Time, Information Services and the Grid

Brian Coghlan, Trinity College Dublin Abdeslem Djaoui, RAL Steve Fisher, RAL
James Magowan, IBM & RAL Manfred Oevers, IBM & RAL

May 31, 2001

Abstract

Here we describe work being done to find a good way to provide a distributed Information Service suitable for the Grid. The importance of time in a Grid Information Service is emphasised.

1 Introduction

Grids are inherently distributed systems that tie together data and compute resources[5]. The idea is that users of a grid do not have to worry where a computation takes place or where the data are located. This is taken care of by the Grid middleware analogous to an electrical power grid: A user does not worry where the electricity is produced or how it gets to the socket, she just plugs in her appliance. Grid middleware should do the same for applications.

Data Grids are grid systems that provide special capabilities for providing transparent access to large (tera- to peta-byte sized) data sets.

The European DataGrid project comprises 21 partners from a number of countries around Europe and is part-funded by the EU. It brings together representatives of end-user science communities drawn from particle physics, earth observation and bio-informatics as well as computer science to work on the common goal of building a prototype large scale data-intensive Grid. Similar scale developments are also underway in the US and elsewhere.

In 2005, four new experiments expected to produce of order 10 peta-bytes per year, will start to operate on the new Large Hadron Collider facility at CERN. Thousands of scientists from around the world aim to exploit the physics potential of these data. These "virtual organisations" of scientists will need to muster distributed computing resources via high capacity networks in order to complete analyses of the data. This is where computational data grids are expected to be of great benefit and why they are actively being investigated as a solution.

A key component of the European DataGrid project is the distributed Information Service which provides important data about the state of Grid resources (computing fabrics, networks, storage) as well as monitoring information. This can be used for tasks such as: fault detection, performance analysis and job scheduling. It must be scalable across wide-area networks and be able to integrate a variety of heterogenous resources. A candidate architecture is the Grid Monitoring Architecture (GMA) which has been proposed by the Grid Performance Working Group[10] of the Global Grid Forum[6].

2 The Grid Monitoring Architecture (GMA)

The Grid Monitoring Architecture as shown in Figure 1 consists of three components: consumers, producers and a directory service which we prefer to refer to as a registry as it avoids any implied structure.

Producers register themselves with the registry, which may itself be distributed, 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.

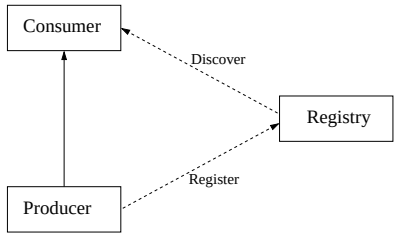

Figure 1: Grid Monitoring Architecture

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 an inter-operable Grid information service. Note that the architecture also supports joint consumer/producer components as illustrated by the component at the centre of Figure 2. This could gather data from several producers and make digested information available to other consumers. It might make use of an RDBMS to support asynchronous access.

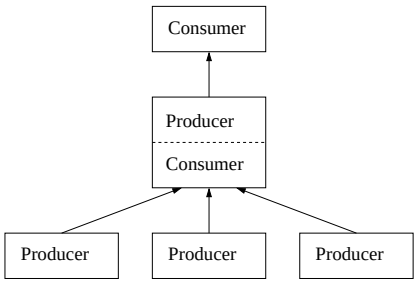

Figure 2: Joint Consumer Producer

3 Time - and some examples

The Global Grid Forum (GGF) has proposed the GMA as a monitoring architecture but we propose to use it for both information and monitoring. We plan to have a common interface to access data, whether it is *fresh* monitoring data or data from an archive. It is preferable to associate all information with a time stamp - then if there are any delays in delivering information there is no ambiguity. If information is archived for the purpose of making a model to predict future behaviour, then clearly all the information must carry a time stamp. Consider three uses of a Grid information system.

Case 1: A scheduler needs information on availability of usable resources worldwide. The scheduler cannot afford to wait if information is slow in arriving. It does not need to find the best solution, but merely one near the optimum. Some information will be out of date, but as Coghlan[1] has explained we should expect that. There is a need to consider queries with a time-out - this is both from the point of view of expressing the query and implementing it - see Plale and Dinda[3]. Jobs will of course run in the future, not when they are submitted, so predictions are important. This implies accumulating old data and having a model. Time stamps are vital on all the information if such a model is to be built.

Case 2: A computing fabric needs to have all recent information held reliably for post mortem analysis when a part of the system dies. Again the time stamp is vital, in this case for reconstruction of the sequence of events leading to the failure. A reliable archival mechanism is also needed. In carrying out the analysis it is undesirable to lock the state of the fabric or to stop collecting logging information. One would typically start by looking at events occurring just before the failure.

Case 3: For application monitoring there is a need to correlate fabric information with information produced by a job. The results may be used by sophisticated jobs to optimize their own behaviour according to their environment, i.e. the conditions on the fabric. Time may be used as an index to the sequence of events, and as an input to application-level models of the fabric and job. The event sequence is also useful for application-level debugging.

Bearing in mind the importance of time, DataGrid is currently investigating how best to realise the architecture by considering possible contributions from LDAP and from a relational approach.

4 LDAP

LDAP as described in RFC 1588 - has a defined wire protocol and is well suited for distributed systems but has the problems (and strengths) of a tree structure. LDAP, in common with other hierarchical structures is fine *if you know all the queries in advance* as you can build your database to answer those questions very rapidly. Unfortunately, if you fail to anticipate the question, getting an answer could be very expensive. The LDAP query language cannot give results derived from computation on two different objects in the structure – or, expressed in relational language, there is no *join* operation.

LDAP is of interest to us because it is a fundamental technology used by the Information System of the Globus toolkit[7] which is the basis of the first test-beds from the DataGrid. A recent paper[2] explains the current Globus developers thoughts on this, including possible use of relational components.

5 Relational Approach

The relational model appeared at the last GGF with two papers; one by Dinda and Plale[3] in which they explain the benefits of the relational model. They advocate a single RDBMS to hold all the information, and thereby probably avoid the need to register producers. A second paper by Fisher[4] explains how the relational model might be used with the GMA performance architecture[10] to ensure good scaling, reliability and performance. Note that this latter paper does *not* propose a general distributed RDBMS system, but a way to use the relational model in a distributed environment where determinism is abandoned and ACID properties are not emphasised. We are currently investigating such an approach.

The API we are considering does not expose the registry. A producer simply has to announce a table name and the row(s) of a table. Behind the scenes the producer will communicate with a servlet, which will register the table name and the identity and value of any fixed attributes. Consumers can issue SQL queries against a set of supported tables (the names and attributes of which can be consulted). Behind the consumer API the query goes to a servlet which analyses the query and makes use of the registry to find suitable producers of information. In the general case queries will be sent to different producers and the servlet acting on behalf of the consumer will process the results. Queries which can be processed by a single producer can be handled efficiently, but others will result in some operations being carried out by the consumer servlet as indicated above. This suggests that there will be *advantages in having Producer/Consumer/RDBMS units able to hold data which will often be joined*. In fact such a unit might be created automatically and then destroyed when it is no longer frequently used.

The schema information, i.e. the table descriptions, must be universally known. This is a problem for application monitoring data where the schema could be very short lived. One solution is to ensure that the registration of new tables is easy to do. A RDBMS can hold both the schema and an associated list of the available producers. In this case ACID properties are essential. This would need to be replicated for scalability and reliability, but it must done in such a way as to allow producers around the world to add new tables. The producers will periodically re-announce themselves, but they will be dropped from the list of producers if they do not do so within a defined interval. When a producer registers itself as a producer of a certain table, if the table is not known it can be added to the schema. If a table is not available from any producer then its definition can be removed. This is convenient for schema evolution.

It is also necessary for a consumer to be able to register with a producer to either receive alarms or a stream of tabular data.

There is also a need to give a level of autonomy to sites and to avoid the registry getting too large so we envision a system as indicated in Figure 3 which shows a global registry and a site registry.

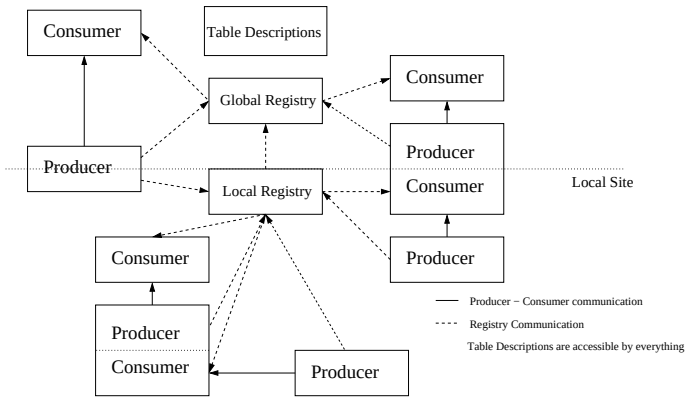

Figure 3: Multiple Registries

6 Work so far

We have investigated the use of a servlet engine to offer an SQL service via JDBC to MySQL[9]. Communication between consumer and producer is carried out via XML over http(s). We are finding it helpful to split the system up into a number of rather simple servlets. An API has been sketched out and we have an SQL parser which has the capability of understanding the SQL. We are about to consider the harder task of query splitting.

Coding has been in Java, making use of Ant, JUnit and Tomcat from Jakarta/Apache[8].

References

[1] Brian Coghlan. A case for relational GIS/GMA using relaxed consistency. Technical report, GGF, 2001.

[2] Karl Czajkowski, Steven Fitzgerald, Ian Foster, and Carl Kesselman. Grid information services for distributed resource sharing. Draft, 2001.

[3] Peter Dinda and Beth Plale. A unified relational approach to grid information services. Technical Report GWD-GIS-012-1, GGF, 2001.

[4] Steve Fisher. Relational model for information and monitoring. Technical report, GGF, 2001.

[5] I. Foster and C. Kesselman, editors. *The Grid - Blueprint for a new Computing Infrastructure*. Morgan Kaufmann, Nov 1998.

[6] Global grid forum. http://www.gridforum.org/.

[7] Globus. http://www.globus.org.

[8] Jakarta from apache. http://jakarta.apache.org.

[9] Mysql. http://www.mysql.com.

[10] Brian Tierney, Rich Wolski, Ruth Aydt, Martin Swany, Valerie Taylor, and Dan Gunter. A grid monitoring service architecture. Technical report, GGF, 2001.