INFORMATION AND MONITORING SERVICES WITHIN A GRID ENVIRONMENT

A. J. Wilson, R. Byrom, L. A. Cornwall, M. S. Craig, A. Djaoui, S. M. Fisher, S. Hicks, R. P. Middleton, J. A. Walk, CCLRC - Rutherford Appleton Laboratory, UK
A. Cooke, A. J. G. Gray, W. Nutt, Heriot-Watt University, UK
J. Magowan, P. Taylor, IBM
J. Leake, Objective Engineering Ltd., UK
R. Cordenonsi, Queen Mary, University of London, UK
N. Podhorszki, SZTAKI, Hungry
B. Coghlan, S. Kenny, O. Lyttleton, D. O'Callaghan, Trinity College Dublin, Ireland

Abstract

The R-GMA (Relational Grid Monitoring Architecture) was developed within the EU DataGrid project, to bring the power of SQL to an information and monitoring system for the grid. It provides producer and consumer services to both publish and retrieve information from anywhere within a grid environment. Users within a Virtual Organization may define their own tables dynamically into which to publish data.

Within the DataGrid project R-GMA was used for the information system, making details about grid resources available for use by other middleware components. R-GMA has also been used for monitoring grid jobs by members of the CMS and D0 collaborations where information about jobs is published from within a job wrapper, transported across the grid by R-GMA and made available to users. An accounting package for processing PBS logging data and sending it to one or more Grid Operation Centres using R-GMA has been written and is being deployed within LCG. There are many other existing and potential applications.

R-GMA is currently being re-engineered to fit into a Web Service environment as part of the EU Enabling Grids for E-science in Europe (EGEE) project. Improvements being developed include fine grained authorization, an improved user interface and measures to ensure superior scaling behaviour.

OVERVIEW OF R-GMA

R-GMA is an implementation of the Grid Monitoring Architecture (GMA) [1] proposed by the Global Grid Forum (GGF), which models the information infrastructure of a Grid as a set of Consumers (who request information), Producers (who provide information) and a single Registry (which mediates the communication between producers and consumers). R-GMA imposes a query language (a subset of SQL) on this model - so producers publish tuples (database rows) with an SQL insert statement and consumers query them using SQL select statements. R-GMA also ensures that all tuples carry a time-stamp, so that monitoring systems (which require time-sequenced data) are inherently supported.

R-GMA presents the information resources of a Virtual Organisation (VO) as a single virtual database containing a set of tables. As well as providing Producer and Consumer services R-GMA provides a Registry Service and a Schema Service, both of which will be replicated and each VO will have its own logical registry and schema. A registry contains a list, for each table, of producers who have offered to publish (provide data for) rows for the table. A schema contains the name and structure (column names, types and settings) of each virtual table in the system.

A full description of the R-GMA specification is contained in the Information and Monitoring Service (R-GMA) System Specification [2] and a full description of the architecture is contained in the EGEE Architecture document [3].

R-GMA COMPONENTS

The Producer Service

There are three classes of producer: Primary, Secondary and On-demand. Each is created by a user or middleware application and returns tuples in response to queries from other applications. As figures 1a-c below show, the main difference is in where the tuples originate.

The Producer Service in these figures is a process running on a server on behalf of the user code. In a Primary Producer (figure 1a), the user code periodically inserts tuples into storage maintained internally by the Primary Producer Service. The Producer Service autonomously answers consumer queries from this storage. The Secondary Producer Service (figure 1b) also answers queries from its internal storage, but it populates this storage itself by running its own query against the virtual table: the user code only sets the process running; the tuples come from other producers. In the On-demand Producer (figure 1c), there is no internal storage; data is provided by the user code in direct response to a query forwarded on to it by the Producer Service.

The tuple-storage maintained by Primary and Secondary Producers can either be in memory, in a file, or in a real database table. Producers that use non-database storage are optimized to answer simple queries quickly, but they can support complex queries too (e.g. by creating

an in-memory database on the fly). In an On-demand producer, tuple-storage (if any) is the responsibility of the user code, but may also be in a real database.

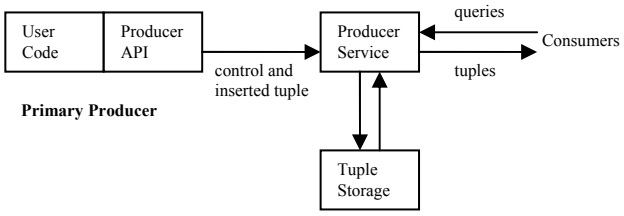

Figure 1a. Publishing via a Primary Producer

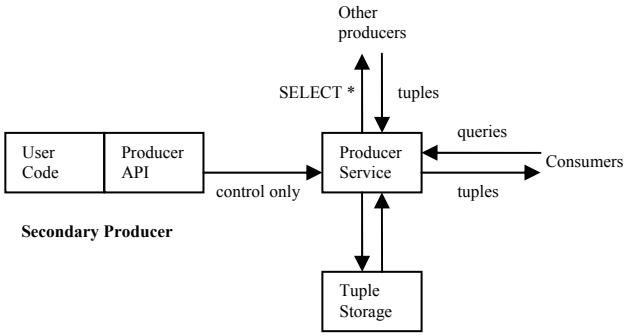

Figure 1b. Publishing via a Secondary Producer

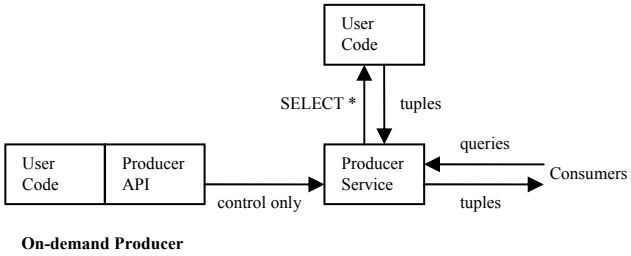

Figure 1c. Publishing via an On-demand Producer

The Consumer Service

In R-GMA, each consumer represents a single SQL SELECT query on the virtual database. The request is initiated by user code, figure 2, but the Consumer Service carries out all of the work on its behalf. The query is first passed to the Registry service to identify which producers, for each virtual table in the query, must be contacted to answer it. This process is called mediation. The query is then passed by the Consumer Service to each relevant producer, to obtain the answer tuples directly. Note that there is no central repository holding the contents of the virtual table; it is in this sense, that the database is virtual.

There are four types of query: continuous, latest, history and static. The set of queries that a particular producer supports is recorded in the registry. All query types except static can take an optional time interval parameter.

A continuous query causes all new tuples that match the query to be streamed into the consumer's tuple storage, as soon as they are inserted into the virtual table by the producers. Streaming continues until the consumer requests it to stop. If a time interval is specified, the consumer will additionally receive any tuples which are already in the virtual table when the query starts, and which are no older than the time interval. There is no guarantee that tuples are time-ordered. All Primary and Secondary producers support continuous queries but On-demand producers do not.

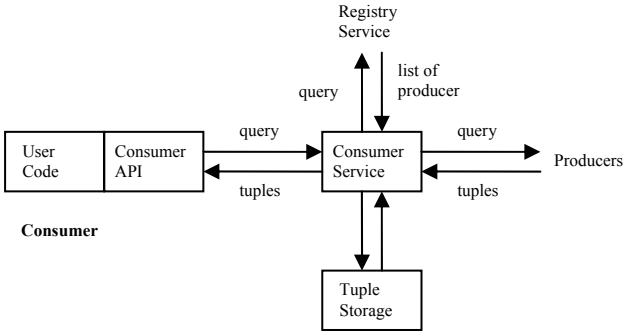

Figure 2. Querying

Latest and history queries are one-time queries: they execute on the current contents of the virtual table, then terminate. In a history-query, all versions of any matching tuples are returned; in a latest-query, only those representing the "current state" are returned, where current state is the most recent versions of tuples which have not exceeded a user defined time period known as the LatestRetentionPeriod.

In both cases, a time interval may be specified with the query, to limit the age of the tuples returned. Primary and Secondary Producers may optionally support one-time queries but On-demand producer do not.

Static queries are only supported by On-demand producers. They are one-off database-like queries and do not contain R-GMA time-stamps. The primary purpose of an On-demand producer is to allow very large data structures to be accessed through the R-GMA infrastructure, without the overhead of copying tuples into a Producer Service.

JOB MONITORING

The role of job monitoring is to enable grid users to monitor the progress of their own jobs and for VO administrators to get an overview of what is happening in the grid. The problem is that the jobs are likely to run remotely behind a firewall and so are not accessible. A solution to this is to use the Producer and Consumer Services of R-GMA to provide transport across firewalls.

There are two approaches to Job Monitoring. A wrapper script can parse the outputs and publish information based on patterns in the output or the user code can be instrumented directly to publish what it is doing. Logically these are the same; it just depends on how the "job" is defined. The job simply wants to announce that something has happened, without being aware of whether or not anything is listening.

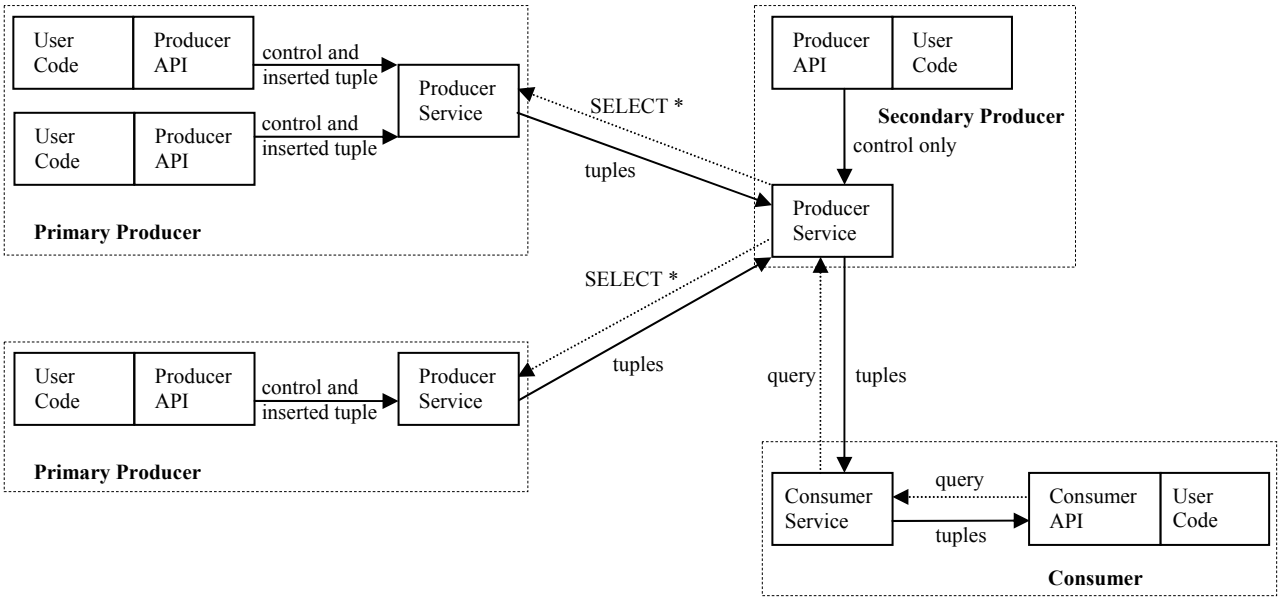

Figure 3. A Primary Producer, Secondary Producer Hierarchy

The Wrapper Approach

We will now describe a generalised method for job monitoring based on the work done by members of the CMS and D0 collaborations. The basic requirement was to be able to monitor the standard out and standard error of running grid jobs.

A wrapper script containing the user's code and calls to R-GMA is submitted to the grid. Upon execution the wrapper script uses the Producer Service to create an R-GMA Primary Producer resource on the local R-GMA server with an in-memory tuple store, which is lightweight and fast. The wrapper script then executes the user's code. Output to standard out and standard error from the user's code is then intercepted by the wrapper script that makes this data available across the grid by publishing it using an R-GMA "insert" statement. The data is transferred to the Primary Producer resource on the local R-GMA server where it is stored in the in-memory tuple store. This data remains in the tuple store even after the application code has terminated for a period dependent upon a user definable variable, RetentionPeriod. After this period has elapsed the data will be discarded from the tuple store.

No prior knowledge of where the job will run is required, data is published from wherever the job lands within the grid. This results in numerous Primary Producers running across the grid, all publishing to the same virtual table. In order to optimize queries and to ensure that a permanent record of the data is kept a Secondary Producer, using a database tuple store, is used to aggregate all tuples published to the virtual table. This Primary Producer – Secondary Producer hierarchy is illustrated in figure 3.

Once the Secondary Producer resource has been created it contacts any existing relevant Primary Producer resources and initiates streaming. As new relevant Primary Producer resources appear they are also automatically contacted to initiate streaming. As new tuples become available in tuple stores of the Primary Producer resources they are automatically streamed to the Secondary Producer resource. Thus the Secondary Producer resource is able to maintain an up to date view of the entire virtual table. Queries across the virtual table are then directed to the Secondary Producer resource by the mediator. This helps minimise the load on the Primary Producers. If the load on the Secondary Producer becomes too high additional Secondary Producers can be set up.

Instrumenting the Code

This may be done by making use of an existing logging service or by instrumenting the code directly with calls to the Producer Service

Logging Service

Within a Java application, the natural way to do job monitoring is with the java logging service or log4j[4]. Following the success of log4j, facilities supporting applications in other languages have been developed and now we see a new Logging Services[5] activity within Apache which "is intended to provide cross language logging services for purposes of application debugging and auditing." A tool, Chainsaw[6], is also part of the Apache logging service project. This is able to pick up logs and visualise them.

Rather than have users instrument their code with calls to R-GMA we are developing an interface to be used with the Apache Logging Services. We are currently developing an appender (an interface) for the Logging Service. When the output of the Logging Service is directed to this appender, it will be automatically published via the Producer Service. In order to make this as light-weight as possible Primary Producer resources with in memory tuple stores will be used. As with the existing method of job monitoring, Secondary Producer

resources will be employed to aggregate the data and provide a long-term storage facility within a database tuple store.

From an application point of view, a message – a single unstructured string is published using the normal logging API which publishes the data via R-GMA. The set of attributes which can be published will follow the Apache Logging Services specification. As indicated, the Logging Service messages are unstructured. If structure is required then R-GMA should be used directly.

Facilities to make the data available to Chainsaw (or equivalent) will also be provided for visualisation.

Instrumenting with R-GMA

There may well be cases when a user does not wish to use the generic data structure provided by the Logging Service but wishes to make use of more complex user defined data structures. If this is the case then the user can use the Schema Service to define tables for use within the virtual database. Once the tables have been registered with the Schema Service then user code instrumented with R-GMA insert statements can be used to populate these tables.

The R-GMA web-based browser can then be used to view the contents of these tables or code can be written to use the Consumer Service to examine the contents of the tables.

THE PERFORMANCE OF R-GMA

Extensive performance testing of the R-GMA components has recently been conducted. The measurements indicated that they have good scaling behaviour.

A single Registry Service resource coped with 125 consumers per second being created and destroyed. The addition of a second, replica, Registry Service resource on a separate machine showed perfect scaling behaviour within the accuracy of the measurements.

The ability of a Consumer Service resource, posing a continuous query, to process different tuple sizes was measured and it was found that individual tuples could be process in a time of 1400µs + 1 µs/byte. It was also found that a single Consumer Service resource was able to consume data from 3200 producers, with no indication of an approaching limit.

A full description of the performance testing along with all of the results can be found in *The Relational Grid Monitoring Architecture: Mediating Information about the Grid* [7]

PAST, PRESENT AND FUTURE

R-GMA was developed and deployed as part of the EU DataGrid project and it has recently been deployed as part of LCG. It is now being taken forward as part of the EGEE project. This includes re-engineered and work to convert R-GMA to web services. A WS-I compliant implementation will be available in the very near future. Work is also in progress on providing multiple VO support and replicating the Schema Service, this requires a different mechanism to the Registry Service replication which has already been implemented.

SUMMARY

R-GMA is a grid information and monitoring system that provides Producer, Consumer, Registry and Schema Services for use by grid users and other components. It is based on a powerful data model and provides the ability for users to define their own schema and to provide and request information using an API based on a subset of SQL. Recent performance measurements demonstrate that it has good scaling behaviour. The system has been deployed within LCG and is now being re-engineered as part of the EGEE project

ACKNOWLEDGMENTS

This work was funded through contributions from the European Commission program INFSO-RI-508833 through the EGEE project and by PPARC through the GridPP program.

REFERENCES

[1] B. Tierney et al. A Grid Monitoring Architecture. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/documents.html, GGF, 2002.

[2] JRA1-UK. Information and Monitoring Service (R-GMA) System Specification. https://edms.cern.ch/document/490223/, EGEE, 2004.

[3] JRA1 Design Team. EGEE middleware architecture. Technical Report https://edms.cern.ch/document/476451/, EGEE, 2004.

[4] Apache Software Foundation. log4j. http://logging.apache.org/log4j/.

[5] Apache Software Foundation. Logging Services. http://logging.apache.org/.

[6] Apache Software Foundation. Chainsaw. http://logging.apache.org/log4j/docs/chainsaw.html.

[7] A. Cooke et al. The Relational Grid Monitoring Architecture: Mediating Information about the Grid. Submitted to Journal of Grid Computing.