Relational Grid Monitoring Architecture (R-GMA)

Steve Fisher / RAL
4/9/2003

<s.m.fisher@rl.ac.uk>
Who we are

- Heriot-Watt, Edinburgh
  - Andrew Cooke, Werner Nutt
- IBM-UK
  - James Magowan, Paul Taylor
- INFN
  - Roberto Barbera, Giuseppe Save, Gennaro Tortone
- Queen Mary, University of London
  - Roney Cordenonsi
- CCLRC/PPARC
  - Rob Byrom, Laurence Field, Steve Hicks, Manish Soni, Antony Wilson, Jason Leake
  - Linda Cornwall, Abdeslem Djaoui, Steve Fisher, Robin Middleton
- SZTAKI, Hungary
  - Peter Kacsuk, Norbert Podhorszki
- Trinity College Dublin
  - Brian Coghlan, Stuart Kenny, David O’Callaghan
Aim of this talk

• To explain what R-GMA is.
• You will not learn about:
  – How it is deployed
  – What applications it has
  – Performance issues
  – etc.
Features of a grid information system

• Provides information on both:
  – The Grid itself
    • Mainly for the middleware packages
    • The user may query it to understand the status of the Grid
  – Grid applications
    • For users

• Flexible infrastructure
  – Able to cope with nodes in a distributed environment with an unreliable network
  – Dynamic addition and deletion of information producers
  – Security system able to address the access to information at a fine level of granularity
  – Allow new data types to be defined
  – Scaleable
  – Good performance
  – Standards based
GMA

• From GGF
• Very simple model
• Does not define:
  – Data model
  – Data transfer mechanism
  – Registry implementation

Producer

Consumer

Registry

Store location

Lookup location

execute or stream
• Use the GMA from GGF
• A relational implementation
  – Powerful data model and query language
    • All data modelled as tables
    • SQL can express most queries in one expression
• Applied to both information and monitoring
  – All tuples have time stamp
• Creates impression that you have one RDBMS per VO
Relational Data Model

• Not a general distributed RDBMS system, but a way to use the relational model in a distributed environment where global consistency is not important.

• Producers announce: SQL “CREATE TABLE”
  • publish: SQL “INSERT”

• Consumers collect: SQL “SELECT”

• Some Producers and the Registry and Schema make use of RDBMS as appropriate – but what is central is the relational model.
3 Kinds of Query

- **Continuous Queries**
  - StreamProducer
  - Tuple
  - Tuple
  - Tuple

- **History Query**
  - DataBaseProducer
  - Tuple
  - Tuple
  - Tuple
  - Tuple
  - Tuple

- **Latest Queries**
  - LatestProducer
  - Tuple
  - Tuple
  - Tuple
  - Tuple
  - Tuple
Producers

- **StreamProducer** – Supports Continuous Queries
  - In memory data structure
  - Can define minimum retention period
- **ResilientStreamProducer** – Supports Continuous Queries
  - Like the StreamProducer but won’t lose data if system crashes
  - So slightly slower
- **DataBaseProducer** – Supports History Queries
  - Information not lost
  - Supports joins
  - Clean up strategy
- **LatestProducer** – Supports Latest Queries
  - Just holds the latest information for any “primaryish” key
  - Supports joins
- **CanonicalProducer** – Supports anything
  - Offers “anything” as relations
  - User has to write code to handle SQL etc.
Registry and Schema

- Separate components
- Registry has two main tables:
  - Producer
    - Table name
    - Predicate
    - Location
  - Consumer
    - Query
    - Location
- Schema holds description of tables
  - Column names and types of each table
- Registry predicate defines subset of “global” table
- Each implemented with one RDBMS per instance
Contributions to the “global” table

<table>
<thead>
<tr>
<th>Facility</th>
<th>Load</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDF</td>
<td>0.3</td>
<td>19055711022002</td>
</tr>
<tr>
<td>ATLAS</td>
<td>1.6</td>
<td>19055611022002</td>
</tr>
<tr>
<td>ALICE</td>
<td>0.5</td>
<td>19055611022002</td>
</tr>
<tr>
<td>CDF</td>
<td>0.4</td>
<td>19055811022002</td>
</tr>
<tr>
<td>CDF</td>
<td>0.5</td>
<td>19055511022002</td>
</tr>
<tr>
<td>CDF</td>
<td>0.4</td>
<td>19055811022002</td>
</tr>
<tr>
<td>CDF</td>
<td>0.6</td>
<td>19055511022002</td>
</tr>
</tbody>
</table>

WHERE country = 'UK' AND site = 'RAL'

WHERE country = 'CH' AND site = 'CERN'
Mediator

• Queries posed against a virtual data base

• The Mediator must:
  – find the right Producers
  – combine information from them

• Hidden component – but vital to R-GMA

• Will eventually support full distributed queries but for now will only merge information from multiple producers for queries on one table or over multiple tables from one producer
Queries over “global” table – merging streams

SELECT * from CPUload WHERE country = 'UK'

<table>
<thead>
<tr>
<th>Country</th>
<th>Site</th>
<th>Facility</th>
<th>Load</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>RAL</td>
<td>CDF</td>
<td>0.3</td>
<td>19055711022002</td>
</tr>
<tr>
<td>UK</td>
<td>RAL</td>
<td>ATLAS</td>
<td>1.6</td>
<td>19055611022002</td>
</tr>
<tr>
<td>UK</td>
<td>GLA</td>
<td>CDF</td>
<td>0.4</td>
<td>19055811022002</td>
</tr>
<tr>
<td>UK</td>
<td>GLA</td>
<td>ALICE</td>
<td>0.5</td>
<td>19055611022002</td>
</tr>
</tbody>
</table>

Mediator handles merging information from multiple producers for queries on one table.

<table>
<thead>
<tr>
<th>Country</th>
<th>Site</th>
<th>Facility</th>
<th>Load</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>GLA</td>
<td>CDF</td>
<td>0.4</td>
<td>19055811022002</td>
</tr>
<tr>
<td>UK</td>
<td>GLA</td>
<td>ALICE</td>
<td>0.5</td>
<td>19055611022002</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Country</th>
<th>Site</th>
<th>Facility</th>
<th>Load</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH</td>
<td>CERN</td>
<td>ATLAS</td>
<td>1.6</td>
<td>19055611022002</td>
</tr>
<tr>
<td>CH</td>
<td>CERN</td>
<td>CDF</td>
<td>0.6</td>
<td>19055511022002</td>
</tr>
</tbody>
</table>
Queries over “global” table – joining tables

```
SELECT Service.URI, Service.emailContact
FROM Service S, ServiceStatus SS
WHERE (S.URI = SS.URI AND SS.up = 'n')
```
Archiver (Re-publisher)

- It is a combined Consumer-Producer
  - Follows the GMA concept but packaged for ease of use
- You just have to tell it what to collect and it does so on your behalf
- Re-publishes to any kind of “Insertable” (i.e. not to the CanonicalProducer)
  - Can support joins if archiving to a DataBaseProducer or a LatestProducer
Topologies

- Normally publish via a StreamProducer
- Archivers instantiated with a Producer and a Predicate.
  - May re-publish via:
    - StreamProducer
    - LatestProducer
    - DataBaseProducer
- Must avoid cycles in the connections – i.e. must be a DAG.

Note that this structure is required because the DBP cannot stream – very soon it will be able to do so...
R-GMA Tools

• R-GMA Browser
  – Application dynamically generating web pages
  – Supports pre-defined and user-defined queries

• R-GMA CLI (edg-rgma)
  – Command Line Interface (similar to MySQL)
  – Supports single query and interactive modes
  – Can perform simple operations with Consumers, Producers and Archivers

• Pulse
  – R-GMA Java client-based GUI
  – Supports streaming and simple graphical displays
SELECT UniqueID, Name, GlueClusterUniqueID, TotalCPUs, LRMSType
FROM GlueCE
WHERE

Query

Type of query:
- History
- Latest
- Continuous
- Cont.+Old

Queries wait for 5 seconds

Use Mediator
- Select Producers you want to query:

There are no available History producers for table GlueCE

Latest Producer
- producerServlet:http://lxshare0382.cern.ch:8080/R-GMA/LatestProducerServlet ConnectionId:2

Continuous Producer
- producerServlet:http://gpprg06.gridpp.rl.ac.uk:8080/R-GMA/StreamProducerServlet ConnectionId:5
- producerServlet:http://testbed012.cmu.infn.it:8080/R-GMA/StreamProducerServlet ConnectionId:3
edg-rgma

- show tables
- describe Service
- show producers of Service
- latest select * from Service
- old continuous select * from Service
$> \texttt{edg-rgma} \hfill$  
\texttt{rgma>} \texttt{stream declare Service}  
\texttt{rgma>} \texttt{stream minret 0.2}  
\texttt{rgma>} \texttt{stream INSERT into Service (\textit{URI, VO, type, secure, emailContact, site, majorVersion, minorVersion, patchVersion}) values (…}}  
\texttt{rgma>} \texttt{timeout 0.1}  
\texttt{rgma>} \texttt{old continuous select * from Service}  

\begin{verbatim}
+-----+----+-----+--------------+------+--------+--------------+--------------+--------------+-----------------+-----------------+
| URI | VO | type| emailContact | site | secure | majorVersion | minorVersion | patchVersion | MeasurementDate | MeasurementTime |
+-----+----+-----+--------------+------+--------+--------------+--------------+--------------+-----------------+-----------------+
| a   | b  | c   | d            | e    | y      | 1            | 2            | 3            | 2003-07-08      | 10:26:58        |
| a   | b  | c   | d            | e    | y      | 1            | 2            | 3            | 2003-07-08      | 10:26:58        |
+-----+----+-----+--------------+------+--------+--------------+--------------+--------------+-----------------+-----------------+
1 Rows in set
\end{verbatim}
APIs

• Exist in Java, C++, C, Python and Perl
• C, Python and Perl follow an object based style reflecting the Java and C++ APIs

Java
   myProducer = new StreamProducer();

C++
   myProducer= new edg::info::StreamProducer();

C
   myProducer = StreamProducer_new();

Perl
   $myProducer = rgmainfo::StreamProducer_new();

Python
   myProducer = rgmainfo.StreamProducer_new()
Some internals

• Currently based on servlet technology
  – Behind every API there is a Servlet
  – Multiple hand crafted APIs
    • Java, C++, C, Python and Perl
  – Tomcat
  – Soft state registration

• Have recently started the migration to web and grid services
  – Apache axis
  – WSDL generated APIs
  – Providing a wrapper for backwards compatibility
Resilience - Registry

- Will have one logical registry and schema per VO
- Each logical registry has multiple physical “copies”
- Each entry in registry has 3 possible states
- Transmit new records and deleted records and checksum after records deleted locally
- Self healing even supports new registry instances
- Consumer uses any instance
- Fail over mechanism not yet implemented
- Schema more tricky
- Similar to UDDI implementation
Soft-state Registration and the Registry

- Registry records existence of Producers and Consumers
- Registry holds last contact time and ‘expiry’ time
- Producers and Consumers periodically refresh their time stamps
- Producer and Consumer servlets avoid unnecessary traffic to Registry
- Scheduled removal of entries that have timed-out
Security

• So far only have basic authentication
  – Today: grid-proxy-init
  – Soon: voms-proxy-init

• Still have to address the very complex issues of authorisation.
  – Is VOMS enough?
Performance

• By design:
  – Very flexible - to avoid bottlenecks
  – Powerful queries allow a single query to be made

• Performance and Optimisation
  – Are using profiling tools to identify bottlenecks
  – Some problems fixed:
    • Slow Java I/O (now use NIO)
    • Excessive number of threads
    • Slow parsing of SQL
Summary

• R-GMA is a combined Grid information and monitoring system
• Supports notion of Virtual Database
• Now deployed in the EDG production testbed
• Continuing to focus on reliability, stability and performance.
Further Information

• Paper in proceedings

• R-GMA web site
  – http://www.r-gma.org/

• Talk to me:

• E-mail
  – s.m.fisher@rl.ac.uk

Thanks to the EU and our national funding agencies for their support of this work