R-GMA: A Relational Grid Information and Monitoring System

Rob Byrom[4], Brian Coghlan[7], Andrew Cooke[1], Roney Cordenonsi[3], Linda Cornwall[5], Ari Datta[3], Abdeslem Djaoui[5], Laurence Field[4], Steve Fisher[5], Steve Hicks[4], Stuart Kenny[7], James Magowan[2], Werner Nutt[1], David O'Callaghan[7], Manfred Oevers[2], Norbert Podhorszki[6], John Ryan[7] Manish Soni[4], Paul Taylor[2], Antony Wilson[4] and Xiaomei Zhu[4]

[1] Heriot-Watt, Edinburgh, UK
[2] IBM-UK
[3] Queen Mary, University of London, UK
[4] PPARC. UK
[5] Rutherford Appleton Laboratory, UK
[6] SZTAKI, Hungary
[7] Trinity College Dublin, Ireland

Abstract

We describe R-GMA (Relational Grid Monitoring Architecture) which is being developed within the European DataGrid Project as a Grid Information and Monitoring System for both the grid itself and for use by applications. Is is based on the GMA from GGF, which is a simple Consumer-Producer model. The special strength of this implementation comes from the power of the relational model. We offer a global view of the information as if each VO had one large relational database. We provide a number of different Producer types with different characteristics; for example some support streaming of information. We also provide combined Consumer/Producers, which are able to combine information and republish it. At the heart of the system is the mediator, which for any query is able to find and connect to the best Producers to do the job. In addition to having some of our own sensors able to publish information, we are able to invoke MDS info-provider scripts and publish the resulting information via R-GMA.

1 Introduction

To provide the motivation for this work, we first explain why we feel that the existing LDAP based information system solutions are not very suitable to use as the basis of a Grid information and monitoring system.

1.1 LDAP

We believe that there should be a common interface to both information and monitoring systems. It would be permissible to have two underlying systems provided that this is not discernible to the end user. It is not even clear how

to distinguish between an information system and a passive monitoring system. We have argued before[2] that the only thing which characterises monitoring information is a time stamp, so we insist upon a time stamp on all measurements - saying that this is the time when the measurement was made, or equivalently the time when the statement represented by the tuple was true. It is very convenient to be able to subscribe to information rather than having to poll; this is not supported by the LDAP standard.

It is notable that there are essentially no commercial hierarchical database systems still in existence. The only remaining hierarchical systems are based on LDAP and the older X.500 directory system which LDAP (as a lightweight version of X.500 was meant to replace). The standard today in most fields is the RDBMS. The fundamental reason for this is that, though many systems can be *approximately* modelled as a hierarchy, there are very few pure hierarchies in the real world; in many organisations people don't have just one manager.

The LDAP DIT is normally centrally organised and there is no provision for a user being able to *define* and publish his own data. A table can stand alone, but adding an object class to the DIT is potentially intrusive.

The most obvious drawback of the LDAP model is that it does not support queries which combine information across objects of different class - i.e. there is no *join* concept. The consequence of this is that it is only possible to answer queries for which the tree structure has been defined. Evolution of relational schemas is not easy - but is easier than trying to fix a tree structure which has existing users.

1.2 GMA and R-GMA

We find the Grid Monitoring Architecture (GMA)[3] of the GGF to be very simple and attractive. The GMA, as shown in Figure 1, consists of three components: *Consumers*, *Producers* and a directory service, which we prefer to call a *Registry*).

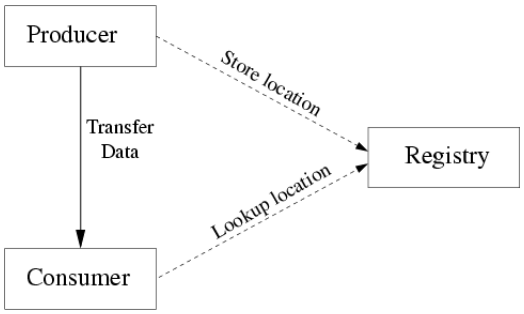

Fig. 1: Grid Monitoring Architecture

In the GMA Producers register themselves with the Registry and describe the type and structure of information they want to make available to the Grid. Consumers can query the Registry to find out what type of information is available and locate Producers that provide such information. Once this information is known the Consumer can contact the Producer directly to obtain the relevant data. By specifying the Consumer/Producer protocol and the interfaces to the Registry one can build inter-operable services. The Registry communication is shown on Figure 1 by a dotted line and the main flow of data by a solid line.

The current GMA definition also describes the registration of Consumers, so that a Producer can find a Consumer. The main reason to register the existence of Consumers is so that the Registry can notify them about changes in the set of Producers that interests them.

The GMA architecture was of course devised for monitoring but we think it makes an excellent basis for a combined information and monitoring system.

The GMA does not constrain any of the protocols nor the underlying data model, we were free when producing our implementation to adopt a data model which would allow the formulation of powerful queries over the data.

Our implementation creates the impression that you have one RDBMS per VO. However it is important to appreciate that what our system provides is a way of using the relational model in a Grid environment and that we have *not* produced a general distributed RDBMS. All the producers of information are quite independent. It is relational in the sense that Producers announce what they have to publish via an SQL CREATE TABLE statement and publish with an SQL INSERT and that Consumers use an SQL SELECT to collect the information they need.

R-GMA is built using servlet technology and is being migrated rapidly to web services and specifically to fit into an OGSA[4] framework.

2 Producer Types

We have so far defined not just a single Producer but five different types: a DataBaseProducer, a StreamProducer, a ResilientProducer, a LatestProducer and a CanonicalProducer. All appear to be Producers as seen by a Consumer - but they have different characteristics. The CanonicalProducer, though in some respects the most general, is somewhat different as there is no user interface to publish data via an SQL `INSERT` statement. Instead it triggers user code to answer an SQL query. The other Producers are all `Insertable`; this means that they all have an interface accepting an SQL `INSERT` statement.

The other producers are instantiated and given the description of the information they have to offer by an SQL `CREATE TABLE` statement and a `WHERE` clause expressing a predicate that is true for the table. Currently this is of the form `WHERE (column_1=value_1 AND column_2=value_2 AND ...)`. To publish data, a method is invoked which takes the form of a normal SQL `INSERT` statement.

The DataBaseProducer writes each record to an RDBMS. This is slow (com-

pared to a StreamProducer) but it can handle joins. The StreamProducer writes information to a memory structure where it can be picked up by a Consumer. The ResilientProducer is similar to the StreamProducer but information is backed up to disk so that no information is lost in the event of a system crash. The LatestProducer also makes use of an RDBMS but only holds the latest records.

Latest records are defined in terms of something similar to a primary key. Each record has a time stamp, one or more fields which define what is being measured (e.g. a hostname) and one or more fields which are the measurement (e.g. the 1 minute CPU load average). The time stamp and the defining fields are close to being a primary key - but as there is no way of knowing who is publishing what across the Grid, the concept of primary key (as something globally unique) makes no sense. The LatestProducer will replace an earlier record have the same defining fields and a time stamp which is not later than the new time stamp.

Producers may need cleaning from time to time. This is especially the case for those using an RDBMS. We provide a mechanism to specify those records of a table to delete by means of a user specified SQL `WHERE` clause which is executed at intervals which are also specified by the user. For example it might delete records more than a week old from some table or it may only hold the newest one hundred rows, or it might just keep one record from each day.

Another valuable component is the Archiver which is a combined Consumer-Producer. You just have to tell it what to collect and it does so on your behalf. It will re-publish to any kind of "Insertable". This allows useful topologies of components to be constructed.

3 Applications of R-GMA

R-GMA has applications right across the Grid. First it can be used as a replacement for MDS. A small tool (GIN) has been written to invoke the MDS-like EDG info-providers and publish the information via R-GMA. Another tool (GOUT) is available to republish R-GMA data to an LDAP server for the benefit of legacy applications. However we expect that most applications will wish to benefit from the power of relational queries.

An example of the a possible use by the other middleware components is shown in Figure 2. The BS components (not part of R-GMA) are publishing status information on jobs. The first layer of Archivers is collecting together jobs for a particular group of users (as defined by their name) and republishing it via a StreamProducer. This information can be used by people who wish to continuously monitor the state of one or more of their jobs. Another layer of Archivers is taking the information from the StreamProducer in the first layer and republishing via a LatestProducer. This only shows the current state of a job. It can hold this state for as Long as is defined by the cleanup policy for the LatestProducer.

R-GMA is also being used for network monitoring and to locate replica catalogs. GRM was written for mounting parallel applications[1] where it writes

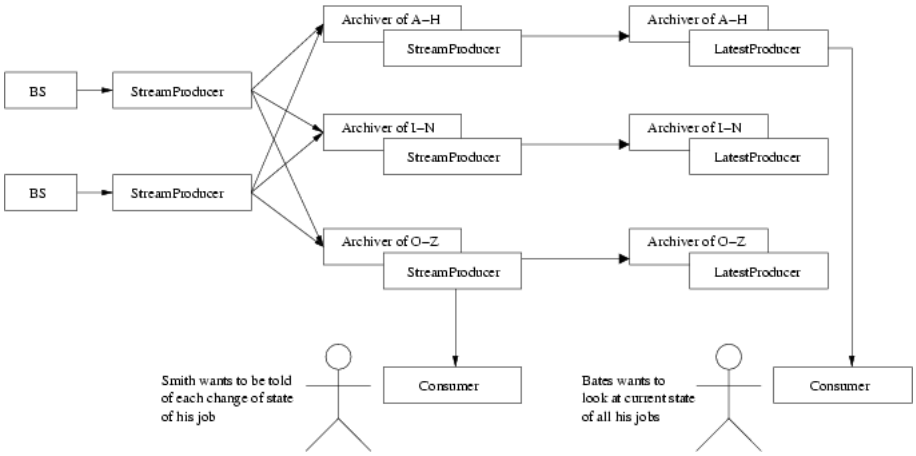

Fig. 2: R-GMA for job information

logging information to a local file. This has recently been modified to make use of R-GMA for transport.

4 Tools

There are a number of tools available to query R-GMA Producers. These are all consumers with different user interfaces. There is a a line mode query: `edg-rgma`, a Java application: Pulse (show below in Figure 3): and a JSP application: the R-GMA Browser which is accessible from a Web browser without any R-GMA installation. The Browser offers a few custom queries, but makes it easy for you to write your own.

5 The registry and the mediator

The registry stores information about all producers currently available. Currently there is only one physical Registry per VO. This bottleneck and single point of failure is being eliminated. Code is being written to allow multiple copies of the registry to be maintained. Each one acts as master of the information which was originally stored in that Registry instance and has copies of the information from other Registry instances. Synchronisation is carried out frequently.

The mediator (which is hidden behind the Consumer interface) is the component which makes R-GMA easy to use. Producers are associated with views on a virtual data base. Currently views have the form:

SELECT * FROM <table> WHERE <predicate>

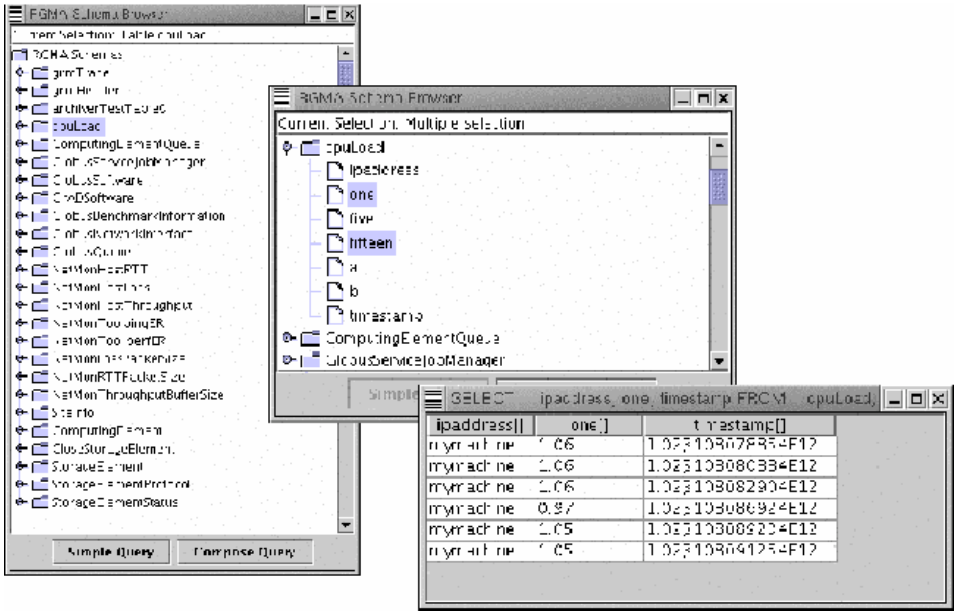

Fig. 3: Pulse

This view definition is stored in the Registry. When queries are posed, the Mediator uses the Registry to find the right Producers and then combines information from them.

6 Architecture

R-GMA is currently based on Servlet technology. Each component has the bulk of its implementation in a Servlet. Multiple hand crafted APIs in Java, C++, C, Python and Perl have been written to communicate with the servlets. We make use of the Tomcat Servlet container. Most of the code is written in Java and is therefore highly portable. The only dependency on other EDG software components is security.

Figure 4 shows the communication between the APIs and the Servlets. When a Producer is created its registration details are sent via the Producer Servlet to the Registry (Figure 4a). The Registry records details about the Producer, which include the description and view of the data published, *but not the data itself*. The description of the data is actually stored as a reference to a table in the Schema. In practise the Schema is co-located with the Registry.

Then when the Producer publishes data, the data are transferred to a local Producer Servlet (Figure 4b).

When a Consumer is created its registration details are also sent to the

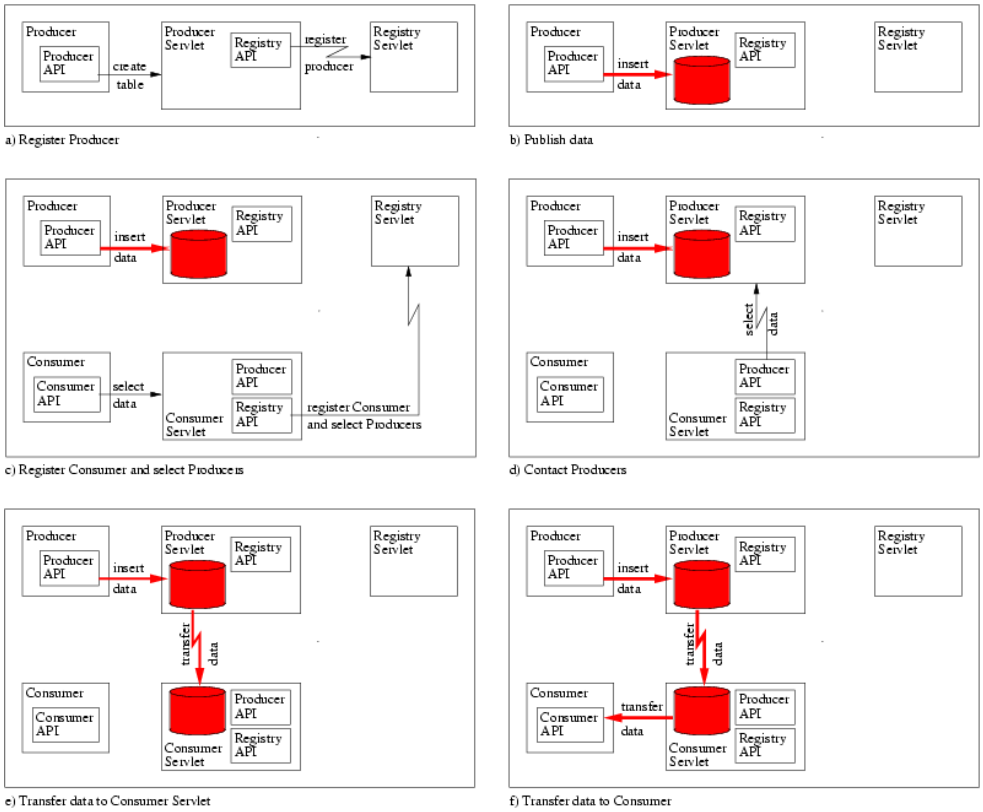

Fig. 4: Relational Grid Monitoring Architecture

Registry although this time via a Consumer Servlet (Figure 4c). The Registry records details about the type of data that the Consumer is interested in. The Registry then returns a list of Producers back to the Consumer Servlet that match the Consumers selection criteria.

The Consumer Servlet then contacts the relevant Producer Servlets to initiate transfer of data from the Producer Servlets to the Consumer Servlet as shown in Figures 4d-e..

The data are then available to the Consumer on the Consumer Servlet, which should be close in terms of the network to the Consumer (Figure 4f).

As details of the Consumers and their selection criteria are stored in the Registry, the Consumer Servlets are automatically notified when new Producers are registered that meet their selection criteria.

The system makes use of soft state registration to make it robust. Producers and Consumers both commit to communicate with their servlet within a certain time. A time stamp is stored in the Registry, and if nothing is heard by that

time, the Producer or Consumer is unregistered. The Producer and Consumer servlets keep track of the last time they heard from their client, and ensure that the Registry time stamp is updated in good time.

We see OGSA as the future of the Grid, and so have started migrating the system to an OGSA framework. We are working towards something similar to the system depicted in Figure 5.

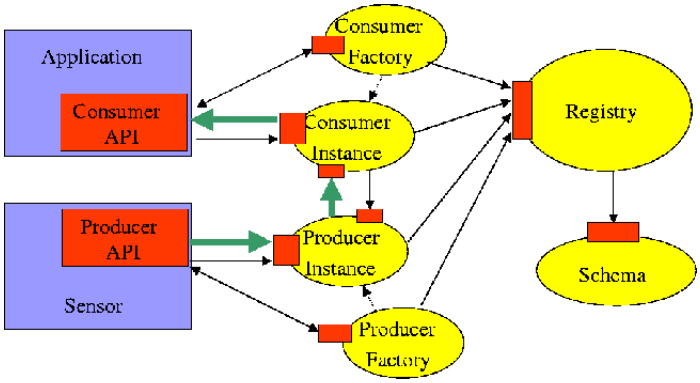

Fig. 5: Forthcoming OGSA R-GMA Architecture

7 Conclusion

We have a useful architecture and an effective implementation with a number of components which work well together. We hope that it will have a long, happy and useful life, both in its current form and when reincarnated within an OGSA framework.

References

1. Zoltán Balaton, Peter Kacsuk, and Norbert Podhorszki. From cluster monitoring to grid monitoring based on grm and prove. Technical Report LPDS-1/2000, Laboratory of Parallel and Distributed System, Hungary, 2000.
2. Brian Coghlan, Abdeslem Djaoui, Steve Fisher, James Magowan, and Manfred Oevers. Time, information services and the grid. In K D Oneill and B J Read, editors, *BNCOD 2001 - Advances in Database Systems*, number RAL-CONF-2001-003 in RAL-CONF. BNCOD, 2001.
3. Brian Tierney, Ruth Aydt, Dan Gunter, Warren Smith, Valerie Taylor, Rich Wolski, and Martin Swany. A grid monitoring architecture. Technical Report GWD-Perf-16-1, GGF, 2001.
4. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, and P. Vanderbilt. Grid service specification. Technical report, GGF, 2002.