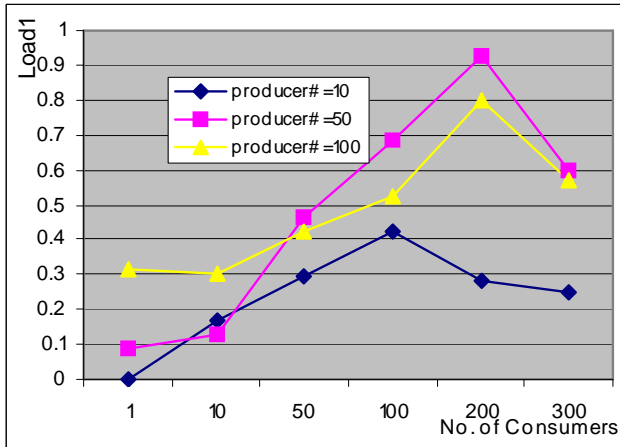


Figure 31: R-GMA Experiment3: ProducerServlet Server Load1 vs. Producers

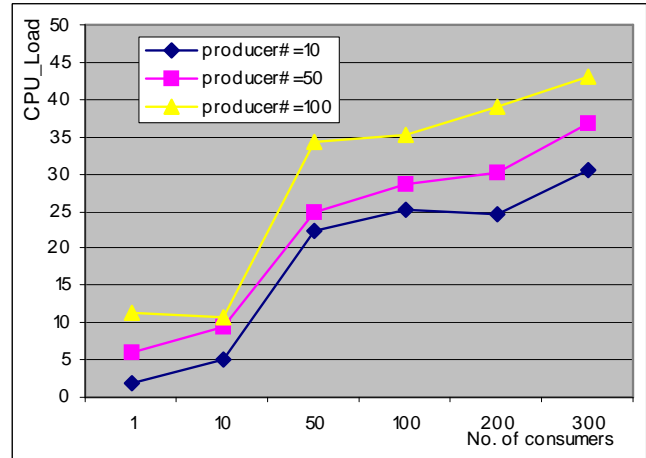


Figure 32: R-GMA Experiment3: ProducerServlet Server CPU Load vs. Producers

5.3.4 Effect of Consumers on ConsumerServlet

In our fourth R-GMA experiment, we investigated the performance of the ConsumerServlet scaling with the number of the Consumers. In the real world, we want to know whether there is any upper bound of the Consumers a ConsumerServlet is responsible for. The answer to this question will help in tuning the ConsumerServlet server so that it is not overloaded by the Consumers.

In the experiment, *lucky3* was used as the ConsumerServlet server. The Registry resided in *lucky1*. Consumers were simulated by the client processes running at UC. Each Consumer continuously contacted the Registry to find out the Producers producing the interested information through the *lucky3* ConsumerServlet for 10 minutes. After the Producers were found, the ConsumerServlet built connections to them on behalf of the Consumers, retrieved the wanted information, and returned the information data to the Consumers. The configuration of the Consumers in this experiment was different from that in the second experiment. Each Consumer not only registered itself to the Registry and discovered the corresponding Producers, but also waited until receiving the information data sent from the Producers. To provide the information interested to all the Consumers, we ran three Producers at *lucky4*.

The performance results are shown in Figures 32 and 33. Figure 35 and Figure 36 show both load1 and CPU_Load of the ConsumerServlet server grows with the increasing number of Consumers. The CPU_Load is approximate 40% at 20 concurrent Consumers and sharply increases after that. It reaches 72% at 50 Consumers. A similar pattern was observed in the plot of the response time (Figure 34). The throughput of the ConsumerServlet reaches its peak value (12 requests/second) at 50 concurrent Consumers, then begins to decrease with the number of the Consumers.

This observation leads us to conclude that 50 is a critical value of concurrent Consumers to access a ConsumerServlet for Producer’s data if there are only a few Producers (3 in our experiment) producing data. We still need to investigate the ConsumerServlet’s performance under the situation that there are a large number of Producers to produce data and the data is consumed by all the Consumers communicating with the ConsumerServlet.

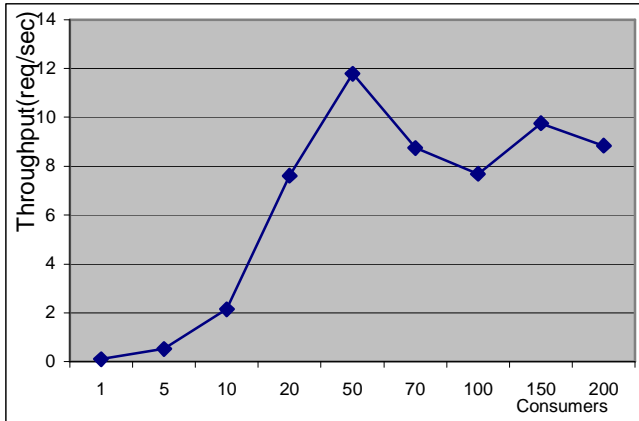


Figure 33: R-GMA Experiment4: ConsumerServlet Throughput vs. Consumers

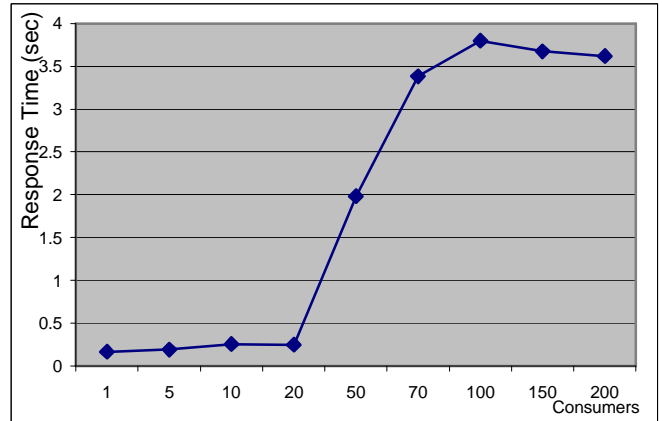


Figure 34: R-GMA Experiment4: ConsumerServlet Response Time vs. Consumers

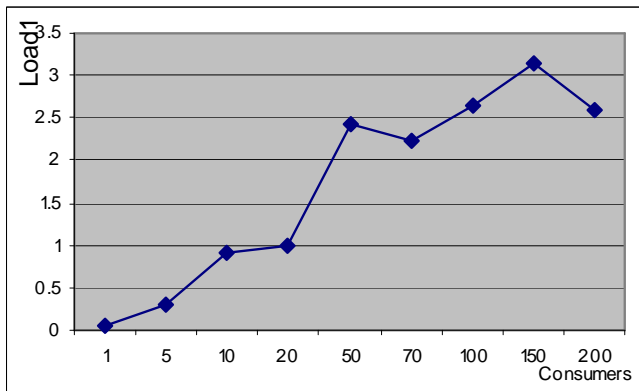


Figure 35: R-GMA Experiment4: ConsumerServlet Server Load1 vs. Consumers

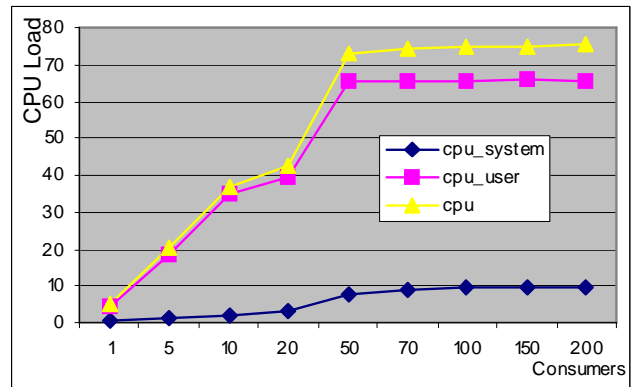


Figure 36: R-GMA Experiment4: ConsumerServlet Server CPU Load vs. Consumers

6. Comparison between MDS and R-GMA

Besides the independent performance study of both MDS and R-GMA described above, we also carried out a joint study by comparing them with each other and using equal match. The comparison is based on their architectures, technologies, and the performance experiments we did for each of them. Through the comparison, we believe we can understand their functionalities and performance advantages better.

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. 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. Both MDS and R-GMA treat the information data in a uniform manner (LDAP in MDS and Relational Data Model in R-GMA) and use a global schema to describe the information. Both systems depend on sensors (information providers in MDS and Producers in R-GMA) to collect information and they handle static and dynamic information in a Grid very well. They both employ soft state registration scheme to handle the lifecycle of their components (lower-level GRIS or GIIS in MDS and Producers or Consumers in R-GMA) and this helps to prevent the information system itself bringing too much overhead to the Grid.

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. In MDS, a lower-level service component can publish information to multiple aggregate service components, which means the whole system is decentralized by natural. 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 inter-architecture should be built in these Registries to guarantee consistency. 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.

MDS employs well-defined protocols among service components and uses LDAP as a common interface and data model, which enables the system to be easily used. R-GMA uses a relation data model to manage the information data and employs protocols based on Java Servlet technologies to construct the communication channels among the service components.

In version 2.0, MDS uses only the pull model for the transfer of the information data; that is, clients build connections to MDS servers and send queries for the data. One concern about the pull model is that when too many clients request information from the same server, the server may be overloaded by too many connections, and networking can be the limiting factor of the server's performance. However, MDS's hierarchical architecture diminishes this effect by replicating the lower-level information among the higher-level. R-GMA supports both the pull and the push models for transferring the information data. Consumers as well as Producers are registered in the Registry. A Producer can initiate the connection to a Consumer who is interested in its data. The push model also enables the Consumer to be informed whenever its interested data is available.

MDS uses LDAP query language. In R-GMA, the query language is based on SQL. Comparing to LDAP query language, the SQL is simpler and more natural to do the join operation on two or more different information object.

We believe the difference between MDS and R-GMA may bring different impact on their performance even when they are tested in similar experiments. It is interesting to compare them with each other. Our study shows the components of MDS and the relationships among them have similar counterparts in R-GMA. As we discuss earlier, the Producer in R-GMA plays a similar role as the information provider in MDS since both of them handle the collection of the information. The Registry in R-GMA acts a similar role as the GIIS in MDS because both of them store and provide information about the resources to the users. However, GIIS does more than Registry. It also provides the aggregated information collected from the lower-level GRIS. Moreover, the ProducerServlet in R-GMA is the counterpart of GRIS in MDS since eventually the ProducerServlet acts on behalf of the registering Producers to transfer the information data to the interested Consumer. A stricter mapping is to let ProducerServlet run on the same machine where Producers run, since in MDS information providers run on the same machine as GRIS.

Using this mapping scheme, we found that Experiment 2 (how much of performance gain is seen if a GIIS enables data caching) of MDS, in which the data is not in cache, is comparable to the experiment of R-GMA to investigate how well the Registry scales with the number of concurrent Consumers sending registration requests. In the former case, the throughput is likely to be positively affected by the number of users, and the throughput is approximately 60 queries/seconds at 60 concurrent numbers. In R-GMA, the throughput increases with the number of Consumers when the number is less than 15; after that, the throughput remains stable. Compared with the value observed in MDS, the R-GMA Registry throughput is much smaller (only approximately 10).

A comparison of Experiment 3 of MDS and R-GMA is also interesting. The MDS performance results show that when the concurrent user number is very big (say, around 500), the GRIS should be responsible for no more than 50 information providers. The R-GMA results illustrate that if the request rate from the Consumers is not very high, the ProducerServlet can be responsible for 300 registering Producers.

Our comparison study of MDS and R-GMA indicates that both systems contain comparable components and fulfill the similar responsibilities, but show different performance results.

7. Conclusion and Future Work

In this paper, we investigated two Grid information systems, MDS and R-GMA. A collection of experiments were performed on each system to study their performance behaviors and constraints. The performance results and analysis give us a better understanding of the performance topics we listed at the beginning of this paper.

Our work shows that both MDS and R-GMA present good scalability with respect to the number of entities producing and consuming the information, although they are built on different architecture and technologies. However, R-GMA presents a centralized architecture that might bring constraints to its performance.

In our future work, we plan to do more experiments to study other characteristics of the two information systems. For example, both MDS and R-GMA testbeds were built in a LAN environment; the experiments should be repeated to study performance in a WAN environment. More user accessing patterns should be included in future performance studies, such as accessing during peak time for network with high workload vs. off-peak time for network with high load or accessing MDS through Simple Authentication and Security Layer (SASL) binding vs. anonymous binding.

ACKNOWLEDGMENTS

We are grateful to many colleagues for valuable discussions and suggestions about the topics presented here, in particular the members of the *Scheduling Prediction and Monitoring (SPM)* group (<http://www-unix.mcs.anl.gov/~shopf/group.html>). Also, we thank John Mcgee at ISI of USC and James Magowan at EU Datagrid for their help on a variety of issues about MDS and R-GMA, and we thank Scott Gose and Charles Bacon for their tremendous help in setting up the testbed at Argonne.

REFERENCES

- [1] Foster, I., and Kesselman, C., eds. *The Grid: Blueprint of a New Computing Infrastructure*. Morgan Kaufmann, 1999.
- [2] Czajkowski, K., Fitzgerald, S., Foster, I., and Kesselman, C. "Grid Information Services for Distributed Resource Sharing". In *Proc. 10th IEEE International Sym.p. on High Performance Distributed Computing (HPDC-10)*, IEEE Press, 2001.
- [3] "The Monitoring and Discovery Service (MDS)", <http://www.globus.org/mds/>.
- [4] "DataGrid Information and Monitoring Services Architecture: Design, Requirements and Evaluation Criteria", Technical Report, DataGrid, 2002.
- [5] "The DataGrid Project: WP3 – Information and Monitoring Services", <http://hepunix.rl.ac.uk/edg/wp3/>.
- [6] "Global Grid Forum". <http://www.gridforum.org/>.
- [7] Tierney, B., Aydt, R., Gunter, D., Smith, W., Taylor, V., Wolski, R., and Swany, M. "A Grid Monitoring Architecture". Technical Report GWD-Perf-16-3, GGF, 2002.

- [8] “The Globus Project”, <http://www.globus.org>.
- [9] “The Condor Project”, <http://www.cs.wisc.edu/condor/>.
- [10] Foster, I., Kesselman, C., and Tuecke, K.. “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”. *International J. Supercomputer Applications*, 15(3), 2001.
- [11] Yeong, W., Howes, T., and S. Kille, “Lightweight Directory Access Protocol”, RFC 1487. In *Performance Systems International*, University of Michigan, ISODE Consortium, July 1993.
- [12] “OpenLdap Project”. <http://www.openldap.org/>.
- [13] Johner, H., Brown, L., Hinner, F., Reis, W., and Westman, J., “Understanding LDAP”. IBM Redbook, 1998.
- [14] Fisher, S., “Relational Model for Information and Monitoring”. Technical Report GWD-Perf-7-1, GGF, 2001.
- [15] “Java Servlet”, <http://java.sun.com/products/servlet>.
- [16] Hunter, J., and Crawford, W, *Java Servlet Programming*. O’Reilly, 1998.
- [17] “Apache Tomcat”, <http://jakarta.apache.org/tomcat/>.
- [18] Banga, G., and Druschel, P., “Measuring the Capacity of a Web Server under Realistic Loads”. *World Wide Web* (Special issue on World Wide Web Characterization and Performance Evaluation), 2(1):69-83, May 1999.
- [19] Ganglia, <http://ganglia.sourceforge.net>.
- [20] Deitel, H., Deitel, P., Nieto, T., and Sadhu, P., *XML: How to Program*. Prentice Hall, 2001.
- [21] Wang, X., Schulzrinc, H., Kandlur, D., and Verma, D, “Measurement and Analysis of LDAP Performance”. In *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems 2000*, Santa Clara, California.