Fault tolerance in the R-GMA Information and Monitoring System

Rob Byrom[4], Brian Coghlan[6], Andy Cooke[1], Roney Cordenonsi[3], Linda Cornwall[4], Martin Craig[4], Abdeslem Djaoui[4], Alastair Duncan[4], Steve Fisher[4], Alasdair Gray[1], Steve Hicks[4], Stuart Kenny[6], Jason Leake[7], Oliver Lyttleton[6], James Magowan[2], Robin Middleton[4], Werner Nutt[1], David O'Callaghan[6], Norbert Podhorszki[5],Paul Taylor[2], John Walk[4], Antony Wilson[4].

1: Heriot-Watt University, 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 (Relational Grid Monitoring Architecture) [1] is a grid monitoring and information system that provides a global view of data distributed across a grid system. R-GMA creates the impression of a single centralised repository of information, but in reality the information can be stored at many different locations on the grid. The Registry and Schema are key components of R-GMA. The Registry matches queries for information to data sources that provide the appropriate information. The Schema defines the tables that can be queried. Without the combined availability of these components, R-GMA ceases to operate as a useful service. This paper presents an overview of R-GMA and describes the Registry replication design and implementation. A replication algorithm for the Schema is also described.

1 Introduction

R-GMA is a monitoring and information application for grid environments. It is an implementation of the Grid Monitoring Architecture [3] proposed by the Global Grid Forum [4]. R-GMA was developed within the European Datagrid [2] project. It is currently being re-engineered as web services, within the EGEE (Enabling Grids for E-Science in Europe) project

2 Grid Monitoring Architecture

The Grid Monitoring Architecture models the information infrastructure of the grid as a set of Producers that provide information, and a set of Consumers that request information. A Registry manages interaction between Producers and Consumers. Producers contact the Registry to store information about the data they provide. Consumers contact the Registry to find Producers that publish information they require. The Registry returns to the Consumer a list of Producers that provide the desired information and the Consumer then contacts the appropriate Producer or Producers to obtain this information.

3 Virtual Database

R-GMA presents a global view of the information produced by the components of a grid system and by applications running on the grid. It presents this information as a virtual database, introducing the relational data model to the Grid Monitoring Architecture. All queries to the R-GMA virtual database must be valid SQL SELECT statements. To a user who queries this virtual database, it seems as though they are querying a single, centralised repository of information. In actual fact, the information obtained from this virtual database can be stored at a number of different locations.

Each table has a *key* column (or group of columns). In addition, each tuple inserted by a Producer has a timestamp added to it by the Producer. The combination of this timestamp and the key columns is similar to a primary key for the table. This is essential if R-GMA is to be used as a monitoring system, as time-sequenced data is required to implement such a system. Users must be able to request data published within a particular time interval (for example, all rows from a table published in the previous 20 minutes). The timestamp associated with each inserted tuple is used to determine the time at which the tuple was published.

4 Producer Service

The Producer service is used to publish data to the virtual database. Producers register themselves with the Registry by declaring their intention to publish to a table specified in the Schema. The Producer service obtains the table structure from the Schema and creates a table with identical structure in its own storage. Data in the Producer’s storage has a retention period associated with it. When data has been in the storage for longer than this retention period, it is deleted. There are three kinds of Producers. The difference between them is where the data they publish is coming from.

Primary Producer: The user's code inserts tuples directly into the Producer’s storage.

Secondary Producer: The Secondary Producer republishes tuples that have already been published by another Producer. It queries Primary Producers and other Secondary Producers and inserts the tuples returned from these queries into its own storage. This can be useful for creating a service that archives the data from multiple Producers, or for answering queries that require joins by combining tables from multiple Producers.

OnDemand Producer: It may not be practical to transfer a large volume of data to a Producer with SQL INSERT statements. The OnDemand Producer allows a user to request such information when it is required. Only then is the information encoded in tabular form and sent to the user. The OnDemand producer has no tuple storage facility. It sends all queries it receives to additional user code that sends tuples back to the consumer.

5 Consumer Service

Consumers allow users to execute SQL queries on the R-GMA virtual database. The Consumer contacts the Registry to find the Producers required to answer the query. It then sends the query to these Producers and collects the tuples returned into an internal store for subsequent retrieval by the user. Consumers allow four types of query:

Continuous: All tuples matching a query are to be automatically streamed to the Consumer when they are inserted into the table. All Primary and Secondary Producers support continuous queries, but OnDemand Producers do not. An example of where receiving a continuous stream of data from a producer is of use would be a real-time job monitoring application, which must update the status of jobs on the grid as they are executing.

Latest: The most recent version of a tuple matching a query is returned to the user. A resource broker which decides where jobs should be executed on a grid may use a Latest producer. The resource broker needs up-to-date information about the resources on the grid. It is only interested in the most recent state of the resources, not their state over a period of time.

History: All available tuples matching a query are returned. This is useful when the Producer is to be used as part of an archiving application.

Static: These are specific to OnDemand Producers, and are handled like a normal database query. Note that neither timestamps nor intervals are associated with static queries.

6 Registry Service

The Registry enables R-GMA to match Consumers to Producers that provide the information they require. Producers advertise what tables and parts of tables in the virtual database they produce rows for. Consumers can consult the registry to find Producers that can answer their queries. The process by which the Registry finds Producer(s) that are capable of providing information requested by a Consumer is called mediation. Mediation allows a Consumer to have a global view of information from sources distributed across the grid.

7 Schema Service

R-GMA uses a Schema service to define the tables that comprise the virtual database. It also defines the authorization rights for these tables. The Schema service allows operations such as adding tables to or removing tables from the virtual database, or getting the name, type and attributes of all columns in a particular table.

8 Namespaces

The term “Virtual Organisation” (VO) has been formulated to represent a set of resources that are shared by a number of users, and rules that specify user’s access rights to resources. Individuals and institutions working on the same problems (for example, a community of researchers working in a particular area, such as High Energy Physics or Earth Observation), benefit from pooling their resources and making them available to all members of the community. It is possible that a user can be a member of more than one VO.

R-GMA allows users to issue queries to multiple VOs. One VO may have a table that has the same name as one in another VO but have different attributes. Clearly a global namespace defined by just the table names is not scalable. The R-GMA solution is to form a virtual global namespace by giving each VO its own disjoint subset of the namespace. Information particular to each VO is contained in a virtual database, the name of the virtual database being that of the VO. To provide a scalable implementation, each VO has its own set of schema and registry services. To issue a query to a particular VO using a consumer, the table name in the SQL query must be prefaced with the VO name. For example, for the LCG VO:

```
SELECT NumberOfJobsRunning FROM LCG.Workload WHERE
Site='RAL'
```

For publishing information we could adopt the same syntactic approach. However as it is a common pattern to publish the same information to multiple VOs, the set of VOs to which a tuple is published is included as a separate parameter in the Producer API function call, and is not included in the SQL INSERT statement.

9 Registry Replication

Although there is only one logical Registry per VO, replicas of the Registry are maintained. If a particular Registry fails at any time, an alternative Registry is used instead. This enables the client (i.e. the Consumer or Producer) to seamlessly carry out a Registry operation in spite of any faults that occur within individual components of the replicated Registry group.

9.1 Selecting a Registry

Clients wishing to connect to a Registry do so by using a selection algorithm, which is carried out internally within the Registry API. A list of available Registry services is provided via a configuration file on the client. During runtime, a simple profile is created by contacting each Registry service identified in the configuration. The quickest response time is then used to select the Registry to handle the client request. This Registry will then be re-used for all further client interaction unless the Registry becomes unavailable - at which point a new one is selected using the same procedure.

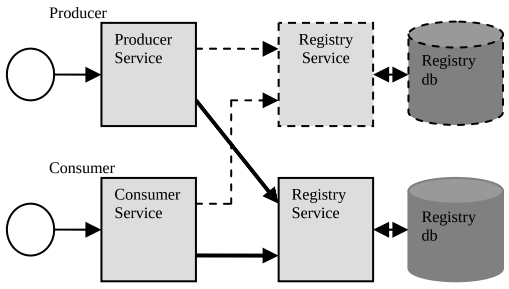

Fig. 1. If a Registry becomes unavailable, an alternative is used.

9.2 Registry Replication Design

Registry replication is based on a distributed model where each Registry pushes locally acquired data to its peers. New data is obtained when a Consumer or Producer carries out a registration process. When this occurs, an entry is copied into the local Registry database along with a *replica status* flag indicating the data is fresh. An additional *origin* tag is added to each entry that identifies the Registry where the new data was added. A dedicated thread that runs periodically on each Registry replica initiates the replication process. It checks the existence of newly arrived registrations by reading the replica status flag of entries in the Registry. An additional check is made to ensure the origin matches the current registry so only local data is considered.

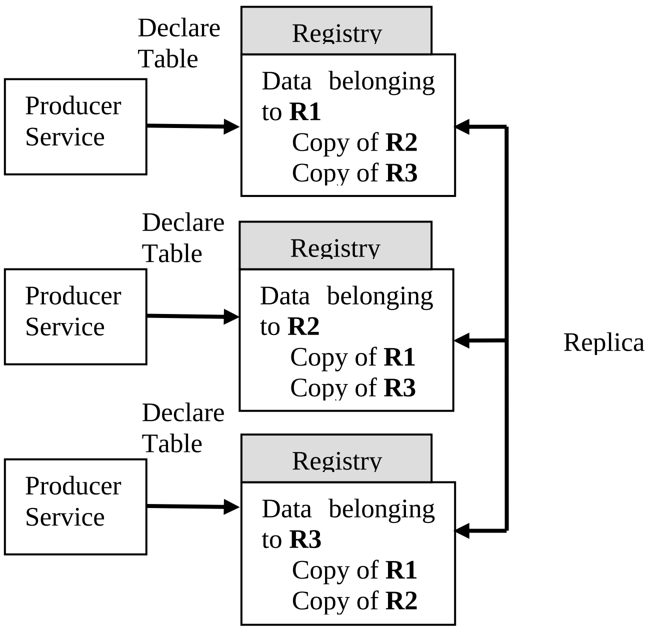

Fig. 2. Only data local to a Registry is replicated

Once a list of candidate data is identified for replication, a replica message is constructed which encodes all the records within an XML format. A checksum is then computed based on the state of the Registry and the final replica message is then sent to each Registry peer. When a replica message is received, the message is passed through a filter that works out in which table each record should be stored. Once each record is inserted into the receiving Registry database, the checksum is re-calculated to validate the consistency of the databases between the sender and recipient. If the checksums match up, a successful response message is returned to the sender. At that point, the replica status flag of all the copied records on the sending registry is set to “replicated” to prevent further replication in future.

If the checksums do not match the replication attempt has failed, indicating that the data stored within the sender's database is inconsistent with that of its peer. In order to resolve this conflict, the original sender will copy all records that match the Registry's origin irrespective of whether the record has been replicated or not. The recipient Registry will then delete all records that match the sender's origin and then perform a clean install using the new replica message. This therefore ensures a high level of consistency is preserved, and that errors cannot go unnoticed.

9.3 Batching Updates

Registry replication updates may be executed in batches, or carried out immediately. Batching helps to reduce the overall load on each Registry participating in the replica configuration but at the expense of some inconsistency between updates. Since services using the Registry are designed to be tolerant of such inconsistencies, a batching approach was implemented meaning that registries will not all be immediately aware of the existence of a new Producer.

10 Schema Replication Design

A replication algorithm for the Schema has also been designed (but not yet implemented). This is a more challenging service to replicate because its data must be consistent with all clients that access the service. The crucial part is ensuring new table definitions and modifications are atomically propagated to all Schema replicas to guarantee that a consistent global view of the R-GMA table definitions is maintained.

Unlike the registry, inconsistency between schema replicas cannot be tolerated. When a schema replica is altered (by a transaction such as table creation and table deletion), R-GMA will use a *two-phase commit* (2PC) protocol to ensure that the transaction is either carried out or aborted by all other replica schemas. In order to prevent write operations from being blocked by an unresponsive replica, some means (either manual or automated) of forcing the transaction to complete on all other replicas is required. This also necessitates a recovery protocol for re-synchronizing the unresponsive replicas when they come back online.

It is vital that all producers contacted by a consumer share the same definitions of any tables mentioned in the consumer’s query. Changes to the schema can have widespread consequences, extending even to user applications. If a table is dropped from a schema, or its definition is altered, it would be almost impossible to propagate the change to all components that depend on that table's definition. To get around this, producers and consumers take their own in-memory copy of a table’s definition, when they first read it from the schema. Thus a consumer will store the definitions of all tables mentioned in its query, and a producer will store the definition of each table to which it is publishing. In this way, producers and consumers become immune to subsequent schema changes.

This means that at any one time, there may be producers and consumers using different versions of a table’s definition. Of course, this only works if the mediator ensures that only producers and consumers using the same version of a table’s definition try to talk to each other. This is achieved by associating a *unique number* with each table definition in the schema. Producers and consumers make a note of the number of the table definition they are using (along with their copy of the definition itself). In addition, the number is added to each producer/table entry in the registry and is also passed to the mediator with every consumer query. In this way, the mediator can ensure that only producers with the correct table definitions are returned for a given query. This number mechanism will be implemented by the R-GMA services, and be hidden from the users.

A new table number is allocated every time a table is created or altered in the schema, and global consistency of the schemas ensures they remain unique.

Changes to table authorization are handled in a similar way to the other write operations on the schema. However, depending upon security requirements, they may need to be propagated rapidly to all producer services, as it is their responsibility to enforce the authorization rules.

11 Deployment of R-GMA

Some applications of R-GMA are described in this section.

11.1 Datagrid

R-GMA was used as an information and monitoring service within the Datagrid project. R-GMA provided information on Computing and Storage Elements (CEs and SEs) to the Resource Broker. This information was used by the Resource Broker to find suitable CEs/SEs for submitted jobs. Network monitoring data was also published using R-GMA [5]. An application that inserted network-monitoring information into the R-GMA virtual database (netmon2R-GMA) was deployed on each network node. A separate network monitoring application was developed to insert GridFTP monitoring information into the R-GMA virtual database (ftlog2R-GMA). A description of the use of R-GMA within DataGrid, along with detailed performance measurements (including indications of the performance gain with replicated registries) has been submitted for publication [11].

11.2 LCG Accounting

The world's largest and most powerful particle accelerator, the Large Hadron Collider (LHC [6]), is being constructed at CERN. The computational requirements of the experiments that will use the LHC are enormous. The LHC Computing Grid [7] (LCG) project aims to meet these computing needs by deploying a grid service composed of computing resources distributed across many different countries. Jobs that run on LCG resources must be properly accounted for so that resources used by VOs and users can be determined. R-GMA is currently being used to implement an accounting service on LCG. The contents of event log files on CEs and SEs are inserted into a MySQL database. The data in this database is then processed to generate accounting records. An R-GMA Producer publishes these accounting records, for retrieval by the Grid Operations Centre.

11.3 SANTA-G

Santa-G (Grid-enabled System Area Networks Trace Analysis) was used to develop a network traffic monitoring application within the CrossGrid [9] project. This NetTracer application produces information that can be used for the validation and calibration of intrusive monitoring systems, and also for analysing the performance of a site in a network. The sensor invokes an open source packet capture application called TCPDump [10], and monitors the log files created by this application. When new logfiles are created, the sensor notifies the QueryEngine, which stores metadata about these logfiles. A LatestProducer then inserts this metadata into the R-GMA virtual database. The QueryEngine allows for queries over the logfiles by using an OnDemand Producer [8]. An SQL SELECT query is sent to the OnDemand Producer, which then forwards this query to the QueryEngine. The QueryEngine returns the appropriate data to satisfy the query.

12 Conclusions

R-GMA provides a global view of information produced by applications distributed over a grid. The Registry and Schema are key components of R-GMA. If these components are unavailable, R-GMA ceases to function. Replication algorithms for both the Registry and Schema have been designed and that for the Registry has been implemented. If one of these components fails at any time, an alternative Registry or Schema is used without clients of the Consumer or Producer being aware. The replication of the Schema and Registry not only eliminates the single point of failure but also improves performance.

13 Acknowledgements

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. A. Cooke et al., *R-GMA: An Information Integration System for Grid Monitoring*, in Proceedings of the Tenth International Conference on Cooperative Information Systems (2003).
2. DataGrid project URL: http://eudatagrid.web.cern.ch/eu-datagrid/
3. 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/gma-wg/papers/gwd-gp-16-3.pdf).
4. Global Grid Forum URL: http://www.ggf.org/
5. F. Bonnassieux, Final Report on Network Infrastructure and Services, deliverable for DataGrid Project, 2003.
6. LHC: Large Hadron Collider Project. URL: http://lhc-new-homepage.web.cern.ch/
7. LCG: 2004, “LHC Computing Grid Project”. http://lcg.web.cern.ch/

8. B. Coghlan et al., The CanonicalProducer: an instrument monitoring component of the Relational Grid Monitoring Architecture, in The 3rd International Symposium on Parallel and Distributed Computing (2004)
9. CrossGrid project URL: http://www.crossgrid.org/
10. TCPdump URL: http://www.tcpdump.org/
11. A.Cooke et al., The Relational Grid Monitoring Architecture: Mediating Information about the Grid, submitted to Journal of Grid Computing