

Production Services for Information and Monitoring in the Grid

Rob Byrom⁴, Brian Coghlan⁶, Andy Cooke¹, Roney Cordenonsi³, Linda Cornwall⁴, Martin Craig⁴, Abdeslem Djaoui⁴, Steve Fisher⁴, Alasdair Gray¹, Steve Hicks⁴, Stuart Kenny⁶, Jason Leake⁷, Oliver Lyttleton⁶, James Magowan², Robin Middleton⁴, Werner Nutt¹, David O'Callaghan⁶, Norbert Podhorszki⁵, Paul Taylor², **John Walk**⁴, Antony Wilson⁴.

1: Heriot-Watt, Edinburgh, 2: IBM-UK, 3: Queen Mary, University of London, 4: Rutherford Appleton Laboratory, 5: SZTAKI, Hungary, 6: Trinity College Dublin, 7: Objective Engineering Ltd.

Abstract

R-GMA is a realization of the Grid Monitoring Architecture (GMA) that also exploits the power of the *relational* data model and the SQL query language. The biggest challenge during the development of R-GMA was to ensure that it could be scaled to operate in a large grid reliably. The system is being used in areas as diverse as resource discovery, job logging and bookkeeping, network monitoring and accounting. A secure, reliable and scalable *web services* version of R-GMA is being developed within the follow-on European project EGEE. Work continues within GGF to define information services for OGSA on the basis of experience with R-GMA.

Introduction

R-GMA is the Grid Information and Monitoring System developed in Work Package 3 of the European DataGrid (EDG) project [1]. Although that project has now ended, R-GMA development continues as part of the EGEE (Enabling Grids for E-science in Europe) project. With EGEE, the emphasis has shifted from research, to the implementation of production-quality middleware.

R-GMA has been described in detail before (see, for example [3]). This paper gives a brief overview of R-GMA, followed by some examples of real experience with R-GMA. We then describe the work being done to take R-GMA forward as production-quality software.

Overview of the R-GMA Architecture

Relational Grid Monitoring Architecture

R-GMA is an implementation of the Grid Monitoring Architecture proposed by the Global Grid Forum [4], that models the information infrastructure of a Grid as a set of *consumers* (that request information), *producers* (that provide information) and a central *registry* which mediates the communication between producers and consumers. Producers contact the registry to announce their intention to publish data, and consumers contact the registry to

identify producers which can provide the data they require. The data itself passes directly from the producer to the consumer: it does not pass through the registry.

Conceptually, the use of a registry in GMA is similar to the use of a UDDI registry for web services: they are both central to the architecture, and are used for the purpose of discovery, but the similarity ends there. A UDDI registry stores mostly static information about services. In contrast, a GMA registry holds information about producers and consumers and it must be able to deal with very high update and query rates in the chaotic and dynamic Grid environment where producers and consumers come and go.

R-GMA adds a standard query language (a subset of SQL) to the GMA model, so consumers issue SQL queries and receive tuples (database rows) published by producers, in reply. R-GMA also ensures that all tuples carry a *time-stamp*, so that monitoring systems, which require time-sequenced data, are inherently supported.

Virtual database

R-GMA presents the information resources of a virtual organisation (VO) as a single *virtual database*, containing a set of *virtual tables* (see Figure 1). A schema contains the name and structure of each virtual table, and the registry contains a list, for each table, of producers who

have offered to publish (provide rows for) the table. When a consumer runs an SQL query against a virtual table, the registry service selects the best producers to answer the query in a process called *mediation*. The consumer then contacts each producer directly, to receive the answer tuples. It is in this sense that the database is virtual: there is no central store of data.

It's important to note that R-GMA is not a conventional distributed RDBMS. The power (and novelty) of R-GMA results from the ability of the mediator to be able to find a set of producers able to answer a consumer's query in the dynamic and heterogeneous environment of the Grid, without the consumer having to be aware of the process.

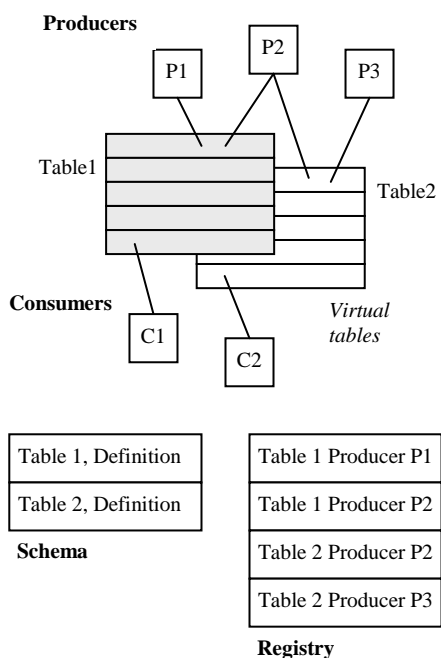


Figure 1

Service Architecture

R-GMA has a service-oriented architecture. Central producer and consumer services act as agents on behalf of user-applications. A user-application interfaces to an R-GMA service via an R-GMA API; there are versions for Java, C, C++ and Python. It requests a service by sending a message (currently an HTTP GET/POST message) to an R-GMA service, and waiting for a response. The two most important operations are INSERT, by means of which a producer publishes a row of data, and

START by means of which a consumer starts execution of an SQL query.

Resilience

As a Grid information system, it is a fundamental requirement of R-GMA that it should tolerate failure of any of its components. There are three aspects to this:

- avoiding single points of failure
- handling failed connections sensibly
- cleaning up links to defunct components

An installation of R-GMA can have as many instances of producers and consumer services as required, but only one registry/schema pair. Replication of the information in the registry and schema is essential to avoid a single point of failure and to provide scalability. Replication of the registry has already been implemented. Several identical replicas can be set up in a single VO, and the Registry API will pick the closest working one. Each registry replica is responsible for its own data and each record in the registry is marked with the URL of the registry replica which originally created it. Each registry periodically sends replicate requests to all other registry replicas in the VO. Each only sends records which it originally created, and it only sends them once to each other replica, unless a replica requests a full set (a checksum is used to allow each replica to determine whether or not it has a full set). Time delays may produce discrepancies between replicas, but these disappear over time, and R-GMA services which use the registry are designed to tolerate them.

Schema replication requirements are different (virtual table creation must be an atomic process across all replicas), and a scheme involving a single, reconfigurable master is being developed.

Each time an R-GMA service is contacted by the API, or by another R-GMA service, a check is made that the service still exists, before an operation is carried out. Where possible, R-GMA will attempt to use an alternative service (e.g. a registry replica, or a different set of producers capable of answering a consumer's query) before gracefully failing, if that is not possible or sensible.

R-GMA services maintain state (e.g. connection details) for each producer and consumer instance in the system. Producer and consumer services, and the registry, must protect

themselves against an accumulation of broken connections; that is, clients that have registered then failed. To achieve this, R-GMA uses soft-state registration, which requires user-applications to maintain periodic contact with their instances, and instances to maintain periodic contact with registry, in order to avoid being timed-out and disconnected. The services keep producer and consumer instances registered with the registry by sending periodic “keep alive” requests on their behalf, and user-applications keep instances alive by making periodic contact (of any sort) with them through the API.

Increasing resilience, optimizing resource use (especially memory and threads) and avoiding synchronisation failures (e.g. deadlocks) were the principal challenges in developing R-GMA. A good level of performance and stability was achieved by the end of the DataGrid project. Some examples of systems which tested R-GMA are presented in the next section.

Practical experience with R-GMA

We have avoided describing in detail the terminology of R-GMA services in this paper, because the EDG terminology is about to change. However some definitions are needed in order to give concrete examples. In the examples which follow, a *Stream Producer*¹ is a producer which publishes tuples in a continuous stream, directly to one or more consumers. A *Database Producer* is one which publishes its tuples into a database and answers one-time consumer queries from there. An *Archiver* is a combined consumer/producer which essentially merges streams of tuples from one or more producers into a single database, which it then re-publishes (usually as a Database Producer) back to the original virtual table.

CMS

R-GMA has been tested by a number of high energy physics projects, including BaBar (based at the Stanford Linear Accelerator Center; interfacing R-GMA to Ganglia), D0 (based at Fermilab – with R-GMA testing done at NIKHEF) and CMS (at CERN).

CMS (Compact Muon Solenoid) is one of the four experiments which will make use of the Large Hadron Collider (LHC) facility, due for

¹ In the new terminology, Stream and Database Producers are examples of *Primary Producers*, and an Archiver is now a *Secondary Producer*.

completion at CERN in 2007. CMS data processing will be grid-based, and a job-monitoring system is a key part of the grid infrastructure. Job monitoring in CMS has been implemented in the BOSS (Batch Object Submission System), which wraps all jobs in such a way that it can monitor all input, output and error streams, and records logging information in a central database.

Originally designed to operate with a local batch farm, it was extended to a Grid environment by using R-GMA to feed information back from remote sites to a central database. Stream Producers at each remote site send data to a single, central Archiver, which writes the data to the BOSS database.

This was found to be a flexible and robust solution, with the added benefit of using standard Web protocols (avoiding firewall configuration problems) [4].

At full data production rates, CMS will need R-GMA to support input from around 2000 simultaneous Monte Carlo data production jobs. Ref [4] describes in detail the work done to test this. A Java application was created to simulate the data production, producing status messages with the right content and at the right frequency to simulate a real job. Since there was no real processing going on, it was possible to use multiple threads on a small number of client systems to simulate full loading.

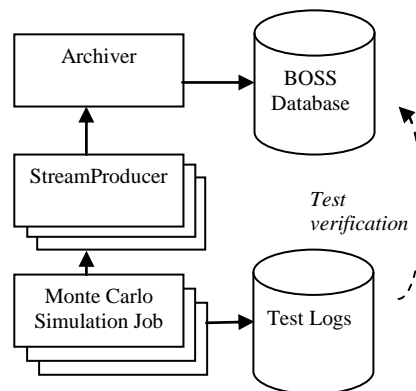


Figure 2

Each thread had its own instance of an R-GMA Stream Producer, and a single R-GMA Archiver was used to collect the information into a database (see Figure 2). The tuples collected were then compared to job log files to ensure no data was lost. Initial results (reported in [4]) showed that R-GMA could only support about a

fifth of the required load, but following improvements to the code and to the Tomcat configuration, R-GMA successfully exceeded the target load. In recent tests, R-GMA has been shown to support 6000 simultaneous jobs across two clients.

DataGrid middleware using R-GMA

R-GMA replaced Globus MDS as the link between the Storage/Computing Elements and the Resource Broker on the DataGrid testbed.

The information requested by the Resource Broker is used to locate suitable SEs and CEs for a job. Fitting R-GMA into the existing LDAP-based MDS infrastructure was achieved by creating two converters, GIN and GOUT. GIN uses an R-GMA Stream Producer to interface to the existing LDAP information providers (CE's and SE's). GOUT uses an R-GMA Archiver to publish to an LDAP database for the Resource Broker. The Replica Manager also uses R-GMA, interfacing to it directly.

R-GMA went through some difficult periods during early development, but monitors set up to measure the performance of GOUT found that R-GMA was achieving good results by the end of the project, when, for at least 95% of queries, response times were consistently better than 10 seconds, information loss (from LDAP to R-GMA and back) was zero when properly configured, and information availability (for 40 sites) through GOUT was 100%.

Network Monitoring (EDG Work Package 7)

Network monitoring is one of the DataGrid middleware services built on top of R-GMA. The network monitoring system is described in detail in the WP7 final report (see [6]), and there is more technical information in the WP7 Web pages. The system has three layers: *measurement*, *data collection and storage*, and *presentation*. The measurement layer uses three different tools (PingER, IperfER, UDPmon) running on network monitoring nodes (NMs) to measure packet loss, round-trip times and throughput. The data collection and storage layer collates these measurements and stores them in a central MySQL database. Data transfer times derived from the GridFTP logs on storage elements (SEs) are also collected. The presentation layer contains tools for visualising and processing the data.

The metrics are used by network managers to monitor the health of the network (using tools such as *MapCenter* developed by WP7) and to test the quality of service. WP7 also developed a *network cost estimation service* which provided a cost metric for bulk-transfers of data between any two nodes on the Grid (based on transfer times derived from GridFTP logs). This service was used by other middleware, in particular the Resource Broker (to assist in the match-making process) and the Replica Manager (to locate optimal replicas).

R-GMA provides the link between the network monitoring nodes (NMs) in the measurement layer, and the MySQL archive in the data collection and storage layer, as shown in Figure 3. Each NM has an instance of an R-GMA Stream Producer, and a single Archiver on the user-interface node brings all of the data into single database. There is a separate (virtual) table (such as NetworkTCPThroughput and NetworkICMPPacketLoss) for each network metric collected.

R-GMA is also used to publish the tables which cross-reference CEs (computing elements) and SEs to their network monitoring node, to allow measurements between specific CEs and SEs to be estimated.

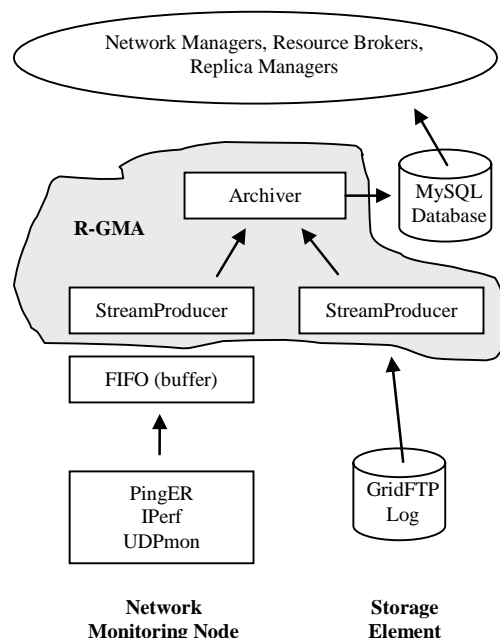


Figure 3

By 12 May 2004 (the last date for which figures are available on the EDG web site), the network

monitoring software had been installed on 56 network monitoring nodes (representing 270 grid elements). Over 650000 GridFTP measurements (coming directly from R-GMA producers running on SEs) and over a million network performance measurements (coming from R-GMA producers on the NMs), had been collected since June 2003, via R-GMA. An outstanding issue is that R-GMA doesn't (currently) guarantee delivery of every published tuple (e.g. in the event of a lost connection), so a complete set of metrics may not always be available for any given time-stamp.

LCG Accounting

R-GMA is being tested by the accounting services for LCG (LHC Computing Grid). Event log files are processed on LCG CE/SE sites to generate accounting records which are sent, using an R-GMA Stream Producer, to the Grid Operations Centre (GOC). R-GMA is being used essentially as a transport mechanism in this configuration. The system has been implemented, and testing will follow R-GMA's certification for use in LCG2.

EGEE - Production Quality Services

The DataGrid project was completed on 31 March 2004. The final version of R-GMA produced by that project had reached a good level of stability and continues to be supported.

The EGEE project (Enabling Grids for E-science in Europe) was launched on 1 April 2004 to take forward the work of DataGrid, to produce a set of production quality middleware. The primary customer will be LCG, but the scope of EGEE is broader, encompassing for example, biomedical applications as well.

R-GMA is providing the information and monitoring services component of EGEE. The duration of the project will be two years, and during that time, the emphasis will be on software quality and stability, rather than new development, although some changes to the R-GMA interface (new API and a Web Services interface) will occur. The next sections outline how R-GMA will be taken forward under EGEE.

Web Services Interface

R-GMA is moving to a Web Services interface. It will use SOAP messaging over HTTP for all user-to-service and service-to-service

interactions². A WSDL document will describe the Producer, Consumer, Registry and Schema services, in machine-readable form. For each instance of a producer, consumer, registry replica or schema replica in an R-GMA installation, there will be a corresponding *resource* on a server, holding the state of that particular instance. Borrowing from the ideas in the proposed Web Services Resource Framework (WSRF) standard, R-GMA will use a *resource framework* to manage the lifetime of its resources. Each resource on a server will have a unique ID by which user-code can identify it, when requesting an R-GMA service to operate on the resource.

Architecturally, this is little different from the existing implementation of R-GMA. Figure 4 shows the old and new interfaces, where the DataGrid (servlet) implementation is shown on the left, and the EGEE (Web Services) one is shown on the right.

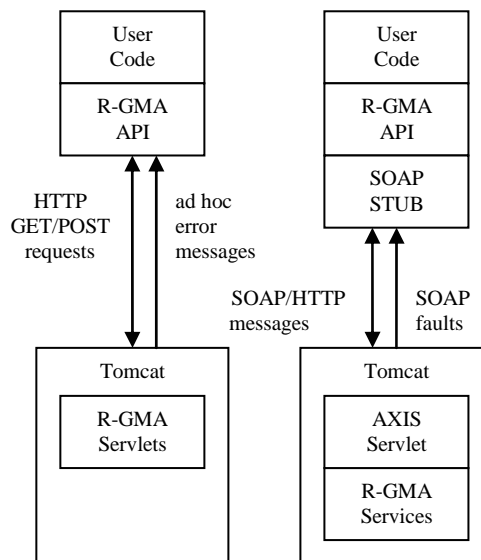


Figure 4

(AXIS is third-party software from Apache, which handles the service-end of the interface and provides tools to generate the client-end code). The old R-GMA interface was characterized by a small number of well-defined and relatively complex operations, initiated by an exchange of messages, with results returned in an XML format. Instances, each with a (locally) unique connection ID were created to represent each producer and consumer at the

² Streaming of tuples from a producer to a consumer uses a direct connection, for efficiency.

server-end. Since all of this translates directly into the Web Services version, performance should be comparable.

It is expected that users will continue to access R-GMA services through the API, rather than directly. Implementations will be provided for the Java, C++, C and Python programming languages. No essential R-GMA functionality will be implemented on the client-side (the API is very thin), but the API will assist user-code with such things as client authentication.

R-GMA will not explicitly adopt WSRF in its first release, as the development of that standard is still unstable. However, we continually monitor relevant standards work, and are actively contributing within GGF to the development of the OGSA architecture, which contains an Information Service as part of its core infrastructure [7].

Software Quality

Formal software development methods are now being applied, to bring the R-GMA code up to production quality. This gives us a chance to clean up the implementation that evolved during the DataGrid project, and to deal with the resource management and synchronisation issues which existed in early versions of the system. We are currently in the specification/design phase, and the work is well advanced. Architecturally, R-GMA will change little (although the API is simpler). R-GMA services will continue to have an object-oriented implementation, written in Java.

Explicit standards for code development are imposed, including coding style. All code will be peer-reviewed. Portability will be improved by the EGEE requirement for support for both Linux RHE and Microsoft Windows XP. For R-GMA, this will mostly affect the installation scripts and some of the API (the rest being written in Java).

The EGEE project has central standards for configuration management, ensuring the reproducibility of builds. Software testing and packaging are also being centralised. A clear split between installation and configuration is required by the project, as are standard tools for the configuration of all EGEE middleware. This, together with good quality documentation, should help spread EGEE middleware into applications beyond LCG and high-energy physics.

Security

The new R-GMA will be designed from the start to fit in with the EGEE security infrastructure (although EGEE security design work is continuing in parallel which is an awkward dependency).

All access to R-GMA is through its Web Services interface (except for streaming, as noted earlier). R-GMA requires mutual authentication between user code (via API) and services, and between pairs of services, at each message exchange. In addition, a secure transport protocol (https) is used for all network transfers. This much was implemented (although not widely used) in the DataGrid project, and ensures that unauthenticated clients cannot access the system (assuming Tomcat is secure), that unauthenticated services can't fool clients, and that the confidentiality and integrity of data in transit is guaranteed.

However there are many other issues to address, such as denial of service, encryption of data within middleware services, and authorisation (deciding what data/services authenticated users are allowed to access). It can be expected that project wide solutions (or at least, the infrastructure on which to build solutions) will be found for many of these issues. However, there are some particularly tricky problems for R-GMA. For example, answering SQL queries requires at least partial access to the contents of the data being returned (to calculate a SELECT statement's column-expression and to match its predicate) which becomes a problem if a user wishes that data to be encrypted.

Authorization also turns out to be remarkably complex. A producer wishes to control access to rows of data it provides, and possibly to specific columns within those rows. Different producers for the same (virtual) table may have different policies. The mediator must know each producer's policy to ensure that it selects producers that not only can, but are willing, to answer a consumer's query. A proposed solution is to control access to virtual tables by creating *views* in the schema and associating required security credentials with each view (like granting user-access to a view in a normal RDBMS), although this requires each producer to register, in advance, each new user/group/role/VO that it might wish to access its data, and makes mediation quite tricky. An alternative solution is to associate security requirements (in the form of an access control

list (ACL) with each tuple. This has the required granularity, and is simple to implement, but is also very inefficient in data terms, and puts the onus back onto the original supplier of the data. An added complication is that a single R-GMA query may involve a number of service-to-service interactions, requiring a trust relationship that, in a loosely coupled system, can be quite hard to define. Altering permissions on the fly (particularly for data already cached in various R-GMA services) also requires some design work.

In order to attract interest from applications beyond high-energy physics (where security requirements are not especially stringent), effective solutions must be found, and experience suggests, must be designed in from the start.

Mediator/SQL processing improvements

Mediation is central to R-GMA: it is the process which makes a dynamic, and potentially unreliable set of producers look like a single, virtual database. In order to ensure that query answers are always as complete and correct as possible, the mediator currently makes various simplifications, such as not using producers with predicates (restricted views) for certain types of query, and not using re-publishers for continuous queries. It is intended to use the opportunity of the redesign to include improvements to the mediator and the SQL command processor.

Conclusion

The design of R-GMA has been shown to meet the needs of a Grid Information and Monitoring system, both for Grid middleware (such as network monitoring), and external applications (such as CMS). It has been extensively tested within EDG, and has been found to work well. Further strengthening within EGEE over the next two years will result in the delivery of *production quality* information services for the Grid.

2002-508833, and funding from PPARC through the GridPP project, are acknowledged. Thanks also to Paul Meador (UCL) for clarification on the use of R-GMA in WP7.

-
1. DataGrid project URL: <http://eu-datagrid.web.cern.ch/eu-datagrid/>
 2. A. Cooke, et al., *Relational Grid Monitoring Architecture (R-GMA)*, presented at UK e-Science AHM, September 2003. (http://www.gridpp.ac.uk/papers/ah03_148.pdf).
 3. A. Cooke, A. Gray et al., *R-GMA: An Information Integration System for Grid Monitoring*, in Proceedings of the Tenth International Conference on Cooperative Information Systems (2003).
 4. B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski and M. Swany, *A Grid Monitoring Architecture*, GGF 2001. (<http://www.didc.lbl.gov/ggf-perf/gmawg/papers/gwd-gp-16-3.pdf>).
 5. H. Tallini, et al., *Scalability Tests of R-GMA based Grid job monitoring system for CMS Monte Carlo Data Production* presented at IEEE NSS 2003. (<http://www.gridpp.ac.uk/papers/ieeetalk-v4.pdf>) + more recent test results
 6. F. Bonnassieux, *Final Report on Network Infrastructure and Services*, deliverable for DataGrid project 2003. (<http://edms.cern.ch/document/414132>)
 7. *The Open Grid Services Architecture v.018*, <https://forge.gridforum.org/projects/ogsa-wg/docman>



Acknowledgements

EU support through DataGrid under contract IST-2000-25182 and EGEE under contract IST-