Infogrid - a Relational Approach to Grid Computing

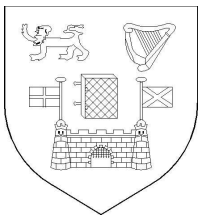
by

Oliver Lyttleton

A Thesis submitted to
The University of Dublin
for the degree of

Doctor of Philosophy

Department of Computer Science,
University of Dublin,
Trinity College



April, 2008

Declaration

This thesis has not been submitted as an exercise for a degree at any other University. Except where otherwise stated, the work described herein has been carried out by the author alone. This thesis may be borrowed or copied upon request with the permission of the Librarian, University of Dublin, Trinity College. The copyright belongs jointly to the University of Dublin and Oliver Lyttleton.

Signature of Author .....

Oliver Lyttleton April, 2008

Summary

Grid middleware is composed of a variety of components which perform the functions that a Grid infrastructure requires. Some generic functions are common to several components. These functions include data storage and retrieval, security, and remote process invocation. There is a degree of redundancy in existing Grid middleware as these functions are implemented using various applications. This thesis will explore how a single database technology can be used to implement these functions, resulting in a less complex Grid middleware that is implemented using fewer applications than at present.

This thesis describes the implementation of a prototype Grid middleware, called Infogrid, that uses a single database technology to perform generic services mentioned previously (data storage and retrieval, security, and remote process invocation) that are implemented using a variety of applications in existing middleware. In addition, for small datasets and debugging, Infogrid uses this technology to implement a relational interface for programs executing on the Grid that allows input data to be type checked as it is submitted to the Grid for processing.

Acknowledgements

I'd like to thank my supervisor Dr. Brian Coghlan for providing guidance, insight and the benefit of his enormous breadth and depth of experience. My colleagues in the Computer Architecture and Grid Research group were invaluable allies and friends in my path towards completing my Ph.D, they gave me the benefit of their knowledge and skills whenever asked, and without the help of the people around in the research group, my journey would have been much more difficult. In particular, Dr. Eamonn Kenny provided crucial support whenever asked, and I am in his debt for the assistance he provided. I am grateful for my friends outside the world of Grid computing and research, they provided balance and perspective when it was most needed. I would like to my family for being a constant source of support during my Ph.D. Mum, Dad, James, Gemma, Tomas, you all provided support in your individual styles.

Oliver Lyttleton

Contents

List of Figures

List of Tables

List of Acronyms

AliEn	ALICE Environment
APEL	Accounting Processor for Event Logs
API	Application Programming Interface
BDII	Berkeley Database Information Index
BLAST	Basic Local Alignment Search Tool
BLOB	Binary Large Object
CA	Certification Authority
CE	Computing Element
CMS	Compact Muon Solenoid
CPU	Central Processing Unit
DEISA	Distributed European Infrastructure for Supercomputing
Applications	
DML	Data Manipulation Language
DN	Distinguished Name
EDG	European DataGrid; the project and its software
EGEE	Enabling Grids for E-science
EUGridPMA	European Policy Management Authority for Grid Authentication in e-Science

FDM/RC	Functional Data Model with Relational Covers
G-DSE	Grid Data Source Engine
GGF	Global Grid Forum; now *OGF*
GIIS	Grid Index Information Service
GIN	Gadget IN
GRAM	Globus Resource Allocation Manager
GridPP	Grid Particle Physics
GRIS	Grid Resource Information Service
GSI	Grid Security Infrastructure
IVDGL	International Virtual Data Grid Laboratory
JDL	Job Description Language
L&B	Logging and Bookkeeping
LCAS	Local Centre Authorisation Service
LCG	*LHC* Computing Grid
LCMAPS	Local Credential Mapping Service
LDAP	Lightweight Directory Access Protocol
LFC	LCG File Catalog
LHC	Large Hadron Collider
LRMS	Local Resource Management System
LSF	Load Sharing Facility
MDS	Monitoring and Discovery Service
NAREGI	National Research Grid Initiative
OGF	Open Grid Forum; formerly *GGF*

OGSA	Open Grid Service Architecture
OGSA-DAI	*OGSA* Data Access and Integration
OGSA	Open Grid Service Infrastructure
PBS	Portable Batch System
PKI	Public Key Infrastructure
PPDG	Particle Physics Data Grid
R-GMA	Relational Grid Monitoring Architecture
RB	Resource Broker
RDBMS	Relational Database Management System
RLS	Replica Location Service
RSL	Resource Specification Language
SQL	Structured Query Language
UDF	User Defined Function
UI	User Interface
URL	Uniform Resource Locator
VDT	Virtual Data Toolkit
VO	Virtual Organisation
VOMS	Virtual Organisation Membership Service
WMS	Workload Management System
WN	Worker Node
X.509	an ITU-T standard for *PKI*
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 What is the Grid?

The term "Grid computing" has a variety of definitions. Broadly speaking, Grid computing is an architecture where geographically dispersed computation and data storage facilities, which are not under centralised control, act in concert to perform computing tasks. Grids are generally used to enable sharing of large scale aggregations of resources, which form a "virtual computer". Users can submit "jobs" to the Grid. A Grid job consists of an executable that is to be run on this infrastructure, along with any input files and command line arguments required by the executable.

Heinz Stockinger's paper [8] presents the results of a survey that aims to clarify what a Grid is. This survey was one of the first attempts to present a view on what the Grid research community thinks defines a Grid. Over 170 researchers from around the world were queried in an attempt to:

> *Try to define what are the important aspects that build a Grid, what is distinctive, and where are the borders to distributed computing, Internet computing, etc.*

Stockinger states that despite advances and developments in the field of Grid computing over the years, there were a number of dominant characteristics of Grid systems

that were generally agreed upon by the Grid researchers who took part in the survey. These characteristics were:

- Collaboration: a Grid spans multiple administrative domains.
- Aggregation: a Grid aggregates many resources.
- Virtualization: Grid software provides an interface that hides the complexity of the underlying resources.
- Service Orientation: Grids are implemented using a Service Oriented Architecture.
- Heterogeneity: a Grid is composed of a variety of hardware and software.
- Decentralised Control: components of the Grid are not centrally controlled.
- Standardisation: a Grid uses standardised interfaces.
- Access Transparency: users of the Grid are not required to be aware of the underlying architecture or network topology.
- Scalability: a Grid enables users to solve problems of a scale that could not be solved using a single desktop machine.
- Reconfigurability: a Grid should be dynamically reconfigurable.
- Security: a Grid provides secure access to resources.

Grid middleware, the software that provides the services required by a Grid infrastructure, is highly complex. Grid middleware is composed of a variety of components which are used to perform functions such as submitting jobs to the Grid, granting or denying access to resources on the Grid, maintaining up to date information about the current state of Grid resources, storing logging information about events that have occurred on the Grid, and many other functions. Some generic functions are required by multiple Grid middleware components:

- Given the distributed nature of a Grid infrastructure, Grid middleware components are required to send and accept remote procedure calls (RPCs) in order to interact with other components.

- There are also Grid middleware components whose primary function is to enable access to various types of data, such as the current state of resources on the Grid, or logging information generated by Grid components. Mechanisms for storing and retrieving data are required by these components.

- Security mechanisms are required by Grid middleware components, in order that their use is restricted to users whose identity has been authenticated and who have been authorised to use the Grid middleware.

Each of these functions is implemented using multiple technologies in existing Grid middlewares. Several database applications are used by some middlewares for storing and retrieving data. Multiple implementations of RPC mechanisms are used to enable the functionality of some Grid middleware components to be invoked remotely.

This thesis will investigate how the complexity of the implementation of Grid middleware can be reduced by using a single technology to provide these functions. Several different types of technologies were considered for the implementation of these services. There are a number of RPC or message passing mechanisms (such as web services [9], the Generic Security Standard Application Programming Interface (GSS-API) [10] or the Java Message Service [11]) that can be used by Grid middleware components to interact with each other. These message passing mechanisms also provide security mechanisms, however they do not provide the advanced data storage and retrieval facilities that a Grid requires. There are also APIs available which provide access to security mechanisms which could be used by various Grid middleware components, for example the Java Security API [12] or the BouncyCastle security API [13], but these security APIs do not provide the RPC or data storage and retrieval functionalities required by Grid middleware.

Database technology however could provide all the generic functions outlined above. The primary function of databases is to enable the storage and retrieval of

data. Databases could be used to store and retrieve data distributed across a Grid. They also provide security mechanisms which can be used to perform authentication and authorisation operations. Databases can provide a federated view of data that is distributed across a network of computers (such as a Grid), and clients can issue requests for data that is stored on a remote machine. Databases could therefore be used to enable an RPC mechanism which could enable Grid middleware components distributed across a Grid to interact with each other. Implementing a Grid middleware using a single database technology to implement all these generic services will result in a less complex implementation of the Grid middleware.

Relational technology can also be used to implement a user interface to the Grid. Users can submit data to the Grid for processing by inserting rows into particular tables in the database. For small datasets the data may be inserted directly, otherwise indirect references to the data must be inserted. This relational interface allows input data for applications to be type checked at the time it is submitted to the Grid for processing. Current Grid middleware specifies input data for Grid jobs as files or command line arguments. However, type-checking on input data is not performed when the job is submitted. If input that is specified for a job does not conform to what the executable requires, a runtime error may occur when the job is executed on the Grid. This can happen if inputs with incorrect datatypes are specified, or the wrong number of inputs are provided. A period of time elapses as the Grid middleware assigns a job to a resource on the Grid, transfers the job to that resource, and the resource becomes available to execute the job. The user is not informed of the error immediately. The user must query the status of the job using tools provided by the Grid middleware to discover that the job has failed. In contrast, implementing a relational user interface for applications that are to be executed on the Grid allows type checking to be performed at the time the job is submitted, and for the user to be immediately informed of errors caused by using incorrect datatypes or specifying an incorrect number of input arguments.

This thesis describes a prototype Grid middleware called Infogrid which uses a single database technology to implement generic services that are currently realised

using multiple technologies in existing Grid middleware, resulting in a simpler implementation of a Grid middleware. Infogrid also provides a relational interface for submitting data for processing on the Grid which enables type checking to be performed at the time that data is submitted.

1.2 Structure of Thesis

The structure of this thesis is outlined as follows. Chapter 2 provides an overview of Grid computing, describing various research projects related to Grid computing, and current applications of Grid computing. How users interact with the Grid is described in detail, and other research projects which integrate Grid computing and relational database technologies are reviewed. Chapter 3 outlines the hypothesis of the thesis and describes the architecture for the Infogrid prototype. The components of the Infogrid middleware, and the interactions between these components, are described independently of the technology chosen to implement the Infogrid middleware. Chapter 4 details the implementation of the Infogrid prototype, while Chapter 5 presents experimental results obtained from the deployment of this prototype. Chapter 6 presents conclusions.

Chapter 2

Computational Grids

2.1 Introduction

Infogrid will implement the functionality provided by multiple components of Grid middleware using a single database technology. This chapter describes the concept of Grid computing in more detail, and an abstract model of a Grid middleware illustrating the main components common to the various types of middleware is presented.

Various forms of research that are taking place in the field of Grid computing are described. These research projects involve the construction of Grid infrastructures, the analysis of data from scientific experiments using these infrastructures, and the development of software that manages Grid infrastructures. The software that manages Grids, Grid middleware, is of particular relevance to this thesis. In order to illustrate how Infogrid could be implemented using any one of a number of Grid middlewares, similarities between the various middlewares in terms of the functions performed by their components are highlighted. In addition, for each middleware, particular attention is given to:

- the different technologies that are used by Grid middleware components that perform the various services that are required to implement a Grid.
- how jobs are submitted, and how input for jobs is specified. Shortcomings in type checking for Grid jobs will be highlighted later in this chapter.

- the various Application Programming Interfaces (APIs) that are available for invoking commands on Grid middleware components that could be replaced by a single API.

Research projects which integrate Grid and database technology are examined at the end of this chapter. Particular attention is given to the GridDB and XG projects, which investigate how providing a relational interface to the Grid can provide type checking at the time of job submission. This chapter will also illustrate the relationship between GridDB, XG and Infogrid by highlighting how Infogrid builds on GridDB and XG by addressing issues that were neglected in these projects and using database technology in Grid middleware to implement services other than job submission.

2.2 Grid computing in brief

Grid computing provides an aggregation of computing and storage facilities. Grid middleware monitors the state of these resources, gathering data on their current load and availability, and uses this information when deciding where on the Grid to execute jobs. A Grid computing infrastructure is similar in functionality to a cluster computing infrastructure. A cluster is a pool of computers that are co-ordinated by software in such a fashion that they can be used as if they were a single system. The distinction between a Grid architecture and a cluster architecture is that the resources that compose a cluster are under centralised control within a single administrative domain, whereas the resources that compose a Grid are not under centralised control, and can be located in multiple administrative domains.

Figure 2-1 illustrates some of the main components common to various Grid middlewares. These include:

- User interfaces through which jobs are submitted.
- Logging systems which store details of events occurring during the life cycle of Grid jobs.

- Information systems which monitor the current state of resources on the Grid.
- Schedulers which match jobs to resources on the Grid capable of executing those jobs.
- Gatekeepers which restrict access to Local Resource Management Systems (LRMS), which execute jobs, to authenticated and authorised users. Typically the LRMS is the front end to a cluster.

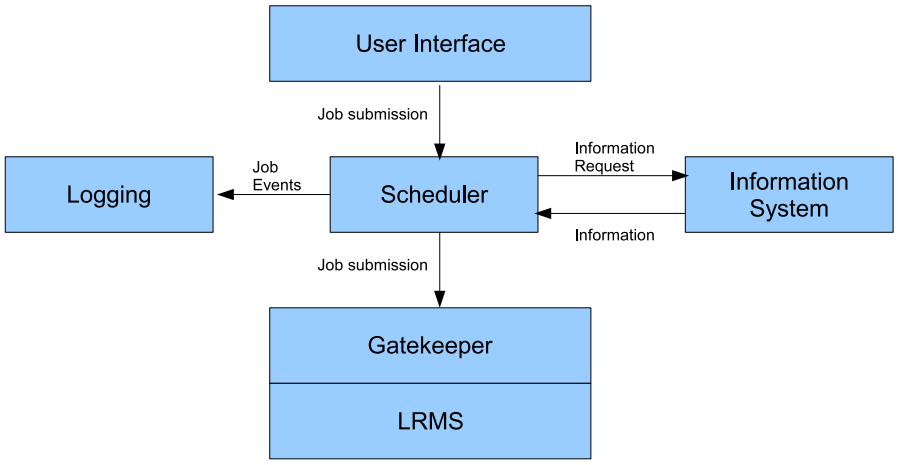


Figure 2-1: Main components of Grid middleware.

The sequence of events that occur when a job is submitted to a Grid is similar for all middlewares. The job is submitted from a user interface, and sent to a scheduler application, which determines what is the optimal resource on the Grid for executing this job. The scheduler application then submits the job to the selected remote resource for execution. A gatekeeper restricts access to Grid resources to users who have been authenticated and are authorised to use the Grid. The LRMS executes the

job on a set of resources, and when the job has successfully completed, the output is retrieved by the user.

There are many Grid computing-related research activities. Some of these research projects use the middleware to implement, test and develop computational Grids. Several examples of these projects that have implemented Grid infrastructures are PPDG [14], iVDGL [15], TeraGrid [16], Deisa [17], NAREGI [18], DGrid [19], GridPP [20], Grid-Ireland [21] and EGEE [22]. The other main types of Grid research projects are those which run experiments and applications on Grid infrastructures, and those which create Grid middleware that implement and manage these Grid infrastructures.

2.3 Grid Experiments and Applications

As mentioned in the previous section, many Grid research projects neither create Grid software, nor deploy it, but execute applications on existing Grid infrastructures. This section will give examples of applications that are currently run on Grids, which users of Infogrid would ideally also be capable of executing by inserting rows into tables in a database that represent an interface to the Grid. One such example is the use of Grid technology to process data generated by high energy physics experiments. Large amounts of computational and storage resources are required to process the data from these experiments, which can accumulate to petabytes over the course of a year. Data produced by physics experiments run on the Large Hadron Collider (LHC) in the European Organization for Nuclear Research (commonly called CERN) will be processed by a Grid computing infrastructure implemented using the gLite middleware[23]. Grid technology is also used to analyse data produced by the Babar High Energy Physics experiment [24].

Genomics research is another field which uses Grid computing to process large amounts of data. GridBLAST [25], G-BLAST [26] and BGBlast [27] are three of the many projects which use Grids to perform gene sequence alignment on genomic datasets using the Basic Local Alignment Search Tool (BLAST) [28]. Grid computing infrastructures are also used in the field of astronomy to process sky surveys. The

GRIST [29] project aims to establish a service framework that complies with Grid and web services standards that can be used to perform astronomical image processing and data mining on the Grid. As an example of an application carried out as part of the GRIST project, data obtained from the Palomar-Quest survey [30] representing the readings obtained by a radio telescope from a section of the sky are used as input to applications executed on a Grid which calculate a probability score that the data represents a quasar or some other transient optical effect that may be of interest to astronomers. Processing readings for the entire sky, over a lengthy period of time, can require a lot of computational power. The GRIST project allows these calculations to be performed in parallel across a Grid infrastructure. Grids are also used to execute applications in a diverse range of research fields, such as medicine, finance, seismology, climate studies, and many more.

2.4 Grid Middleware Projects

There are several research projects that have developed Grid middleware. These projects have created and maintained the software that is used to implement Grid infrastructures. Three projects which have produced widely used Grid middleware are Condor [31], the Globus Toolkit [32], and gLite [23]. These middlewares all provide broadly the same set of components presented in our abstract model of a Grid middleware at the beginning of this chapter. The multiple technologies used by these middlewares to provide the services required by a Grid will be illustrated. This will demonstrate the degree to which Infogrid, which will use a single database technology to provide some of these services, could be a simpler Grid middleware than those that exist at present. Examples of how jobs are described by each of these middlewares are provided, and the lack of data typing provided by these job description formats highlighted.

2.4.1 Condor

Condor is a cluster computing tool developed by the University of Wisconsin. Condor also allows clusters which are in separate administrative domains to be aggregated using a feature called flocking. Condor provides mechanisms for job submission, scheduling jobs on resources, monitoring the state of resources, and managing those resources. Condor has several unique features, such as harvesting CPU cycles from idle processors, and the ability to checkpoint and migrate jobs midway through execution to alternative resources. The main components of the Condor system are shown in Figure 2-2 (which is based on a diagram taken from the Condor instruction manual).

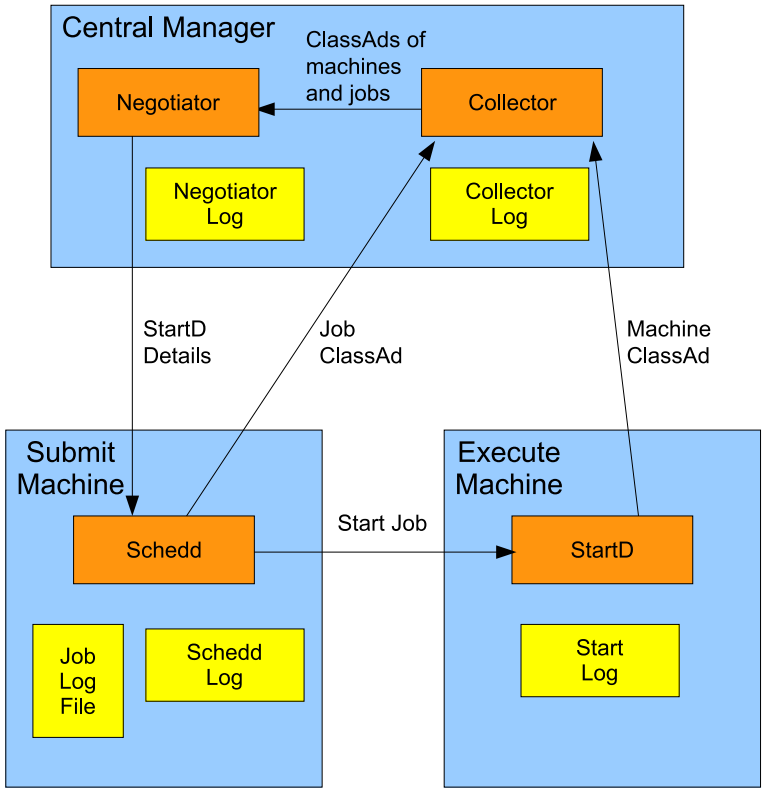


Figure 2-2: Main components of Condor (from Condor instruction manual [1]).

Condor matches jobs to computing resources using the ClassAds [33] mechanism. ClassAds are a format for describing the hardware and software resources available on a particular machine, and the hardware and software requirements for a job. Machines in a Condor cluster can have three roles: a Central Manager, a Submit

Machine, and an Execute Machine.

- Central Manager: There are two components on this machine which are of particular importance. The Collector process on the Central Manager retrieves ClassAds describing the current state of all the resources available on the Cluster (Figure 2.2 shows the Collector on the Central Manager receiving a "Machine ClassAd" from the Execute Machine) and obtains details of jobs submitted to the Condor cluster. The Negotiator component matches the jobs to resources which are suitable for executing them.

- Submit Machine: This machine contains a process, Schedd, that submits jobs to the Central Manager. Figure 2.2 shows a Schedd process sending a ClassAd describing a submitted job to the Central Manager ("Job ClassAd").

- Execute Machine: Jobs are executed on this machine. The StartD Process initiates job execution on the machine.

Logging information related to the execution of jobs is stored on each of these machines. The Negotiator, Collector, Schedd, and StartD components all store information on events related to their operation to logs on the same machine they are running on (as illustrated in Figure 2.2). Information related to events in the lifecycle of jobs is also written to a log file for job events on the Submit machine. There is no centralised logging mechanism for Condor, which precludes logging information located on machines distributed across the cluster to be accessed via a single mechanism. In addition, the Information System, which maintains an up-to-date view of resources on the cluster, is implemented by the Collector, and is separate from the logging information. Condor does not provide a single access mechanism for obtaining information on the current state of cluster resources, and logging information concerning past events which have occurred during the operation of the various components of Condor. It is an aim of Infogrid that a single database technology will be used to store these different types of information.

Users submit jobs to Condor by describing them in a job description file, and using a command line executable (called condor_submit) to submit that file to the

Schedd process on a machine designated as a Submit machine. The condor_submit executable converts the job description into the ClassAd format before sending it to the Schedd process. An example of such a job description file is shown below:

```
Universe = vanilla
Executable = simple
Input = test.data
Arguments = 4 10
Log = simple.log
Output = simple.out
Error = simple.error
Queue
```

This job description does not specify data types for the "Arguments" attribute, nor does it specify the structure or types of the data contained in the input file test.data. It is an aim of Infogrid to provide a job submission interface using the same database technology that implements logging and information systems that would allow type checking to be performed on data as it is submitted for processing to the Grid.

2.4.2 Globus toolkit

The Globus toolkit is a set of applications that can be used to provide services required by Grid infrastructures. It is developed by a number of research institutes and universities called the Globus Alliance. The Globus toolkit includes applications such as the Monitoring and Discovery System (MDS) [34], which monitors resources on the Grid, data management tools (Replica Location Service (RLS) [35], GridFTP [36]), software for executing processes on remote resources (Grid Resource Allocation Management (GRAM) [37]), and security tools (MyProxy [38], SimpleCA [39]).

Jobs are submitted to a Grid implemented using the Globus Toolkit by describing

them in a language called the Resource Specification Language (RSL) [40], and using a command line executable to submit this RSL file to a GRAM gatekeeper which executes that job on a remote resource. An example of an RSL file is shown below.

```
& (executable = generateData )
(directory = /home/user1 )
(stdin = input.dat)
(arguments = 100 )
(count = 1)
```

This is a simple job description file, which uses attributes similar to those used in the Condor job description file outlined in the previous section to describe a job. As with the Condor job description, this job description does not allow the types of input data to be specified for command line arguments (defined by the "arguments" attribute) and input files (defined by the "stdin" attribute).

Figure 2-3, which is based on a diagram taken from a paper describing the Globus resource management architecture ([2]), illustrates how various components provided by the Globus Toolkit are deployed to create a Grid site. In Figure 2-3, a Grid Scheduler uses the MDS API to query the MDS information system in order to find a suitable resource for executing a job. Each site also uses the MDS API to provide MDS with information on the current state of resources on that site. Having found such a resource, the Grid Scheduler uses the GRAM API to submit the job to the Gatekeeper on a Grid site, which uses the Globus Security Infrastructure to restrict access to resources on that site to authenticated users. The Gatekeeper will launch a job manager process to submit a job to the LRMS on that Grid site. The Gatekeeper writes information on events that occur as the job submission and authentication events take place to the Globus gatekeeper log, and the LRMS writes logging information concerning events that take place as it executes jobs to the LRMS

log.

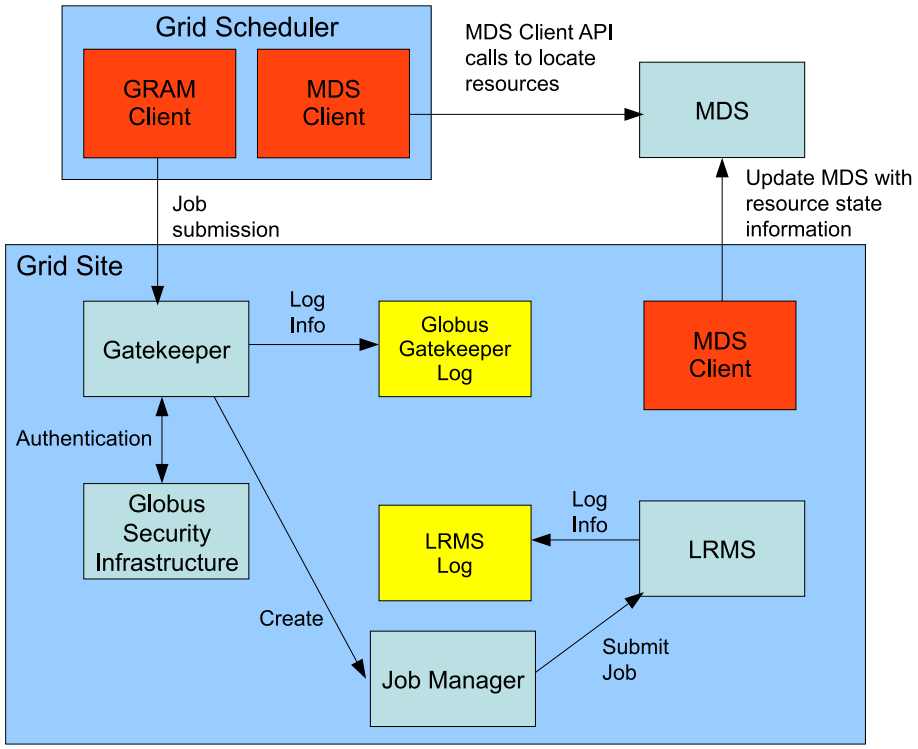


Figure 2-3: A Grid site implemented using components from the Globus Toolkit (based on diagram from [2]).

The Globus Toolkit provides multiple APIs used to perform several functions (e.g. submitting a job to the Gatekeeper, querying the MDS information system, implementing security). It is clear that Infogrid would be a simpler implementation of a Grid middleware by providing access to these functions via a single database API. As with Condor, logging information is stored in text files on the Grid site, and there is no single mechanism for centrally accessing logging information distributed across several Grid sites. In a Grid implemented using the Globus toolkit, as is also the case with Condor, the logging system and the information system are implemented as separate components. A single database technology could also be used to implement these components.

2.4.3 gLite

One of the largest Grid middleware projects is gLite [23], which is being developed by a number of organisations collaboratively as part of the Enabling Grids for E-Science in Europe (EGEE) [22] project. gLite makes use of software developed in other Grid middleware projects, including Condor and the Globus toolkit, and also draws on work from the Large Hadron Collider Computing Grid (LCG [41]), the Virtual Data Toolkit (VDT) [42], the Global Grid Forum (GGF) [43], the Open Grid Services Architecture (OGSA) [44], and the Alice Environment (AliEN) [45], amongst others. The gLite middleware is composed of a number of components that provide services required by Grids. These services can be broadly categorised under 5 different groups:

- Access services: Access services allow users to submit jobs to the Grid for processing, query the status of jobs running on the Grid, retrieve the output of jobs, etc. These operations can be performed using command line executables or APIs.

- Security: Security services restrict access to resources on the Grid by establishing the identity of the user that is trying to use those resources, and establishing if that user is authorised to do so.

- Data Management: Data management services enable access to data distributed across the Grid. Location transparency is provided by the LCG File Catalog (LFC) [46], which maintains a lookup table that maps Logical File Names (LFNs) to the physical location of those files (called Storage URLs). LFNs are independent of the physical location of the data, therefore allow data to be referenced without specifying its location on the Grid. The Storage Element (SE) is a virtualisation of a storage resource (such as large disk arrays or mass storage systems) which provide disk space for jobs that execute on the Grid. It provides a single interface to a number of different storage mechanisms. The gLite File Transfer Service allows files to be moved from one location on the Grid to another.

- Workload Management: Workload management services are responsible for allocating jobs to resources on the Grid. There are two main workload management services in the gLite middleware, the Workload Management System (WMS) and the Computing Element (CE). The WMS is responsible for deciding which site on a Grid to submit a job to, while the CE is responsible for scheduling jobs at a site level. The CE implements local security policies for sites hosting resources which are part of a Grid. Both the WMS and the CE expose interfaces that allow users to submit and manage jobs.

- Information and Monitoring: Information and Monitoring services enable the publication of information describing the current state of resources on the Grid to a database, and allow this data to be consumed by applications or users which need this information, such as workload management services which must decide where on the Grid to execute a job. The Berkeley Database Information Index (BDII) Server [47] used by the gLite middleware provides a centralised access point for information on resources distributed across a Grid.

Figure 2-4 illustrates the interaction between components of the gLite Grid middleware. The User Interface (UI) is a machine which hosts the command line tools and APIs that provide access services for the Grid. In Figure 2-4, the gLite WMS API is used to submit a job to the WMS. The WMS interacts with other middleware components using various APIs. The WMS uses the BDII API to query the BDII server to obtain information on Grid resources in order to match the job to a resource that fulfills the criteria specified by the user (e.g. available memory, CPU, etc.). When the WMS has determined on what Grid resource a job is to be executed, it uses CondorG (an implementation of the GRAM API) to submit a job to the CE which enables remote access to that resource. The Globus Gatekeeper is used by the CE to authenticate and authorise job submissions. The LRMS is the local cluster computing system which performs the execution of the job. The CE uses the Grid File Access Library (GFAL) API to query the LFC to locate files required by jobs, and to transfer input files for jobs from the SE to the CE. Throughout the lifecycle of a job, the WMS records details of middleware events related to the processing of

the job using the Logging and Bookkeeping (L&B) Server API.

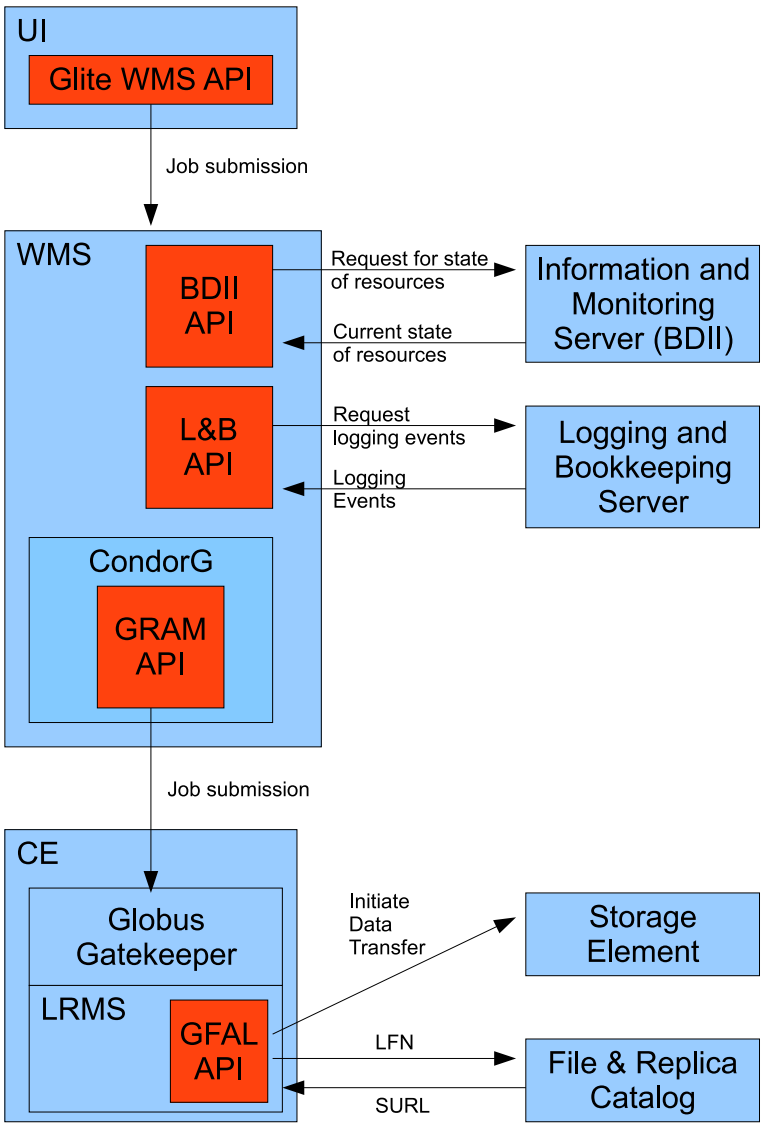


Figure 2-4: Components of the gLite middleware.

Jobs that are submitted to the gLite middleware are described in Job Description Language (JDL) files. The following is an example of a JDL file which describes a gLite job using a set of attribute-value pairs.

```
Executable = "/bin/analyseData";
Arguments = "2 4";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = "input.dat";
OutputSandbox = {"std.out", "std.err"};
Requirements = other.GlueCEPolicyRunningJobs = 0;
Rank = other.GlueCEStateEstimatedResponseTime;
```

As with the previous examples of job description files used by Grid middlewares, this job description file does not specify the datatypes of the command line arguments used by the job (defined by the "Arguments" attribute) and the input file used by the job (defined by the "InputSandbox" attribute).

2.5 How do users submit jobs to the Grid

A Grid can provide a substantial amount of computational resources to users. The user does not need to know the physical location of these resources, nor the technologies used to implement them. The Grid middleware hides the details involved in matching a job to a resource, transferring data to that resource, and initiating the execution of the job on that resource from users. However, in order to submit a job to the Grid, it is still necessary to prepare a file describing the job. As seen in previous sections, there are a variety of job description languages used by the various Grid middlewares. The gLite software uses JDL to describe jobs, Condor has a job description format based on ClassAds, and Globus gatekeepers accept job descriptions specified in RSL. There has also been research into a universal job description language called the Job Submission Description Language (JSDL) [48]. Broadly speaking, a job description file can (but is not required to) specify:

- what program will be executed on the Grid.
- command line arguments for the program executable.
- input environment (input files required by the job and environment variables).
- output file (what the name of the file containing the output from the job will be).
- hardware and/or software requirements for the job.

Command line executables and APIs can be used to submit jobs to a Grid. These can also be used to perform other operations, such as determining the status of a job (e.g. is the job running, aborted, finished, cancelled), cancelling a job, or retrieving the output of a job once its execution has successfully terminated. Figure 2-5 shows a screenshot of Unix-based gLite middleware commands being used to perform a series of operations. The glite-wms-job-submit command is used to submit a JDL file describing a job to the Grid. Upon submission the job is assigned a unique identifier by the Grid middleware (https://cagraidsvr20.cs.tcd.ie:9000/yT2G0MHTu8yUdrWHl4rLLg), which is passed as an argument to the commands for querying the status of a job (glite-job-status) and retrieving the output of the job when the status of the job is "Done" (glite-wms-job-output). There has also been research into developing more intuitive Graphical User Interfaces (GUIs) for Grid technologies. Examples of these include the Genius [49] portal, Migrating Desktop [50], GridSphere [51] and P-Grade [52].

```
[lyttleto@ui ~]$ ./glite-wms-job-submit -a hello.jdl

Connecting to the service https://cagraidsvr20.cs.tcd.ie:7443/glite_wms_wmproxy_server


====================== glite-wms-job-submit Success ======================


The job has been successfully submitted to the WMProxy
Your job identifier is:

https://cagraidsvr20.cs.tcd.ie:9000/yT2G0MHTu8yUdrWHl4rLLg

==========================================================================

[lyttleto@ui ~]$ ./glite-job-status https://cagraidsvr20.cs.tcd.ie:9000/yT2G0MHTu8yUdrWHl4rLLg


*************************************************************
BOOKKEEPING INFORMATION:

Status info for the Job : https://cagraidsvr20.cs.tcd.ie:9000/yT2G0MHTu8yUdrWHl4rLLg
Current Status:     Done (Success)
Exit code:          0
Status Reason:      Job terminated successfully
Destination:        gridgate.cs.tcd.ie:2119/jobmanager-lcgpbs-gitest
Submitted:          Wed Sep  6 14:35:20 2008
*************************************************************

[lyttleto@ui ~]$ ./glite-wms-job-output https://cagraidsvr20.cs.tcd.ie:9000/yT2G0MHTu8yUdrWHl4rLLg

Connecting to the service https://cagraidsvr20.cs.tcd.ie:7443/glite_wms_wmproxy_server


================================================================================
                        JOB GET OUTPUT OUTCOME

Output sandbox files for the job:
- https://cagraidsvr20.cs.tcd.ie:9000/yT2G0MHTu8yUdrWHl4rLLg
have been successfully retrieved and stored in the directory:
/tmp/user_yT2G0MHTu8yUdrWHl4rLLg

================================================================================

[lyttleto@ui ~]$
```

Figure 2-5: gLite job submission, status, and output retrieval commands.

2.5.1 Input for Grid Jobs

Input for Grid jobs can be specified in a variety of ways. Most of the Grid middleware job description languages allow command line arguments to be specified for executables. When describing a job using the gLite JDL for example, the "Arguments" attribute is used to specify what command line arguments will be used when the executable specified in the JDL file is invoked on the Grid. In the following JDL file, the executable is a program which takes one command line argument, as specified by the line "Arguments = 2.44".

```
Executable = "/bin/analyseExperiment";
Arguments = 2.44;
StdOutput = "std.out";
StdError = "std.err";
OutputSandbox = {"std.out", "std.err"};
```

The above JDL file defines the location of the executable on the gLite User Interface (UI) as /bin/analyseExperiment. When this executable is invoked on the Grid, it will use 2.44 as a command-line argument, as shown below:

```
analyseExperiment 2.44
```

As mentioned in the descriptions of Grid middlewares previously, it is a shortcoming of current job description languages that constraints on the number of inputs and types of data cannot be defined. There is no means of specifying how many arguments should be defined for an executable using the "Arguments" attribute, or what the datatypes of those arguments should be. Errors that occur due to incorrectly typed arguments, or the wrong number of arguments being defined, will not be detected at the time of job submission. If an argument is of an invalid datatype, or an incorrect number of arguments is specified, a runtime error will occur when the job executes on the Grid.

Files can also be specified as input for Grid jobs. The following gLite JDL file specifies an input file using the InputSandbox attribute (InputSandbox=("jobInput.dat")).

```
Executable = "/bin/searchData";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = {"jobInput.dat"};
OutputSandbox = {"std.out", "std.err"};
```

As with command line arguments, the datatypes contained in such an input file cannot be defined using current Grid job description languages, except as job-specific arguments, so errors caused by invalid data contained in input files cannot be detected as the job is submitted. If a job executing on the Grid requires an input file containing integer data, but instead a file containing floating point or textual data is specified in the JDL, a runtime error will occur when the job executes on the Grid.

In addition, in cases where input files contain a number of inputs delimited by a special character (such as Comma Seperated Version (CSV) files) job description languages do not allow the number of inputs per line, nor the delimiter character for the data, to be specified. Errors in these will not be detected until the job executes on the Grid. Finally, storing input and output data for jobs in files means that data cannot be sorted and filtered in the way that it can be in relational databases.

2.6 Database Technology and Grids

There is prior and continuing research into the integration of Grid and Database technologies. Generally, this research is investigating two problems:

1. How can Grid jobs access data stored in a database?

2. How can data be submitted to the Grid using relational interfaces?

A third potential integration, that of harnessing database technology to perform services that are provided by multiple applications in existing Grid middleware, does

not seem to be under investigation elsewhere, but is a major subject of this thesis, as will be detailed in Chapter 3.

Two of the best known research efforts into the first problem are the Open Grid Services Architecture Data Access and Integration (OGSA-DAI) [53] initiative and the Grid Data Source Engine (G-DSE) [54]. However, the second problem is more relevant to the issue of type checking data at the time of job submission introduced in the previous section. Providing a relational interface to a Grid, through which data can be submitted, allows data to be type checked as it is submitted to the Grid for processing. XG [3] and GridDB [4] are two projects which, like Infogrid, are researching relational interfaces to Grid middleware.

2.6.1 XG

XG [3] was developed by researchers in IBM. It is a proposed architecture for data processing based on a fusion between a data and a computation layer, but it has not been fully implemented. The data layer is a relational database application which serves as an interface to the Grid. XG uses IBM's DB2 for this purpose. The computational layer is a Grid middleware developed by IBM for the XG project. The data layer communicates with the computation layer via DB2 User Defined Functions (UDFs). UDFs provide type checking and structure for input data to applications executing on the XG Grid. The syntax for creating a DB2 UDF is shown below.

```
CREATE FUNCTION <function_name> ([<argument_name> <datatype>,...])
   RETURNS <datatype> LANGUAGE SQL
```

Invoking a UDF with an incorrect number of input arguments, or with incorrectly typed input arguments, will result in an error. In order to submit jobs to the XG Grid middleware, data is retrieved from the DB2 database using a SELECT statement, and is sent to the Grid for processing with a UDF. An example is shown below, where

a UDF called "sequenceAlign" is used to perform the BLAST sequence alignment operation (as described in section 2.3) to generate similarity scores for genetic sequences.

```
SELECT sequenceAlign(sourceSequence, targetSequence)
   FROM GeneticSequences
```

This query selects the contents of the sourceSequence and targetSequence columns from the GeneticSequences table and uses them as input for the sequenceAlign UDF. The tuples returned by this query will contain the values obtained when the sequenceAlign function is executed with the sourceSequence and targetSequence columns as input.

The architecture of the XG Grid middleware is similar to that of other Grid middlewares. XG uses a two-tier scheduling mechanism to match jobs to computing resources on the Grid. The main components of this architecture are:

- MetaScheduler - schedules jobs to cluster resources on the Grid
- XGResDiscovery - provides information on cluster resources, which is used by the Metascheduler when allocating jobs to clusters
- XGScheduler - schedules jobs to resources on a local cluster
- Local cluster - performs execution of jobs

This architecture is illustrated in Figure 2-6 (which is a simplification of a diagram illustrating the XG architecture in [3]). The steps involved in processing data using XG are numbered as follows:

1. The UDF which submits data stored in a table to the Grid for processing is invoked from within DB2.
2. The data is sent to the MetaScheduler, which determines which cluster on the Grid the job should be submitted to.

3. The MetaScheduler submits the job to a cluster for processing. The details of the job are sent to the XGScheduler, which decides which resources on the cluster are to be used to process the job.

4. The data from the data layer is transferred to the Cluster Store, a data storage facility used by the cluster. When the cluster is running the job, it will access the data from the Cluster Store.

5. The XGScheduler component uses the XGResDiscovery component to locate resources on the cluster which can perform the job.

6. Data is moved from the Cluster Store to the worker node on the cluster where the data is processed

7. The job execution is initialised on the cluster.

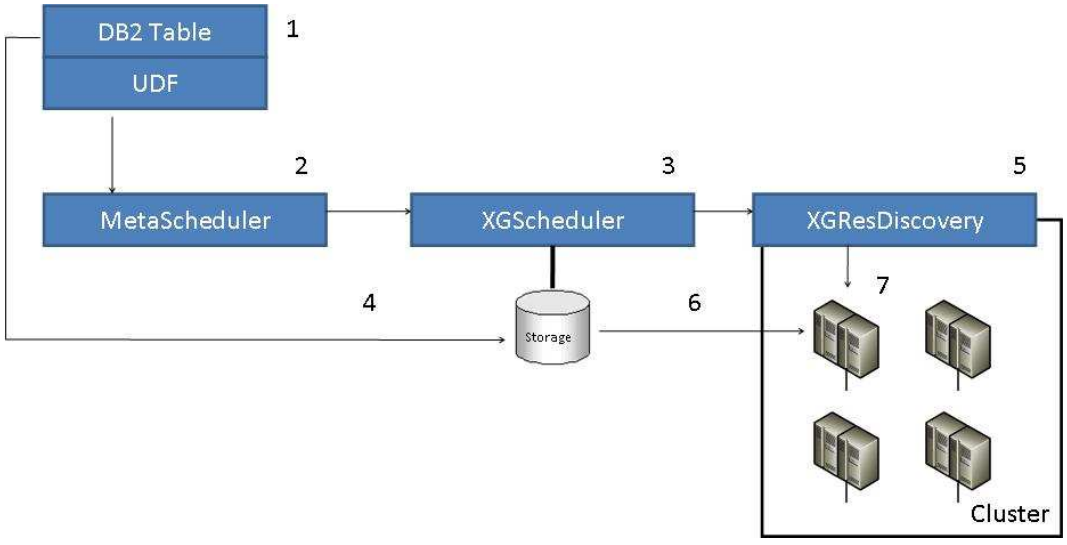


Figure 2-6: Job submission using XG (simplified from diagram in [3]).

A shortcoming of the XG prototype is that it does not implement the MetaScheduler, therefore jobs cannot be distributed amongst multiple sites of clusters. In addition, the XG architecture does not specify how security is implemented. Infogrid will use the security mechanisms of the database technology used to implement it to restrict access to the Grid to authenticated and authorised users. The XG architecture also

does not use the DB2 database to implement other services in the underlying Grid middleware. Unlike Infogrid, using a database technology to provide a number of services that are implemented using separate components in Grid middleware is not investigated by the XG project.

There may be potential problems that arise when using XG due to the fact that job submission to XG is a "blocking", or synchrononous operation: when a client submits a job via a UDF, control of the client will not return to the user until the job has completed. In contrast, job submission to existing Grid middleware, whether using command line executables or APIs, is an aysnchronous operation. Users submit jobs, and can then perform other tasks, perhaps periodically checking the progress of the job. When the job has completed, the user can retrieve its output. However, when submitting jobs to XG, the SELECT statements used to invoke the UDF that sends data to the Grid for processing will "block" until the output of the Grid job is available, and the Grid client is unavailable for other purposes during that time. Grid jobs can require a lengthy amount of time to complete, as they can involve executing computationally expensive algorithms and processing large amounts of data. It is not desirable that job submission to Grid middleware is a blocking operation. It is preferable that this operation is an asynchronous, non-blocking operation. Users will submit jobs to Infogrid by inserting rows into tables in a database. These insert operations will be non-blocking, which are a more suitable job submission mechanism for a Grid.

2.6.2 GridDB

GridDB [4] provides a relational interface to the Condor middleware. GridDB represents applications that are executed on the Grid as tables in a relational interface. This enforces type checking and provides a structure for input data to Grid jobs. These relational interfaces are specified with a language created for the GridDB project, the Functional Data Model with Relational Covers (FDM/RC).

The first step in specifying a relational interface to an application using FDM/RC is to define datatypes for inputs to and outputs from this application. Using again

the example of gene sequence alignment, the inputs for an application that uses the BLAST algorithm to perform sequence alignment on a pair of gene sequences is shown below:

```
transparent type in=sourceSequence:string, targetSequence:string;
transparent type out=output:float;
```

These statements define two datatypes, "in" and "out". Datatypes can be composed of several primitive datatypes. For example, the "in" datatype is composed of two string components, sourceSequence and targetSequence, which represent the gene sequences being compared. The "out" datatype has only one component, a float called output. The use of the "transparent" keyword indicates that these types are composed only of primitive datatypes. GridDB allows datatypes composed of non-typed data called opaque datatypes, similar to the Binary Large Object (BLOB) datatype in existing Relational Database Management Systems (RDBMSs), to be used.

FDM/RC is also used to define functions which use these datatypes. In the example below, a function called "sequenceAlign"is defined.

```
atomic fun sequenceAlign (params:in):(result:out) =
exec(''clustalW'',[(''paramsFile'',params)],
   [(/.outputFile/, result, ''adapterX'')]);
```

The atomic keyword indicates to the GridDB software that this function represents an application that is executed on the Grid. Other GridDB function types are composite functions, which are composed of several atomic functions, and map functions, which iterate through a set of inputs and apply a function to each of the elements in

the set. The sequenceAlign function accepts as input a variable called params which is of type "in" (params:in), and returns as output a variable called result which is of type "out" (result:out). The exec function submits a job to the Grid that invokes an executable called "clustalW" (a widely used program for performing sequence alignment operations) with the data contained in the params variable as input. GridDB defines the process of mapping input parameters from the relational layer to input files for executables as "unfolding", and the process of mapping the output of these executables to the output of the function as "folding". The unfold operation is defined in the second argument to the exec function (`[(''paramsFile'',params)]`). The params variable will be written to a file called "paramsFile". This file is used as input by the clustalW executable. The fold operation is defined in the third argument to the exec function (`[(/.outputFile/, result, ''adapterX'')]`). A program called "adapterX" will read the output of the clustalW executable from a file named "outputFile", and assign this data to the "result" variable.

Another language created for the GridDB project, the Data Manipulation Language (DML), is used to express the statements that create tables representing functions, to submit data for processing and retrieve the output from jobs. The following three DML statements declare tables representing input and output data for the sequenceAlign function. The first two statements declare two tables, SequenceInput and AlignmentScores. SequenceInput is a set of elements of type in, and is used for holding the input for GridDB jobs that invoke the sequenceAlign function. AlignmentScores is a set of elements of type out, and will store output from these jobs. The third statement sets the SequenceInput table as input to the sequenceAlign function, and the AlignmentScores table as the destination for output from this function.

```
SequenceInput:set(in);
AlignmentScores:set(out);
(AlignmentScores) = sequenceAlign(SequenceInput);
```

An INSERT statement is used to submit a job which invokes the sequenceAlign function on the Grid, as shown below.

```
INSERT INTO SequenceInput VALUES in={('GTGGCGGTC','GTAATGGC')};
```

The fold and unfold operations take place after this insertion. The output from this job is inserted into the AlignmentScores table when it is available. When the output is available, it can be retrieved using a SELECT statement and a GridDB function called autoview.

```
SELECT * FROM autoview(SequenceInput, AlignmentScores);
```

Each function declared in GridDB has a process table associated with it. This table maps inputs to outputs for all invocations of that function. If users want to view input and output sets for a function, the autoview function performs a table join between data in the input table, the process table and the output table, and displays the results of the join to the user, as illustrated in figure 2-7.

GridDB also enables the reuse of cached results. If a row is inserted into a table which contains values that have previously been evaluated by the function that the table serves as an interface to, the unfold function will not be evaluated for that

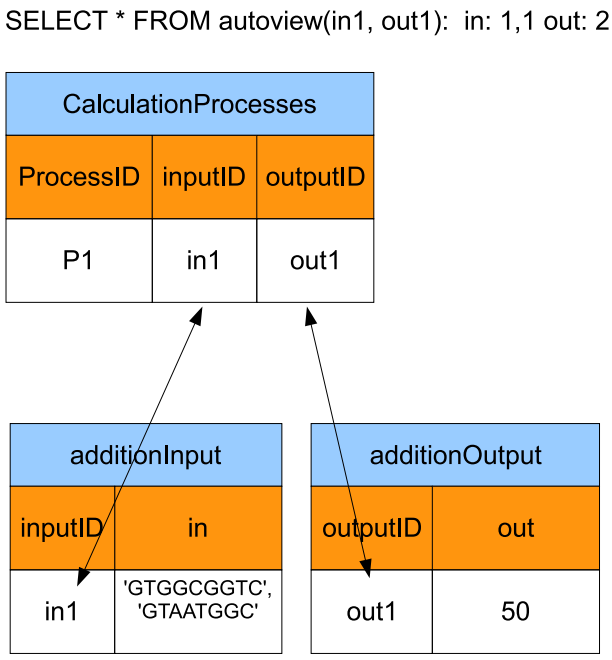


Figure 2-7: Retrieving results from GridDB function (based on diagram from GridDB paper [4]).

row. As is the case with XG, GridDB does not use database technology to implement other Grid middleware functions beyond job submission. There are also several shortcomings with the GridDB prototype. As with XG, GridDB integrates database and Grid technology, but uses custom software only deployed within the project for one of these technologies. The relational layer used by GridDB is an implementation of the FDM/RC and DML languages, which are unique to the GridDB project, and are not widely used relational languages such as SQL. This may increase the difficulty with which users learn to use GridDB. There is little information available that details precisely what functionality the relational interface in GridDB offers, or the performance and reliability of the software.

There is also no API available for GridDB. Data must be inserted into GridDB's relational interface by manually creating and executing INSERT statements. This is not a scalable approach if large numbers of inputs are to be processed, or if GridDB is to interoperate with external applications. In addition, performing joins across multiple tables to link input and output data is inefficient. Table joins can take a lengthy period of time to perform, compared to searching for input and output data

within a single table. Finally, the GridDB prototype does not restrict who can access the underlying Grid resources, or the actions they can perform, by providing security services such as authentication or authorisation.

2.7 Conclusion

In this chapter, the field of Grid computing was described in more detail. An abstract model of Grid middleware was described which contained the main components common to most types of Grid middleware. Various research efforts in the field of Grid computing were described, and grouped into three categories: research that creates and maintains Grid computing infrastructures, research which runs applications and experiments on these Grid infrastructures, and research into developing the middleware that is used to manage the execution of jobs on these infrastructures. Particular attention was given to these Grid middlewares: Condor, the Globus Toolkit and gLite. Each of these middlewares was composed of a set of components similar to the set of components in the generic Grid middleware described earlier in the chapter. The potential for using a single database technology to provide the services offered by some of these components was highlighted. A job description file for each middleware was presented, and potential problems with type checking and jobs with incorrect numbers of inputs were described. Two projects which used database technology to check the type and number of inputs for Grid jobs, XG and GridDB were described. Shortcomings of the relational interfaces to Grid middleware offered by XG and GridDB (e.g. lack of security, blocking job submission operations, lack of APIs, use of non-standard query languages) were highlighted. There is potential for Infogrid to be an advance on these projects by using a database technology to implement a number of functions required by a Grid, beyond job submission, which neither XG nor GridDB aim to do. In the next chapter, the Infogrid hypothesis will be presented in more detail.

Chapter 3

Infogrid: a database approach to Grid middleware

3.1 Introduction

Section 2.6 discussed two possible ways of integrating database and Grid technologies, and indicated a third potential integration, that of harnessing database technology to simplify Grid middleware, which does not seem to be under investigation elsewhere.

This chapter highlights the components of a Grid middleware that could be implemented by Infogrid using a database technology, and also proposes three different means by which jobs can be submitted to Infogrid. An architecture for the Infogrid prototype is then proposed. The chapter concludes with a discussion of the necessity of security and fault tolerance mechanisms in the Infogrid architecture.

3.2 Implementing Grid components using database technology

The following is a list of the operations on Grid components that an interface to Grid middleware should enable:

- User interface: the user interface should allow users to submit jobs to the Grid

which accept input as either command line arguments or files, query the status of jobs, cancel jobs, etc. The interface should allow hardware and software requirements for these jobs to be specified.

- Job scheduler: the job scheduler should listen for jobs that are submitted from the user interface.

- Remote resource: the remote resource should listen for jobs that are submitted from the job scheduler. The interface provided by this resource should allow a variety of operations to be performed, for example, querying the status of a job, cancelling a job, define environment variables for jobs, specify a directory that a job must execute in, etc. As with the user interface, it should be possible to specify hardware and software requirements for jobs.

- Information system: the information system must provide information on resources on the Grid in response to queries made by either users or job schedulers. The information system must also enable Grid resources to publish information to it.

- Logging: the logging system must provide information on past events which have occurred on the Grid, and enable Grid components to publish information to it.

Access to the services provided by these interfaces must be restricted to authenticated and authorised users using security mechanisms.

As was stated previously, Grid middleware components require multiple generic services. For example, both the Information and Monitoring System and the Logging and Bookkeeping Server components of the gLite middleware require mechanisms for storing and retrieving information. Condor stores logging information in a number of files located on the various machines contained in a Condor cluster, and uses the Collector component to gather information on the current state of machines in the cluster. The Globus toolkit stores logging information in individual files for its components, such as the Gatekeeper log and LRMS log, and uses MDS to store information on the current state of Grid resources.

Components of Grid middleware also require APIs which allow their functionality to be accessed remotely. As illustrated previously in Figure 2-4, components of the gLite middleware interact using APIs such as the WMS API, BDII API, L&B API and the GRAM API which is used by CondorG. Figure 2-3 illustrated a Grid site implemented using the Globus toolkit, where the MDS API is used to get information on available Grid sites and the GRAM API is used to submit jobs to a particular site. Security services, such as authorisation and authentication are required by all these components, in order that only parties who are authorised to access the Grid can use them.

I wish to postulate that a single database technology can be used to implement all these services. For example, in the gLite middleware, the Information and Monitoring System, and the Logging and Bookkeeping Server, could be implemented using a single database technology, instead of using an OpenLDAP server and a MySQL server respectively as at present. Relational database technology could also be used to implement an RPC mechanism. Figure 3-1 shows how tables in a database schema can represent an interface by which clients can invoke functions on remote applications. In this example, the ServerTable in the database represents an interface to a remote middleware component. ServerTable contains one column, Action, representing an operation on the remote component that the client wishes to invoke. When the client inserts a row into ServerTable, the server retrieves this row and performs an action associated with the value in the Action column.

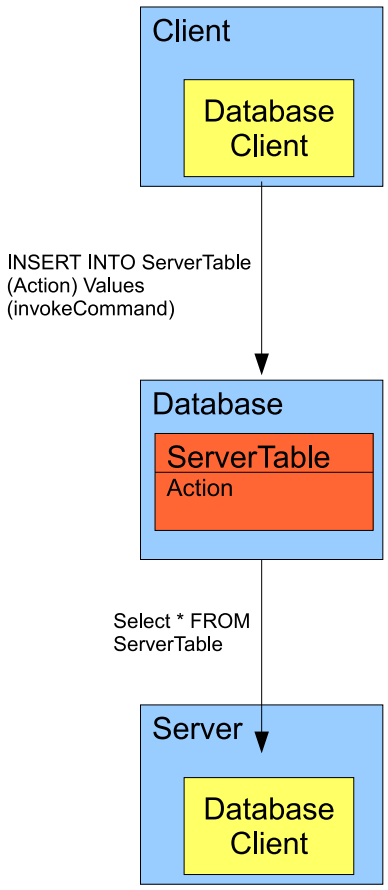


Figure 3-1: RPC using a database

I call this conceptual Grid middleware system which uses a single database technology to perform these multiple services *Infogrid*. As an example, Figure 3-2 illustrates how a database technology could be used in the gLite middleware to perform the roles of information and monitoring system, logging and bookkeeping server and how the database API could be used in place of the various other APIs to enable the functionalities of the middleware components to be invoked remotely. In addition to offering the interfaces to Grid components specified in this section, Figure 3.2 illustrates how the concept of using a database technology to provide services can be applied more broadly, with interfaces to the SE and File and Replica Catalog also implemented using a database API.

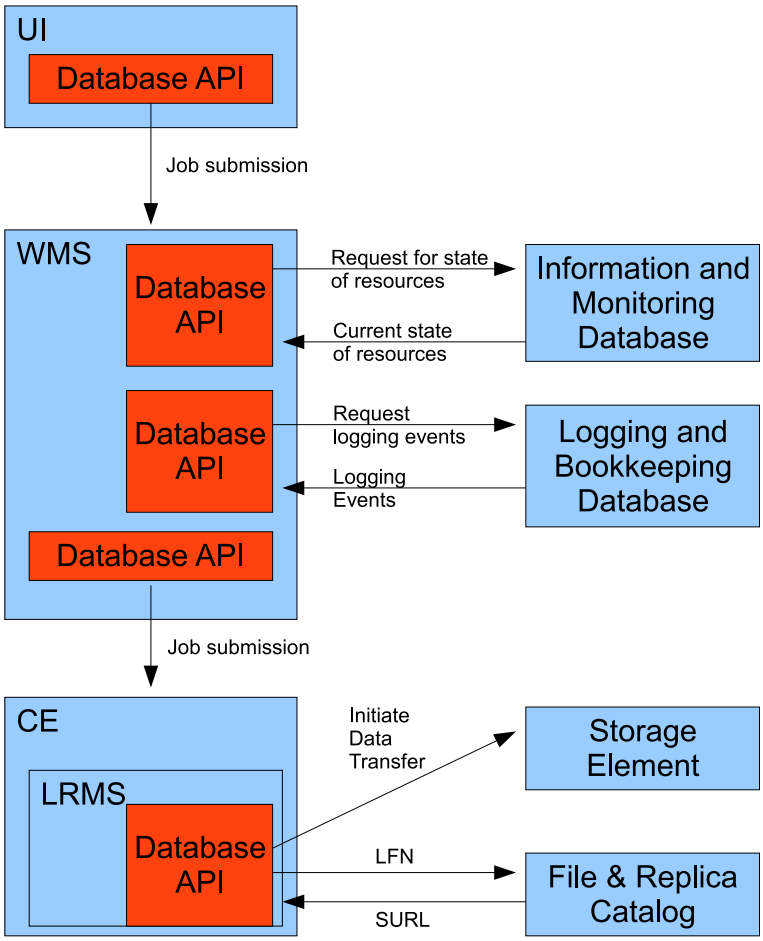


Figure 3-2: Extending the use of a relational database in the gLite middleware

3.3 Grid job submission via relational tables

In addition, a database technology can be used to perform type checking of data as it is submitted to the Grid. As outlined in section 2.5.1, the job description formats used by various Grid middlewares do not allow data types to be specified for input data. If input for a Grid job is incorrectly typed or delimited, error detection does not take place at the time of job submission, but when a Grid resource attempts to execute the job. This can be a significant period of time after the job has been submitted, as there is a latency generated by the sequence of processes that take place between job submission and execution. This sequence of processes typically include the Grid middleware querying the information system to discover the current state of resources on the Grid, scheduling the job for execution on a particular resource,

transferring the job executable and input files to that resource, and the resource becoming available to execute the job. Runtime errors due to an incorrect data type being used, or an incorrectly formatted input file, may occur several minutes after the job submission (in the case of the gLite middleware, assuming there is a resource immediately available to execute the job, typically between two and five minutes). In addition, the user is not automatically informed that the job has failed. The user must check the status of the job in order to determine if it has executed correctly. Users may be too busy or forgetful to check the job status, which can lead to errors not being detected until lengthy periods of time have elapsed since the data was submitted to the Grid for processing.

A relational database could be used to implement an interface to the Grid which enforces type checking at the time of job submission, and provides structure for input data. As described in a previous section, input for jobs can be command line arguments or files. Individual rows in a table could represent command line arguments for jobs, while sets of rows or entire tables could represent larger datasets that are stored in files. I propose using a database technology to implement three job submission interfaces to the Grid: active tables, the SubmitDataset table, and the SubmitLargeDataset table. These are described in Sections 3.3.1, 3.3.2 and 3.3.3 respectively.

3.3.1 Active tables

Many applications executed on the Grid accept a number of command line arguments as input, and return a particular output for that set of inputs. Infogrid uses active tables to provide an interface to these applications. Active tables are similar conceptually to triggers and stored procedures in existing RDBMS technologies. Triggers are instructions to perform a particular action in response to another action (such as an insertion, deletion, or update) that takes place in the database. Triggers can initiate stored procedures. Stored procedures can be either sets of SQL statements or applications external to the RDBMS which are executed on the machine hosting the RDBMS. However, in the case of active tables the computational process is

outsourced to the Grid, as opposed to being performed by the local machine. The output produced by these jobs is inserted into a table in the database where it can be retrieved by using SQL SELECT statements.

Active tables have input and output columns, into which partial tuples containing only values for the input columns are specified. Each active table is linked to an application which Infogrid executes as a job on the Grid using the data inserted into the active table as input. When the job has completed, Infogrid updates the tuple in the active table so it contains both input and output values. Infogrid consults a table in the database called the active tables metadata table to determine which columns represent input and output.

The database technology used to implement active tables performs type checking as data is inserted, and will immediately report any errors to the user, who can then take the appropriate remedial action. The relational interface also provides a structure for input data. The scenario described previously where data in input files was delimited incorrectly by the user resulting in runtime errors on the Grid would not happen in the Infogrid application, as the database table prescribes a syntax for defining the expected structure of input data. Data in active tables can also be selected and filtered using SQL SELECT operations.

The analysis performed by the GRIST project on astronomy data, as mentioned previously in section 2.3, where data from a sky survey is analysed to search for visual phenomena of interest, could be implemented as an active table. Figure 3-3 shows an example of an active table which processes data from a sky survey. The blue cells in the table represent input to the active table. The input columns (Radio, Opt r, Opt i, Opt z and Infrared) are readings taken from the sky survey data which were obtained using a radio telescope. The "Quasar Candidate" and "Transient" columns represent output from the active table. These columns are probability values that the telescope readings in a particular row indicate the presence of these phenomena. Initially, these columns are empty.

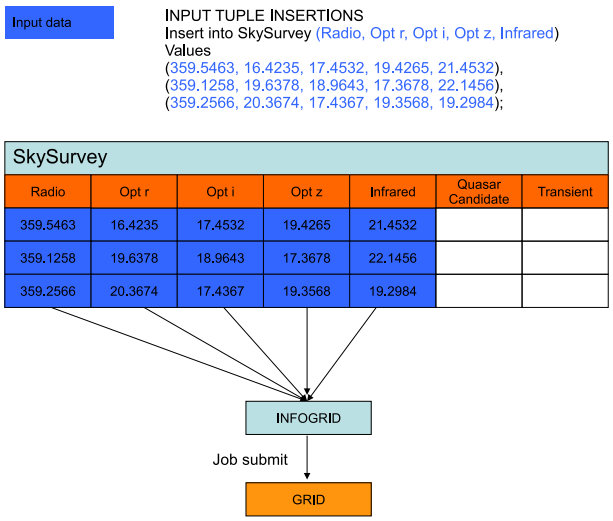


Figure 3-3: Active table processing data from a sky survey - input data only.

The SkySurvey active table submits jobs to the Grid which invoke an executable which uses the data in the input columns as command line arguments for each of the rows. When the output is available from these jobs, the rows in the active table are updated so that the output columns contain this output, as illustrated in Figure 3-4, where the green columns represent output from the executable.

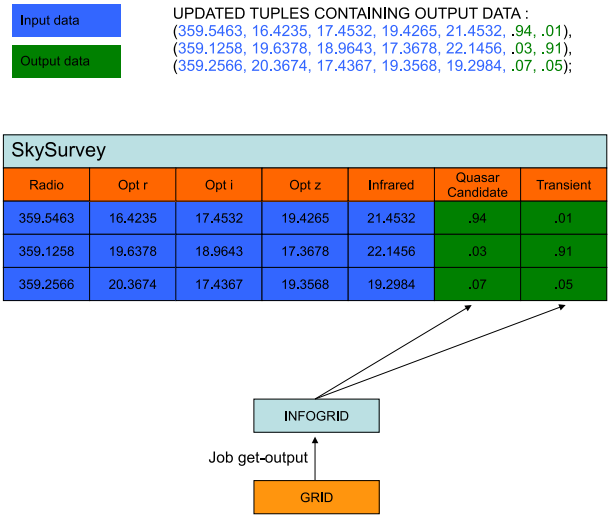


Figure 3-4: Active table processing data from a sky survey - input and output data.

High throughput sequence alignment is another potential application that can be invoked by inserting input data into active tables. As was also mentioned in section 2.3, sequence alignment operations are performed on the Grid in many research

projects. Sequence alignment algorithms, such as BLAST [28], are used to compare genetic data sequences to see how similar they are. Sequence alignment is a potentially computationally expensive algorithm, and is often performed on large data sets. An active table can implement the BLAST alignment algorithm, as illustrated in Figure 3-5. As in the previous example, input to the active table is represented by the blue columns, and output is represented by the green column. The sequence alignment computations are distributed across the Grid, and the results of these computations can be filtered and sorted using SQL statements. In Figure 3-5, an SQL SELECT statement is shown which retrieves the sequence alignments from the BlastAlignment active table which have generated an output score (as indicated by the SimilarityScore column) greater than 90.

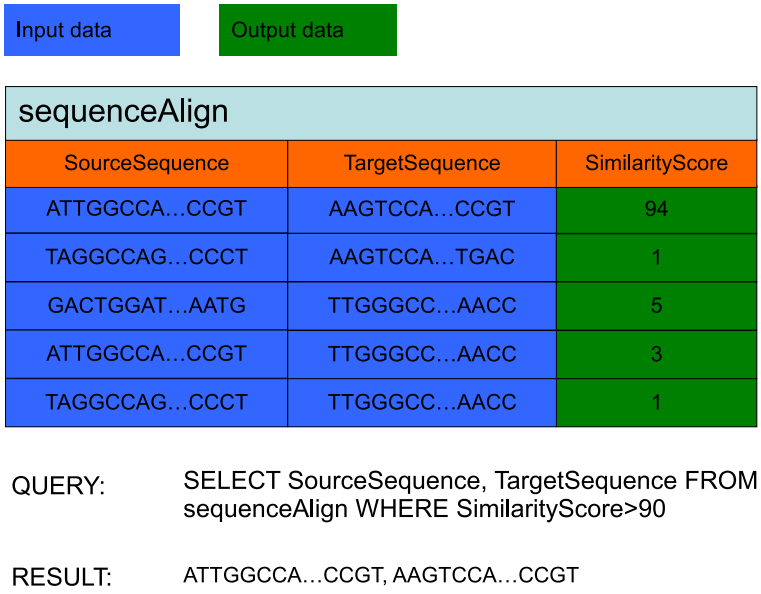


Figure 3-5: Filtering results from sequence alignments performed by an active table

Users of active tables do not have to be aware of the underlying Grid software that is processing the data. The Infogrid software automatically polls the status of jobs that are performing functions on data inserted into an active table, and inserts the output of those jobs into the database when it is available. Authentication and authorisation rules can also be specified in order to restrict access to active tables, in order to prevent unauthorised users from submitting data to the Grid by inserting data into active tables, or reading data from tables with restricted access. Active

tables also allow the reuse of cached results. If a tuple is inserted into an active table, which has the same input values as a tuple already in the table, that tuple will not be processed by Infogrid.

Current Grid middleware does not provide facilities for connecting streams of data to the Grid in such a way that jobs are submitted to the Grid that perform a particular function on data as it appears in the stream. Instead, current Grid middleware systems require the data to be collected and submitted manually. Infogrid will issue continuous queries on active tables, which retrieve data as it is streamed from a source (or multiple sources, as insertions into active tables can originate from many producers of data) into an active table. allowing Infogrid to submit data to the Grid as it is streamed into an active table. An example of an existing application that processes streams of data using the Grid is the AstroGrid-D application [55]. Many eScience applications, especially in Astrophysics, process continuous data streams. These applications may need to process particular data from these streams which must be filtered out from the entire stream. The data stream management system (DSMS) for AstroGrid-D is designed to cope with continuous data streams. Users may publish and/or subscribe to streams. The AstroGrid-D middleware is built as a service oriented architecture. The functionality is realized as web services either using standard web services on SOAP or stateful web services using the Web Services Resource Framework.

Unlike the XG architecture examined in the previous section, active tables provide an asynchronous mechanism for submitting data to the Grid for processing. The Infogrid client application used to insert data into these tables does not wait for the Grid job that processes that data to complete before the user is able to perform additional operations. Operations performed on Infogrid active tables such as inserting or retrieving data are performed using commands expressed in SQL, which is a widely used, mature standard for performing such operations. This is in contrast to GridDB's usage of the FDM/RC and DML languages which were developed and used solely within the context of the GridDB project. In order to allow large numbers of rows to be inserted into active tables, the database technology used to implement

Infogrid will provide an API which can be used by programs to execute large numbers of SQL INSERT statements, unlike the GridDB prototype, which provides no API and requires users to write these statements manually. Operations performed on active tables are also subject to authentication and authorisation, unlike the GridDB and XG prototypes, which do not implement these security mechanisms.

However, active tables as described above have several limitations. They are only suited for invocations of applications which require a small number of input parameters. Submission via active tables of large input datasets (e.g. a dataset containing millions of input parameters, as frequently occur in scientific computing) is not practical, as the active tables would be of an unwieldy size (and possibly in breach of the operating limits of the database software). A user may need to invoke an application that submits a single job to the Grid which uses multiple rows of parameters as input. These scenarios can be more easily handled if the data is referred to indirectly, as discussed below.

3.3.2 Submitting Tables for Processing to Infogrid

Entire tables, or a set of rows from a table, can be used as input for Infogrid. This allows larger datasets to be submitted to Infogrid, and supports automation of bulk data processing for data-intensive sciences. Although this data could be stored independently of a database technology, for example in files as is the case with existing Grid middlewares, the use of a database technology to contain job input and output allows Infogrid to use SQL operations to check the type and structure of this data in order to determine if it conforms to the types and structure that a particular application requires. Jobs that process tables could be invoked by inserting rows into a database table, as illustrated in Figure 3-6. Users insert a tuple into the SubmitDataset table identifying the dataset they wish to process, and the executable they wish to process this dataset with. The Input column in the SubmitDataset table identifies the rows that are input for this job, and the Executable column defines the application that uses this input.

Infogrid can check that the input dataset is composed of the correct datatypes

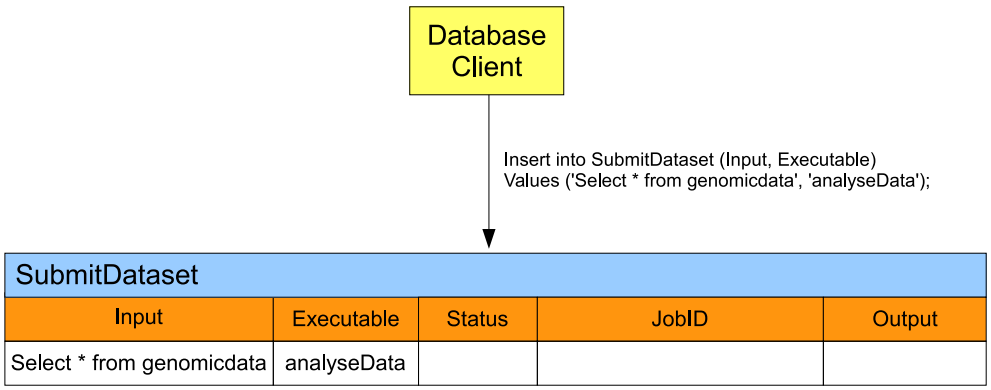


Figure 3-6: Table for processing multiple rows.

and has the right number of columns by consulting a table which contains this information for applications that execute on Infogrid, the InfogridApplications table, as illustrated in Figure 3-7. This table contains three columns. The Executable column defines which executable a row pertains to. The Input column defines the datatypes required as input to this executable. The Output column contains an SQL "CREATE TABLE" statement that is used to create a table that contains the output for a job. Each job submitted to Infogrid via the SubmitDataset table has its own output table; tableName is a unique name for this table that Infogrid generates when the output for a job is available.

InfogridApplications		
Executable	Input	Output
analyseData	INTEGER, INTEGER	CREATE TABLE <tableName> (FLOAT Result)

Figure 3-7: Input and output for applications executing on Infogrid.

If a row is inserted into the SubmitDataset table where the structure and datatypes of the dataset defined in the Input column are different to those defined in the Input column for the executable as specified in the InfogridApplications table, the status column of that row is updated to contain a message indicating an error. Otherwise, the data specified in the Input column is written to a file, and is transferred along with the executable to a resource on the Grid where the executable is invoked using the file as input. When the output is available, it is transferred into an output table,

and the initial row is updated so that the Status column is set to "Cleared" and the Output column is set to the name of the table where the output is stored, as illustrated in Figure 3-8.

SubmitDataset				
Input	Executable	Status	JobID	Output
Select * from genomicdata	analyseData	Cleared	yT2G0MHTu8yUdrWHl4rLLg	genomicOutput268

Figure 3-8: Table for processing multiple rows, after job completion.

3.3.3 Submitting Unstructured and Larger Datasets to Infogrid

Grid jobs which require large datasets as input may also require additional data management software, used to transfer large datasets from one location on the Grid to another, to execute. This software is not invoked by jobs that are submitted to Infogrid via the SubmitDataset table. In addition, not all input files for applications executed on the Grid will have the format assumed by the SubmitDataset submission mechanism, i.e. a set of tuples from a single table, all of equal length, that contain sequences of atomic values of the same format. Input data for jobs can be more flexibly structured, for example, tuple elements can have multiple values, or consist of data from multiple tables, or be encoded in Extensible Markup Language (XML). The applications that process data deal with this heterogeneity of input. Jobs using datasets which require specific data management software, or which do not fit the assumptions of the SubmitDataset table, can be submitted to Infogrid by inserting a row into the SubmitLargeDataset table, as shown in Figure 3-9.

Inserting a row into the SubmitLargeDataset table will result in Infogrid submitting a job to the Grid which uses the dataset specified in the "LogicalFileName" column as input. The underlying Grid data management software (e.g. file catalogs, grid file transfer applications) is used to transfer the data identified by the LFN to where the job executes. File catalogs are Grid software that serve as a directory of files distributed across the Grid, which does not refer to files by their physical location,

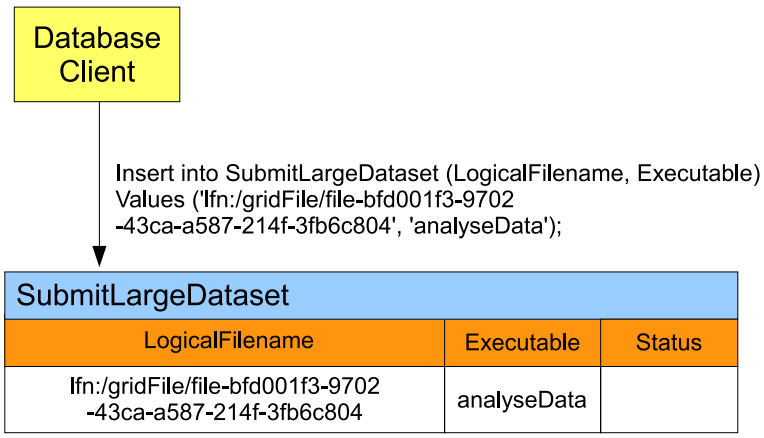


Figure 3-9: Table for processing large datasets, before job completion.

but by using logical identifiers (i.e. LFNs). Grid file transfer applications can specify files for transfer from one machine to another using these logical identifiers.

After the job has completed, the row is updated to indicate the status of the job, and the LogicalFileName column contains the name of the file containing the job output, as shown in Figure 3-10. This file can be obtained using data management tools provided by the Grid middleware.

SubmitLargeDataset		
LogicalFilename	Executable	Status
lfn:/gridFile/file-bfd001f3-9702 -43ca-a587-214f-3fb6c804_OUTPUT	analyseData	Cleared

Figure 3-10: Table for processing large datasets, after job completion.

In principle this job submission interface could optionally provide type checking by following the LFN, with definition of an associated type checking function. For prototyping simplicity, it does not provide type checking. However it extends the range of jobs that can be submitted to Infogrid via insertions into database tables. The SubmitLargeDataset table enables the submission of jobs which require input data which is not stored as rows in a database table, or require the data management tools used to manage large input datasets. Such jobs could not be submitted to Infogrid via active tables or the SubmitDataset table.

3.4 Infogrid prototype architecture

I now propose an architecture for a prototype of Infogrid, which will test the hypothesis that a single database technology can be used to implement the functionality currently provided by a variety of applications in existing Grid middleware, and can provide an interface for job submission to the Grid which can perform type checking of input data for a job as it is submitted. The following Grid middleware functionalities are implemented using a relational database technology in this architecture:

- Job submission: in addition to the regular interfaces offered by Grid middleware for job submission, users submit data for processing on the Grid by performing SQL INSERT operations on tables.
- Information and Monitoring: information on the current state of Grid resources is stored in a database.
- Logging and Bookkeeping: a log of events that occur in the Grid middleware is stored in a database.
- RPC: a database is used to implement an RPC mechanism that allows the functionality of Grid middleware components to be invoked by remote clients by inserting rows into tables.
- Security: a database's authorisation and authentication features are used to perform all of the above operations securely.

Figure 3-11 illustrates this architecture.

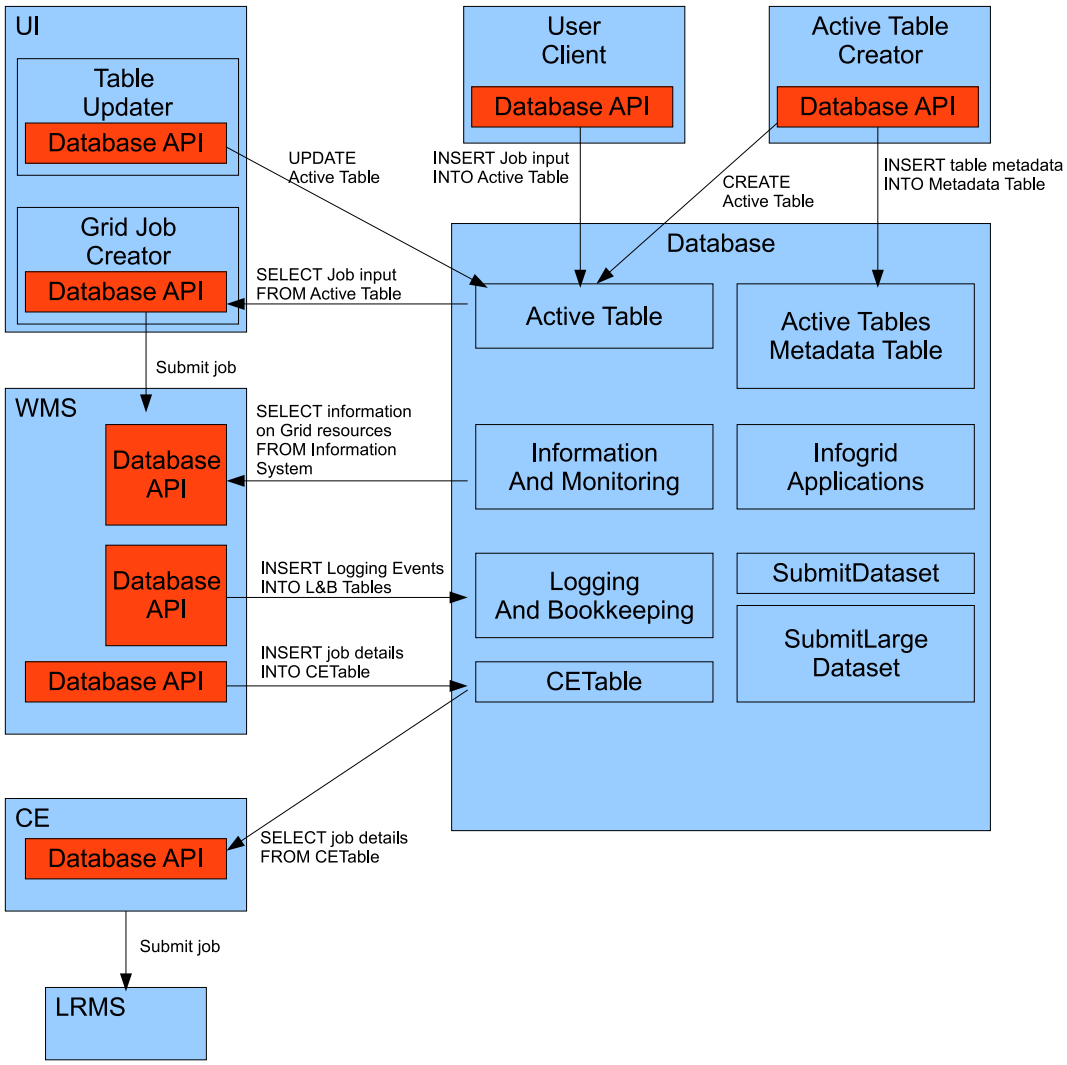


Figure 3-11: Infogrid Architecture.

This architecture however requires that the executable required by jobs is already in place on a particular machine. If a user of this Infogrid prototype wishes to create an active table which invokes a new application for example, they must transfer the application to the UI in advance. Jobs submitted to Infogrid via the SubmitDataset table must also invoke an application that is already present on the UI, and jobs submitted via the SubmitLargeDataset table require that the application they invoke can access a Grid File Catalog.

The three job submission methods offered by the Infogrid prototype offer three different usage models. The active tables user model is based on that proposed by GridDB in the previous chapter. Users can create active tables, which can then also

be used by other users, to process tuples subsequently inserted into that table. These tuples will be updated to contain the output of this processing when it is available. For example, in the case of sequence alignment, a user copies an executable that performs this operation to the UI, and defines an active table that serves as a front end for this executable. The SubmitDataset table offers a user model whereby larger datasets, consisting of sets of rows taken from tables in the database, can be specified as input for an executable already present on the UI. The SubmitLargeDataset table offers a user model whereby jobs processing large datasets or data that is not structured as columns in a table format (e.g. tree data structures, XML, etc.) that is stored on the Grid and identified using a logical filename to be submitted. Another user model of the Infogrid prototype would be to provide interfaces to programmers who wished to write applications that invoke operations on the Grid middleware. For example, methods in the database API could be invoked by programs to perform operations on the Grid middleware.

By using the active table and SubmitDataset job submission mechanisms, type checking can be performed, and errors detected, at the time of submission, Users of these job submission mechanisms can use the flexibility and power of the relational data model and SQL to filter input and output data for Grid jobs. For example, as seen in section 3.3.1 of this thesis, users could select all rows from an active table containing pairs of gene sequences and similarity scores for those sequences where the similarity score is greater than 90.

As this is only a prototype, the functionality offered by the Infogrid interface is not as extensive as would ideally be expected from a Grid middleware. Some of the functionality that existing Grid components offer will not be present in the prototype. For example, as outlined at the beginning of this chapter, the full range of functionality offered by Grid job submission tools is quite wide. Existing job submission interfaces can be used to cancel jobs, specify software/hardware requirements for jobs, define maximum running times or environment variables for jobs, specify a specific resource that a job must execute on, etc. The job submission interfaces offered by the user interface and gatekeeper components of the Infogrid prototype however

will only implement the minimum functionality required to submit data for processing to the Grid, and get the results of this processing. While production quality middleware would demand an interface that offers a wider range of functionality, the Infogrid prototype is a proof-of-concept for which a simplified interface is acceptable.

3.5 Infogrid components

As illustrated in Figure 3.11, the components of Infogrid are:

- User Client: this is implemented using a database technology, and as described previously is used to submit jobs to Infogrid via three types of database tables (active tables, the SubmitDataset table, and the SubmitLargeDataset table).
- Active Table Creator: this component allows users to create active tables.
- Grid Job Creator: takes input specified by insertions into an active table or the SubmitDataset/SubmitLargeDataset tables, and submits a job which uses this input to the Grid.
- Table Updater: this component inserts the output of jobs into the appropriate database table (for example, in Figure 3-11, an active table is being updated with output from a job).
- Information System: maintains information on the current state of resources on the Grid.
- Workload Management System: decides where to execute jobs on the Grid.
- Infogrid CE: allows access to Grid resources and initiates the execution of jobs on them.
- Logging and Bookkeeping System: keeps record of events related to jobs (e.g. submission, execution, completion)

These components will be examined in more detail in the following sections.

3.5.1 User Client

Job submission to the Infogrid prototype can be performed by inserting data into either an active table or the SubmitDataset/SubmitLargeDataset tables. The user client can be any software that uses the database API to execute SQL statements. This software may be a program which inserts large numbers of rows into an active table by invoking functions in the database API, a Graphical User Interface which serves as a front end to the database API, or tools provided with the database that allow users to perform SQL operations from a command line environment. Users are authenticated and authorised before these operations are performed. The process of submitting jobs to the Grid that process this data, finding resources on the Grid for executing these jobs, and retrieving the output from these jobs when it is available is hidden from the end users.

3.5.2 Active Table Creator

The active table creator is used to create active tables. It is used to specify the names and datatypes of the columns in active tables, which of these columns represent input to a function, which columns will hold the function's output, etc. The active table creator uses the active tables metadata table to store this information. The database API is used by the active table creator to invoke CREATE TABLE statements that create these active tables in the database. The active table creator also spawns a Grid Job Creator process for each active table that issues a continuous query on that table.

3.5.3 Grid Job Creator

Grid Job Creators convert data stored in relational tables into a format suitable for submission to the Grid. As an example, the Grid Job Creator in Figure 3-11 is retrieving data from an active table. There are different types of Grid Job Creator for each of the types of database table used by Infogrid to implement job submission interfaces, as will be shown in the next chapter.

3.5.4 Table Updater

The Table Updater updates active tables and the SubmitDataset/SubmitLargedataset tables when the output for jobs is available.

3.5.5 Information System

The information system is consulted by the WMS, which uses the database API to execute SQL operations in order to determine what the current status of resources on the Grid is, in order that resources which are available to execute jobs can be located. The Information System is composed of processes which monitor the state of resources distributed across the Grid, and insert this information into a Grid-wide database, as illustrated in Figure 3-12.

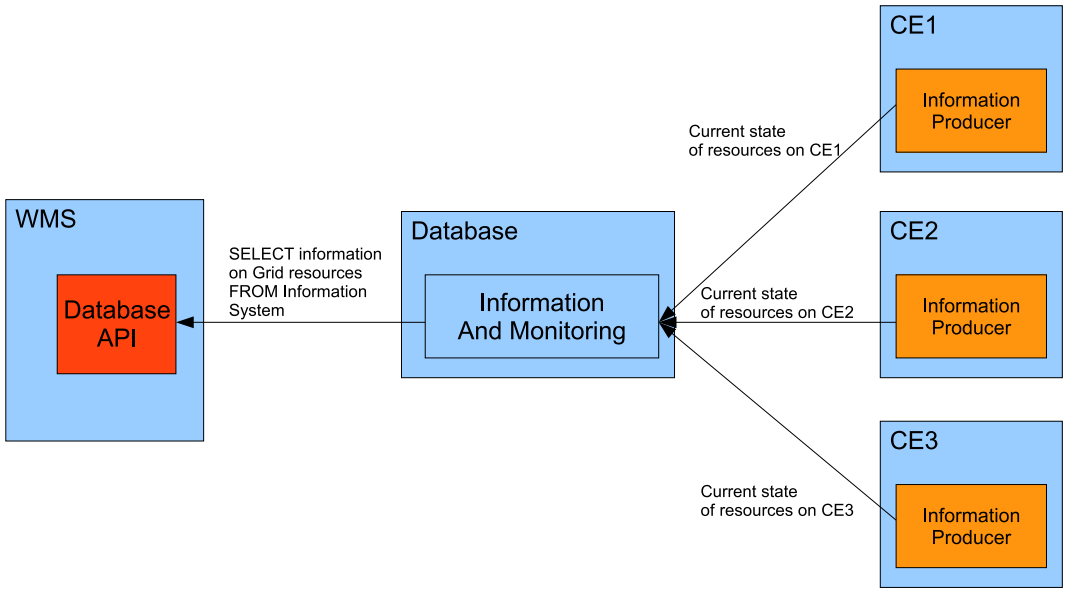


Figure 3-12: Database containing information on Grid resources

3.5.6 Workload Management System

The Workload Management System (WMS) matches jobs submitted to the Grid to resources that are available and suitable for executing those jobs. As shown previously, the WMS queries the Information System to find these resources. When the WMS has matched a job to a resource that will execute it, it submits the job to

that resource by inserting a row into a database table which represents an interface to the resource. Figure 3-13 shows two such tables, CE1Table and CE2Table. The rows that the WMS inserts into these tables contain the names of directories on the WMS containing the "sandbox" for jobs. The sandbox is the collective term for the executable and any input files required by a job.

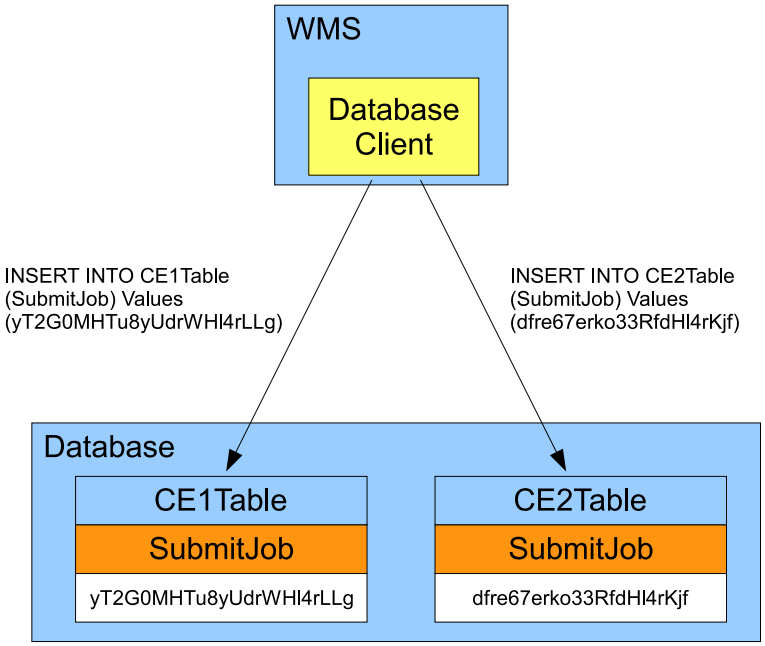


Figure 3-13: Infogrid WMS submitting jobs to Infogrid CEs

3.5.7 Infogrid CE

There are numerous CEs in a Grid infrastructure, typically at least one for each administrative domain which has resources connected to the Grid. Infogrid CEs issue continuous queries on the relational tables that represent the interfaces used by the WMS to submit jobs to them, as illustrated in Figure 3-14.

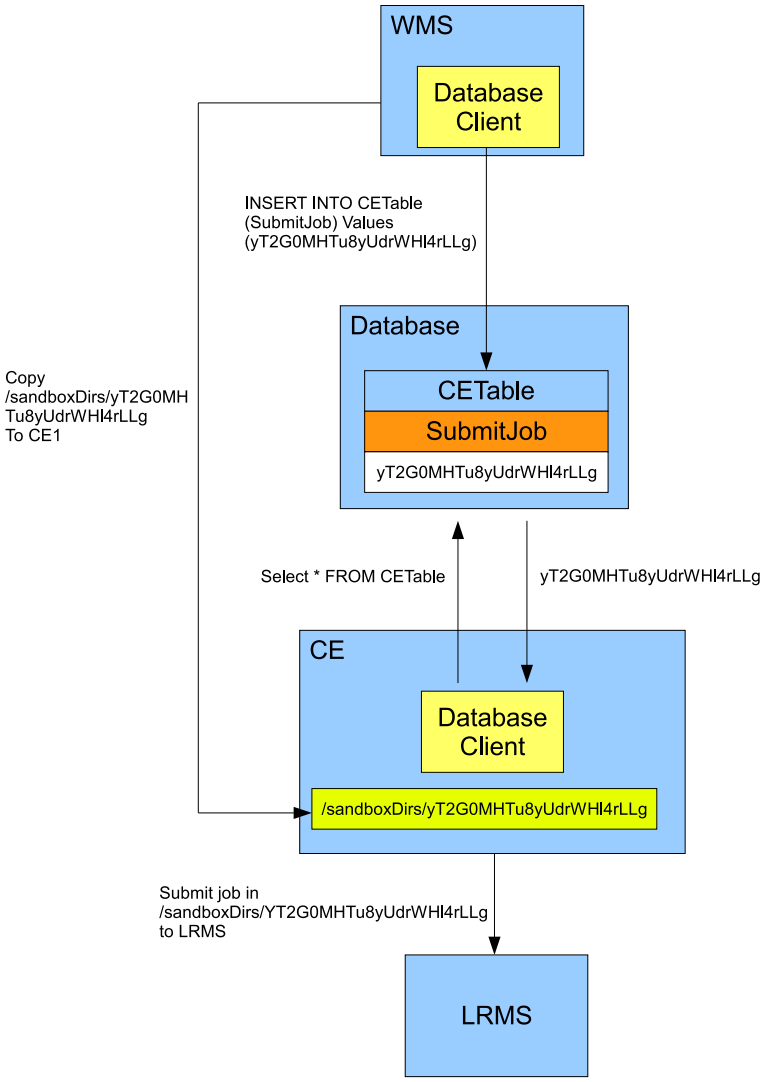


Figure 3-14: Remotely invoking jobs on CEs using a database

The Infogrid CE authenticates and authorises insertions to these tables, which represent job submissions to local resources. The database's security mechanisms are used to perform these operations. If authentication and authorisation are successful, the Infogrid CE copies the sandbox directory that is named in the SubmitJob column in

the table row from the WMS to the CE, and submits the job to the LRMS.

3.5.8 Logging and Bookkeeping System

The Logging and Bookkeeping (L&B) System records events related to the submission and execution of jobs on the Grid. It provides a centralised view of events occurring on resources distributed across the Grid, as illustrated in Figure 3-15.

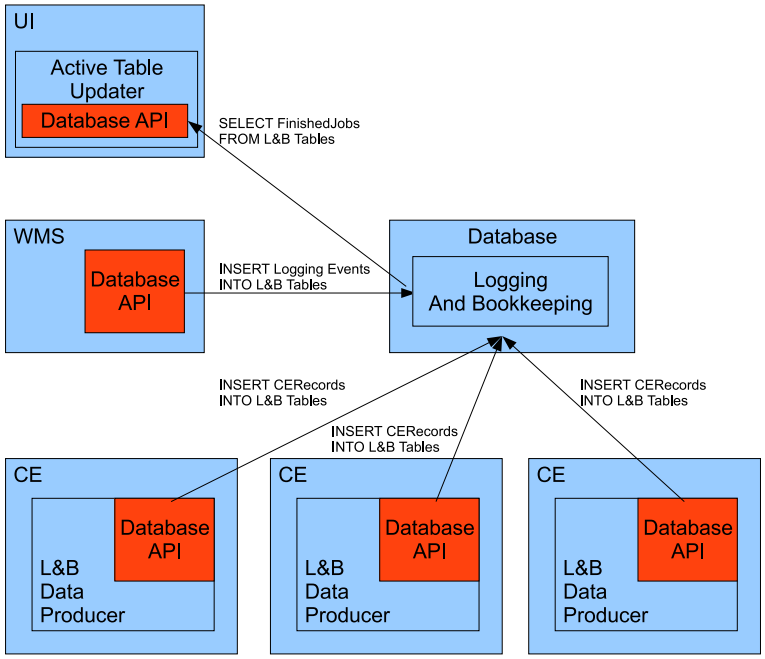


Figure 3-15: L&B system providing centralised view of events across the Grid

The information retained by the logging system can be used for a variety of purposes, such as monitoring the progress of jobs currently executing on the Grid, troubleshooting jobs that have failed to execute correctly, auditing activity on the Grid for security purposes, etc.

3.6 Security and Fault tolerance in Grids

Grids may be intended for use by a restricted group of people. It can be necessary to authenticate and authorise users before allowing them to perform operations using Grid middleware. It is also desirable that actions taken by users on the Grid are

non-repudiatable. In the case of potential commercial Grid infrastructures which in future may charge users for submitting jobs, it is essential that the allocation of resources to jobs be recognised as being a result of a particular user submitting that job. Also, in cases where Grid middleware is used maliciously, it will be desirable for administrators to establish which users are responsible.

Users must not be able to insert rows into or read the contents of the tables in the database technology used to implement Infogrid that they are not authorised to access. Infogrid will use the security mechanisms of the database technology used to implement it to authenticate and authorise users before allowing them to perform such operations on tables.

Authentication is the process of the Grid middleware identifying who a user is. Authentication can be performed using a Public Key Infrastructure (PKI) [56]. For example, in the case of the gLite middleware, each user of the Grid possesses an X.509 [57] certificate, issued by a certificate authority. In order to authenticate themselves to the Grid infrastructure, users create "proxy certificates". These proxy certificates are certificates which are credentials signed using the user's private key, and which have a limited validity period (for example, 12 hours). The proxy certificates can be used in place of the credentials issued by the certification authority. Because these proxy certificates have a limited lifespan, the potential for malicious abuse of Grid facilities in the case of these credentials being stolen is reduced. An example of software used to implement authentication in the gLite middleware is the Globus Grid Security Infrastructure (GSI) [58]. Most Grid middleware now conforms to the authentication profiles of the International Grid Trust Federation (IGTF) [59], a federation of three zonal policy management authorities, EUGridPMA [60] for Europe, TAGPMA [61] for the Americas and APPMA [62] for Asia-Pacific.

Authorisation assumes that a user has been authenticated. Authorisation determines what actions that user is allowed to perform on the Grid, and what resources they are allowed to perform those actions on. In gLite, authorisation is based on a user's membership of a virtual organisation (VO). Members of a VO can submit jobs to the Grid which make use of resources (e.g. computing elements, storage elements)

which are also in that VO. Ideally, Infogrid would support authorisation down to the level of individual table rows.

Grid middleware provides a degree of fault tolerance through replicating important components in order to avoid single points of failure. Grid middleware also performs load balancing so that the resources of the Grid are used optimally, and particular resources are not overloaded. The database technology used to implement Infogrid will also require fault tolerance mechanisms that provide replication mechanisms for important components, ensuring that if a component of the database malfunctions, a replica can continue in its place.

3.7 Conclusion

In this chapter, an architecture for the Infogrid middleware which used a single database technology to perform functions which existing Grid middlewares use multiple applications to perform was proposed. Different jobs require different forms of input (e.g. command line, file, large dataset, unstructured data, etc.), and alternative job submission mechanisms that Infogrid will offer in addition to standard job submission methods to cater for these input data were described. The importance of the security and robustness of Grid middleware was also emphasised.

Chapter 4

InfoGrid Implementation

4.1 Introduction

This chapter describes the implementation of the prototype Infogrid middleware. It begins with a discussion of the various database technologies considered for this implementation, and follows with a detailed description of the database technology chosen, the Relational Grid Monitoring Architecture (R-GMA) [63]. The implementation of the Infogrid prototype is then described, with information on how the components of the Infogrid architecture described in the previous chapter were realised.

4.2 Choosing technologies for implementing Infogrid

When choosing a database technology for implementing Infogrid, the following criteria were considered.

- Is this database technology freely available, at little or no cost?
- Support for publish-subscribe communication? Infogrid must issue "continuous queries" on tables which serve as interfaces to Grid components by "subscribing" to these tables. Does this database allow continuous queries to be issued on streams of data by supporting publish-subscribe communication patterns?

- Provides replication for load balancing and fault tolerance?
- Supports authorisation down to the individual table row level?

The table below illustrates how a number of database technologies that were considered for implementing the Infogrid prototype met the above criteria.

Database Technology	Cost	Publish-Subscribe	Replication	Row level authorisation
Oracle	High	Yes	Yes	Yes
DB2	High	Yes	Yes	Yes
SQLServer	High	Yes	Yes	No
MySQL	Free	No	No	No
PostgreSQL	Free	No	Yes	No
RGMA	Free	Yes	Yes	Yes

Table 4.1: Comparison of candidate database technologies for Infogrid

Publish-subscribe communication patterns enable streams of data to be represented as SQL SELECT statements. In a publish-subscribe system, senders label each message with the name of a topic ("publish"), rather than specifying a specific destination for the message. The messaging infrastructure is responsible for ensuring that the message is forwarded to all entities that have asked to receive messages on that topic ("subscribe"). Neither the publisher nor the subscriber need to be aware of the location of the other. Publish/subscribe is a very loosely coupled architecture, in which senders often do not know who their subscribers are [64]. Processes in the Infogrid prototype which issue continuous SELECT queries on tables in the database will do so by "subscribing" to these tables. Oracle, IBM's DB2 and Microsoft SQLServer are well known commercial RDBMSs which support publish-subscribe communication patterns, which are essential for the Infogrid prototype. However, these commercial systems are expensive to acquire, with large licencing fees required for their use. Open source options which can be obtained for free include MySQL and PostgreSQL, but neither offer support for publish-subscribe communication as standard.

An Infogrid prototype could be realised by using any of the database technologies listed above. In order to do this, it would be necessary to write interfaces to Grid components using the APIs provided by these database systems, and to modify existing Grid components so that they interact with each other via these APIs. However, excepting R-GMA, there is no guarantee that this is possible. Software and library clashes may result in either the Grid software or the RDBMS not functioning. Even if this issue does not arise initially, there is a possibility that at any time such a compatibility issue will arise as database technologies such as Oracle, PostGRES, DB2, etc., are developed separately from Grid middlewares. In addition, re-implementing Grid components such as an information system, or a logging infrastructure for the Grid using database technologies which are not already used to perform these functions in a Grid middleware is a difficult task.

R-GMA stands out as an ideal choice for the database technology used to implement the Infogrid prototype. R-GMA was originally designed as a monitoring system for the gLite Grid middleware. It provides a relational view of data distributed across a virtual organisation. R-GMA supports publish-subscribe communication, and is part of the standard gLite middleware. It is available for free, and there are none of the potential incompatibility problems associated with its use as a database technology in an Infogrid prototype constructed using the gLite middleware that may be encountered if other technologies, which are not part of the gLite middleware, are used. R-GMA has also been developed with security mechanisms based on X.509 certificates for authorising and authenticating users, and can be used to implement a secure interface to Grid components. R-GMA's security mechanisms enable row-level authorisation (unlike some other database technologies considered, which do not provide this feature as standard). R-GMA's security features are also compliant with the authentication profiles specified by the IGTF [59] (the International Grid Trust Federation, which was briefly mentioned in Section 3.6). R-GMA also does not enforce database ACID properties, in order to enhance its performance as a distributed information system. For these reasons, a combination of R-GMA for the database technology and gLite for the Grid middleware has been chosen to implement the In-

fogrid prototype. Indeed it is notable that R-GMA is the only database technology that Infogrid could be easily built from. And since the implementation of R-GMA's full specification (especially its security model) was only completed in late 2008, it is reasonable to infer that Infogrid is a research concept that can only now be easily realised in a prototype form and further explored.

4.3 R-GMA

R-GMA was initially developed as part of the European Datagrid (EDG) project [65]. It resembles a "virtual database", but without enforcement of the traditional ACID properties [66]. Users insert data into and retrieve data from this virtual database by using SQL statements such as INSERT and SELECT. R-GMA is a relational implementation of the Grid Monitoring Architecture (GMA) [67], which was conceived by the Global Grid Forum (now the Open Grid Forum [68]) as an architecture for implementing a monitoring system that provided a real-time view of the status of resources on the Grid. From the perspective of users querying the R-GMA database it seems that they are performing operations on a single, standalone database, but the data is generated and stored at a number of locations across the Grid. Figure 4-1, which is based on a diagram from the architecture/design section of the R-GMA website [5], illustrates the main components of R-GMA. The Schema component contains the definitions of tables in the virtual database. It defines the columns in the tables, and the datatypes associated with those columns. It also defines the authorization rights for these tables. The Schema can be used to add or remove tables from the virtual database. Figure 4-1 shows the R-GMA API being used to contact the Schema to create the CEInfo table. The CEInfo table in Figure 4-1 contains a number of tuples that have been inserted by Producers (labelled Producer 1, Producer 2, and Producer 3) using SQL INSERT statements. Each Producer is highlighted in a different colour, and the tuple in the CEInfo table inserted by each Producer has the same colour.

The Registry component performs a similar role to the logical-to-physical index of LFC[46] or RLS[35], by matching requests for data to producers of that data.

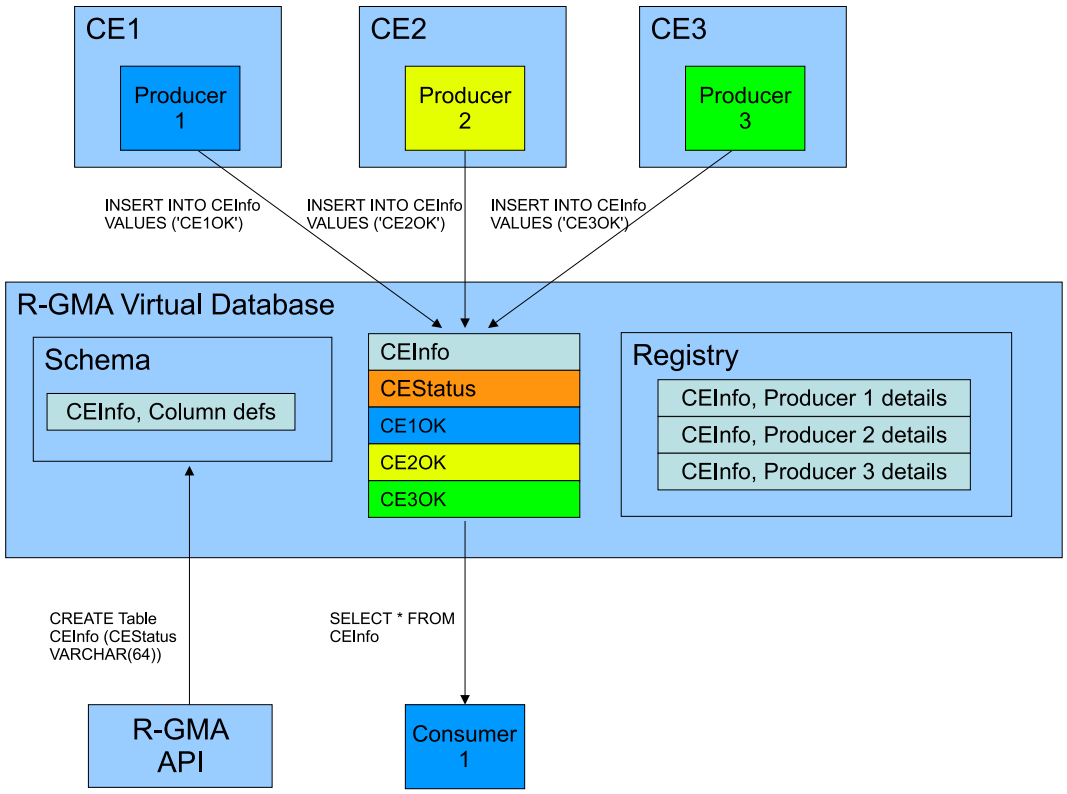


Figure 4-1: Components of R-GMA (based on diagram from R-GMA website [5]).

Producers contact the Registry to advertise what tables in the virtual database they publish rows to, and provide details of their physical location.

Producers can be instantiated using the R-GMA API as illustrated in Figure 4-2, which is based on a similar diagram from the architecture/design section of the R-GMA website [6]. The Grid Component uses the R-GMA API to instantiate a Producer and insert tuples into its tuple storage. Consumers can contact this Producer to retrieve data from it in response to queries. Rows in R-GMA have a retention period associated with them. A retention period is set for a Producer when it is instantiated. It indicates how long a Producer should keep tuples in its storage. The Producers periodically purge the tuples in their storage, by deleting tuples that have exceeded the retention period. This mechanism ensures that the data stored by Producers does not grow to an unmanageable size, and that data is stored only for as long as it is needed.

Consumers can request tuples from R-GMA that satisfy an SQL query. The

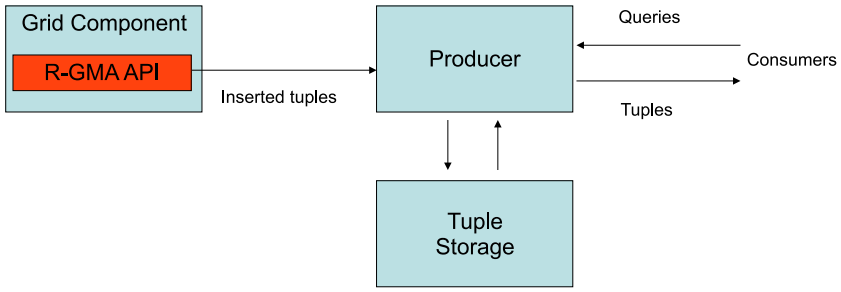


Figure 4-2: R-GMA API - Producer (based on diagram from R-GMA website [6]).

Registry component of R-GMA matches queries for information from Consumers to Producers that provide this information. The Consumer then contacts these Producers to obtain the information. This process, called mediation, allows a Consumer to have a global view of information from sources distributed across the Grid, and enables Consumers to obtain data from Producers without having to know their physical location. R-GMA's mediation feature allows a publish-subscribe messaging system to be implemented that enables data that is inserted into a table by Producers to be automatically streamed to Consumers that have subscribed to that table by issuing SQL SELECT queries on it. Figure 4-3, which like the previous two diagrams is based on a diagram from the architecture/design section of the R-GMA website [7], illustrates the process by which the R-GMA API is used to request that a Consumer consult the Registry in order to find Producers that provide particular information, and obtain this information from Producers.

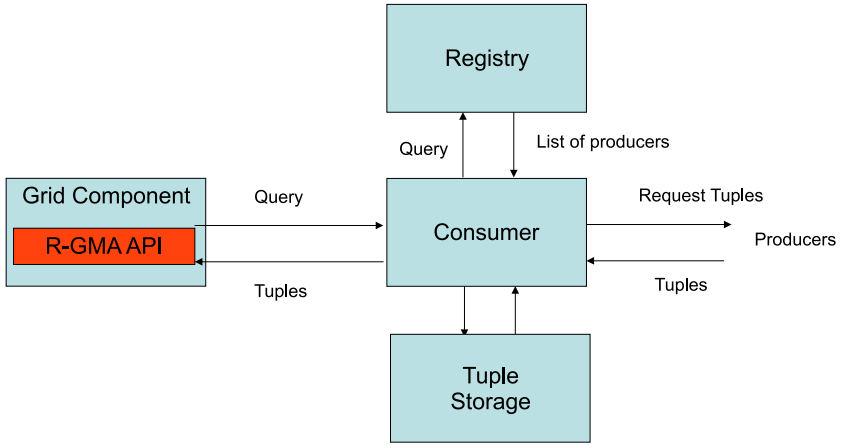


Figure 4-3: R-GMA API - Consumer (based on diagram from R-GMA website [7]).

Figure 4-4 gives an example of a global view offered by R-GMA on data that is distributed across multiple sites on a Grid. Several Producers have published tuples to the CEDetails table. The Registry provides a view across all these Producers, and mediates between the Producers and Consumers so that when Consumers query the CEDetails table, it appears that they are querying a single table.

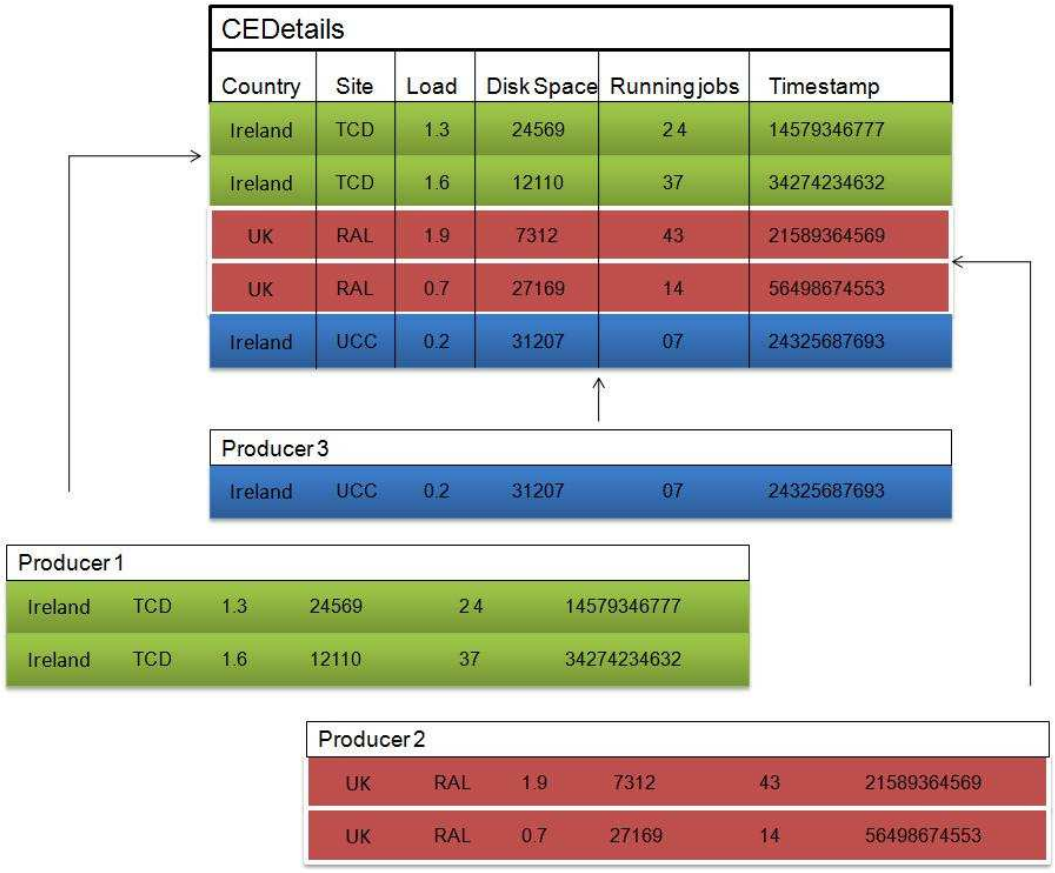


Figure 4-4: R-GMA Schema mediation.

Consumers can issue several types of query. The two query types used by the Infogrid prototype are history queries and continuous queries. A history query returns all tuples that match the query published within a specified time frame (e.g. all tuples published up to twenty minutes before the query being executed). Continuous queries run over a specified period of time (e.g. listen for tuples that are published to a table for 10 minutes) and retrieve tuples that match the query as they are published to the database within that time period. Although R-GMA Consumers do not offer the functionality offered by other stream query languages (e.g. StreamSQL

[69], Continuous Query Language [70] or the Condensative Stream Query Language [71]), the stream querying capability it does offer (a selection on a single relation) is sufficient for the purpose of implementing the Infogrid prototype.

4.3.1 Security in R-GMA

It is necessary for Grid middleware to provide security mechanisms that restrict access to the Grid to users whose identity has been established and who are authorised to use the Grid resources. As the Infogrid prototype implements interfaces to Grid components as relational tables, it is necessary to allow only authenticated and authorised users to insert data into or retrieve data from these tables. Operations performed by Grid components, such as CEs providing updates on their status to the information system or adding entries to a logging system, must be subject to security checks in order to ensure that the information contained in these systems is genuine and provided by actual Grid components. Using R-GMA to implement interfaces to Grid components and to implement an information system and a logging system for the Grid allows a single security service, that provided by R-GMA, to provide the authentication and authorisation mechanisms required by a number of components of the Infogrid prototype.

R-GMA can impose table and row level security restrictions. In order to perform read or write operations on tables in R-GMA, users must have a valid X.509 certificate. The certificate must have Virtual Organisation Membership Service (VOMS) attributed authorisation extensions, therefore it must be generated using the voms-proxy-init command provided by the Virtual Data Toolkit [42]. For example, a certificate that is used to authenticate a user who wishes to perform R-GMA operations in a virtual organisation called compChem is created by issuing the following command on a machine with the voms-proxy-init tool installed:

```
voms-proxy-init --voms compChem
```

The R-GMA API can be used to specify authorization rules for tables, defining which users are allowed to perform operations on the tables. These rules have the following format:

```
predicate:credentials:action
```

The predicate defines the rows of a table which this rule grants access to, e.g.:

```
SELECT * FROM Experiment
```

The credentials define the set of users that this rule applies to. For instance, the following credential specifies that a rule applies to all users whose X.509 certificate indicates that they are members of the cs.tcd.ie Organizational Unit (OU):

```
OU=cs.tcd.ie
```

The action component specifies what actions users who match the credentials can perform on the subset of rows defined by this rule. These actions can be "Read" (R), "Write" (W), or "Read and Write" (RW). A full example of a rule is as follows:

```
SELECT * FROM Experiment :OU=cs.tcd.ie:R
```

This rule authorizes all R-GMA clients whose certificates identify them as belonging the cs.tcd.ie OU to read all rows in the Experiment table.

4.3.2 Fault Tolerance in R-GMA

The R-GMA Registry and/or Schema may fail (for whatever reason). R-GMA has replication mechanisms for these components [72], so that in the event of either failing, a replica can continue operating in the same manner, with transparent recovery when the failed component returns to health. R-GMA also uses a soft-state registration mechanism to periodically purge its Registry of records of Producers and Consumers which are faulty. Producers and Consumers must periodically send a "heartbeat" signal to the Registry to indicate that they are functioning, regardless of whether or not they are currently performing an operation. If the Registry has not detected any activity from a Producer or a Consumer as a result of a query execution involving it, and it has not received a "heartbeat" signal from that component within a particular window of time, the Registry assumes that Producer/Consumer is no longer functional, and deletes its record.

4.4 The Infogrid Prototype

4.4.1 Active Table Creator Servlet

The Infogrid active table creator is implemented as a Java servlet. The servlet accepts input data entered from a web interface (e.g. an SQL statement defining the table structure, which columns represent input and output, the executable invoked by the active table, etc.), and uses this data to create an active table. This web interface is illustrated in Figure 4-5.

After the user enters values in the text boxes and presses the "Create Active Table" button, an active table is created which has the following characteristics:

- the active table has the structure defined in the CREATE TABLE statement.

Figure 4-5: The active table creator web interface.

The syntax for this CREATE TABLE statement is an extension of standard SQL. Columns in the active table can be defined as being either input or output by using the INPUT and OUTPUT keywords. For example, in order to define the sequenceAlign active table described in section 3.3.1, the following statement would be used:

```
CREATE TABLE sequenceAlign
(SourceSequence VARCHAR(65535) INPUT,
TargetSequence VARCHAR(65535) INPUT, SimilarityScore Double OUTPUT)
```

Before executing the SQL CREATE TABLE statement however, the active table creator modifies it by adding two columns to the statement in addition to those specified by the user. These columns are called *Status* and *JobID*.

The Status column indicates whether data in an active table row contains out-

put from a job which has already executed (these rows will have a value of "Cleared" in the Status column). The JobID column contains the gLite job ID associated with this row.

- the active table processes data in the columns defined as input, using the program defined in the executable text box (at present, for the sake of simplicity, the prototype assumes that the executable is already present in a particular directory on the User Interface machine).

The chunking limit and chunking timeout parameters will be explained in a later section. The active table creator creates active tables in two steps, as illustrated in Figure 4-6.

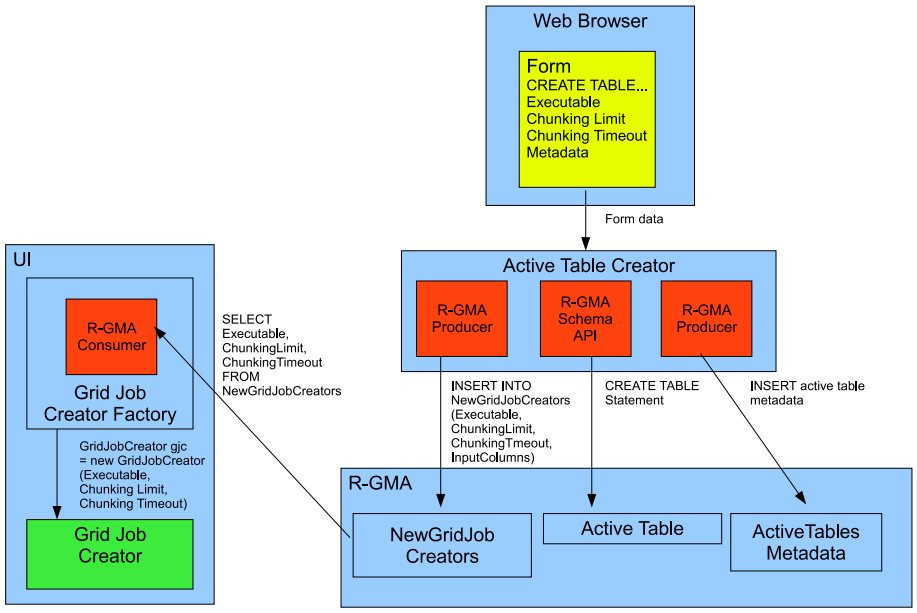


Figure 4-6: Creating an active table using the active table creator.

Firstly, the active table creator uses the R-GMA Schema API in order to add this table to the schema. Then the active table creator inserts the information entered from the web interface into an R-GMA table called NewGridJobCreators. The purpose of this table is to enable the Grid Job Creator Factory, which is a process running on the UI, to obtain this information from the active table creator servlet, which runs on another machine. Upon receiving this information, the Grid Job Creator Factory launches a thread in which a Grid Job Creator Java object is instantiated. This Grid

Job Creator object uses an R-GMA Consumer to issue a continuous query (to await input data) on the newly created active table.

The R-GMA schema used by Infogrid can contain both active tables and regular tables. The active tables metadata table can be used to differentiate between an active table and a regular table in the R-GMA schema. In order for an active table to function properly, there must be rows in the active tables metadata table inserted by the active table creator specifying what columns in that active table represent input and output. If no such rows exist in the active tables metadata table for a particular table in the R-GMA schema, that table does not function as an active table.

4.4.2 Active Tables Metadata

The active tables metadata table contains rows inserted by the active table creator which define what columns in active tables represent input and output. The active tables metadata table has the following structure.

- ActiveTable (VARCHAR(255)) : name of active table.
- ColumnName (VARCHAR(255)) : name of column in active table.
- ColumnType (VARCHAR(255)) : this should be set to "Input" if the column represents input, or "Output" if the column represents output

4.4.3 R-GMA User Clients for Infogrid

Java, C++, C and Python APIs have been developed for R-GMA, which provide methods for performing operations on the R-GMA virtual database. These APIs send data securely via HTTPS connections, performing authentication and authorisation as required. User client programs can be written that submit data to the Infogrid prototype by using the R-GMA API to instantiate a Producer which inserts data into active tables, as illustrated in Figure 4-7.

This client program uses an R-GMA Producer to insert a pair of gene sequences into a sequenceAlign active table, in order to generate a similarity score for these

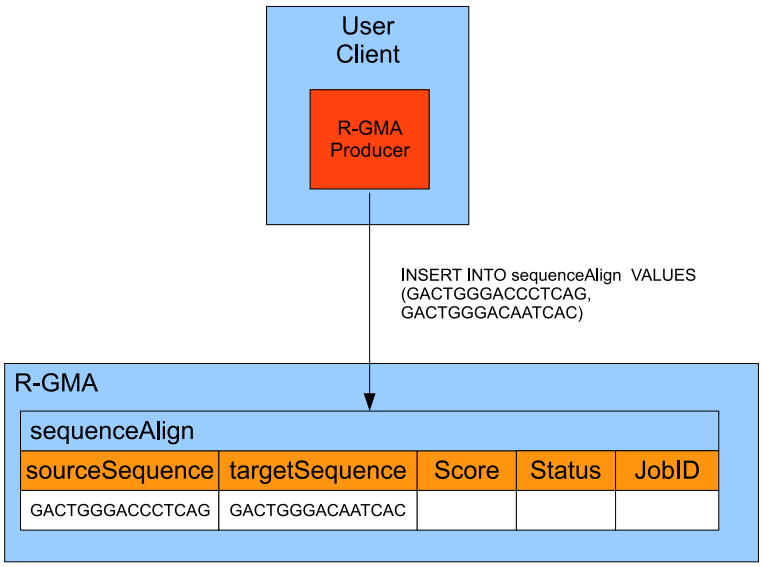


Figure 4-7: A user client program inserting data into an active table using R-GMA

sequences. The following code illustrates how the R-GMA Java API is used to instantiate a Producer which inserts data into an active table.

```
// Instantiate a ProducerFactory object, which is used to create
// Producers.
      ProducerFactory factory = new ProducerFactoryStub();

// The TimeInterval object will represent the length of time the
// Registry will keep a record of the Producer.
      TimeInterval ti = new TimeInterval(60, Units.MINUTES);

// Specify the properties of the Producer:  this Producer uses memory
// for tuple storage.
      ProducerProperties props =
         new ProducerProperties(Storage.MEMORY, 0);

// Create a PrimaryProducer using the ProducerFactory
      PrimaryProducer pp = factory.createPrimaryProducer(ti, props);
```

```
// An empty predicate String, which is used for Producers which can
// insert rows into a table which can contain any value.
      String predicate = "";

// create two additional TimeInterval objects, which represent the
// retention periods for history and latest tuples for the Producer.
      TimeInterval historyRP = new TimeInterval(60, Units.MINUTES);
      TimeInterval latestRP = new TimeInterval(60, Units.MINUTES);

// Declare the table that this Producer inserts rows into
      pp.declareTable("sequenceAlign", predicate, historyRP, latestRP);

// Declare string containing SQL that inserts data into the
// sequenceAlign active table.
      String insert = "INSERT INTO sequenceAlign
         (SourceSequence, TargetSequence)
         VALUES ('GTGGGCCATGTAG', 'ATGGGACATGTAG')";

// Execute the SQL string
      pp.insert(insert);
```

R-GMA also provides a command line tool for executing operations, in addition to APIs. The user types "rgma" at the command line in a Unix terminal on a machine with the R-GMA client software installed to start the R-GMA command line tool. A series of statements are displayed indicating the location of the R-GMA server which hosts the R-GMA services, and Uniform Resource Locators (URLs) for the Registry and Schema services. Figure 4-8 shows a screenshot where a dataset is submitted to Infogrid for processing using this command line environment.

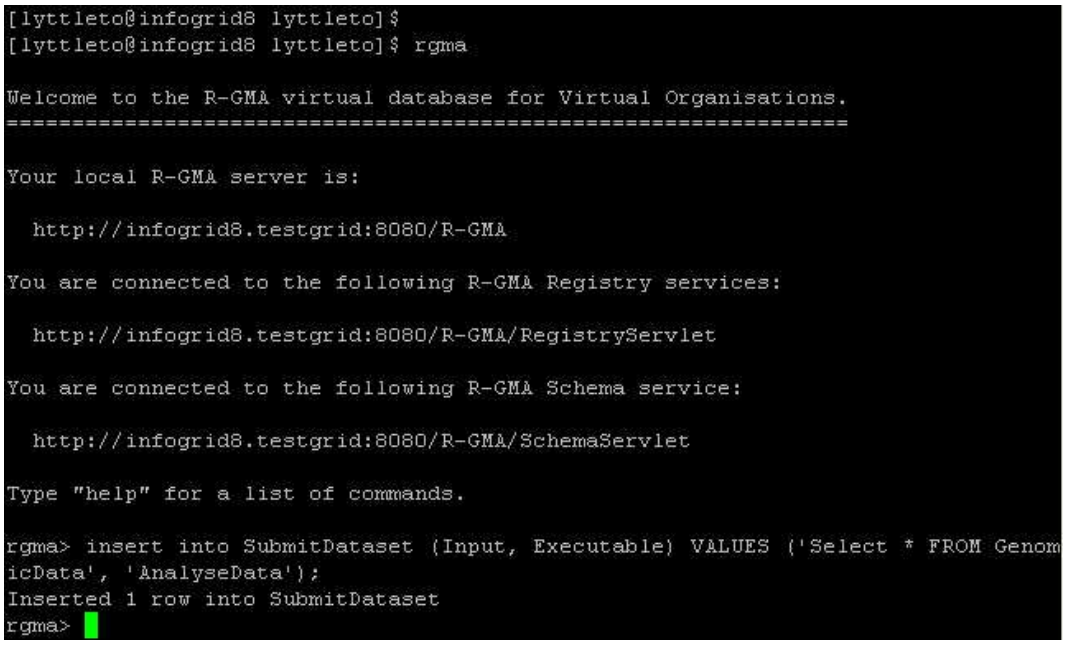


Figure 4-8: Using the R-GMA command line client to submit a job to Infogrid

In Figure 4-8, an SQL statement is executed which inserts a row into the SubmitDataset table, requesting that a job which executes the AnalyseData application be submitted to the Grid, using the dataset returned by the query "SELECT * FROM GenomicData" as input.

Job submission using the standard command line tools to the Infogrid prototype is also still possible. Users may wish to use these tools for a number of reasons. They may not wish to specify job input as tables in a database, they may not wish the input and/or output from their jobs to be visible to other members of the VO, or they may want to execute a job on the Grid which cannot be executed using an active table or by inserting tuples into the SubmitDataset/SubmitLargeDataset tables. With the Infogrid prototype, the standard tools provided by the gLite middleware for submitting jobs, querying the status of jobs, and retrieving the output of completed jobs work as they normally do.

4.4.4 Infogrid Grid Job Creator

Grid Job Creators submit jobs to the Grid in response to insertions into R-GMA tables which serve as job submission interfaces. There are three types of Grid Job

Creators, one which processes data from active tables, one which processes data inserted into the SubmitDataset table, and one which processes data inserted into the SubmitLargeDataset table.

Grid Job Creator for Active Tables

Each active table has its own Grid Job Creator process, which issues a continuous query on that active table. The Grid Job Creator constructs executable scripts that invoke the executable associated with an active table for each row of inputs inserted into that table, using the data in that row as command line arguments (e.g. "sequenceAlign GACTGGGACCCTCAG GACTGGGACCATCAC"). The Grid Job Creator is coded in Java, and uses the R-GMA Java API to instantiate Consumers and issue queries on the active table, as shown in Figure 4-9.

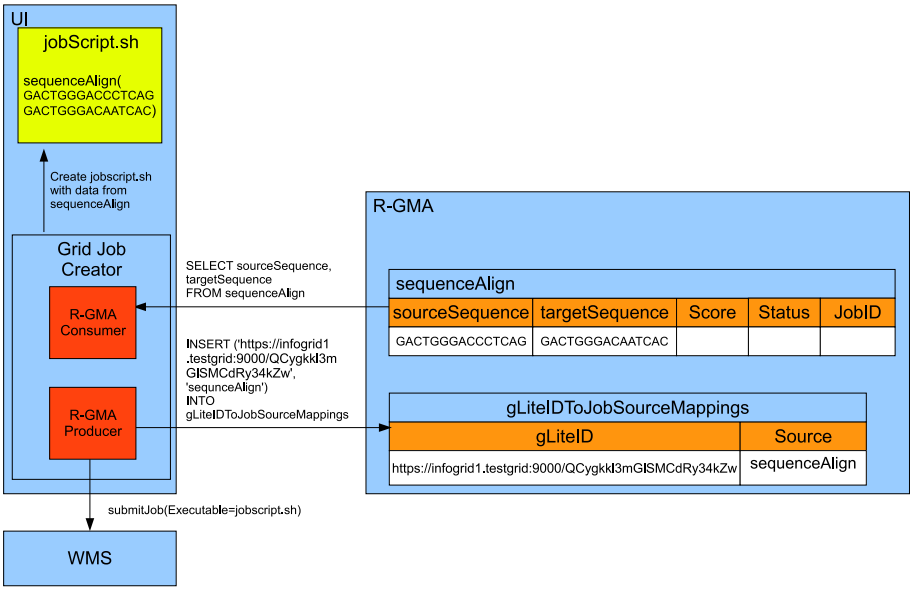


Figure 4-9: Infogrid Grid Job Creator for an active table

As data is inserted into the sequenceAlign active table by a User Client, it is retrieved by the Grid Job Creator that is issuing a continuous query on that table. Assuming that the functions that tables represent are deterministic, if a tuple is inserted with particular input values that are already present in the active table, there may be no need to perform the function again for this set of inputs, as the output for these inputs is already available or is in the process of being generated. This process of

using cached results is called memoisation. For each row inserted into an active table, the active table's Grid Job Creator will count how many tuples are in the active table which have the same input values. If the result is zero, these inputs are not already in the active table, and processing of that tuple continues as normal. If these inputs are present in the active table however, the Grid Job Creator will disregard the row.

If the running time for the function performed by an active table is short, it may be desirable to submit a single job to the Grid which calculates outputs for multiple rows in the active table. The Grid Job Creator can create scripts which iterate over input data from multiple rows, and invoke an executable using each of these inputs as command line arguments. The process by which the Grid Job Creator creates an executable script from a number of rows in an active table is called chunking. When creating an active table, it is possible to specify how many rows from the active table are submitted to the Grid in each job. This value, briefly mentioned when the interface for creating an active table was described, is called the chunking limit. Each Grid Job Creator counts how many rows have been inserted into an active table since the last time it submitted a job. When a new input is added to an executable script, this count is incremented. When the count equals the chunking limit, the Grid Job Creator submits this script to the WMS, which will find a resource on the Grid for executing it. The count is then reset to zero, and the process repeats. Grid Job Creators that process data from active tables also have chunking timeouts associated with them. If a Grid Job Creator has only partially created a script that processes a chunk of inputs, if there are no further insertions within the timeout period, the script is completed and submitted to the Grid. The use of chunking is illustrated later. The following code illustrates how the Grid Job Creator uses the R-GMA API.

```
//instantiate a ConsumerFactory object, which is used to create
//Consumers.
      ConsumerFactory factory = new ConsumerFactoryStub();

//The TimeInterval object will represent the length of time the
//Registry will keep a record of the Consumer.
      TimeInterval ti = new TimeInterval(1000, Units.DAYS);

//instantiate a Consumer object, which issues a continuous query on a
//table, selecting the data in input columns.
      Consumer c = factory.createConsumer(ti, "SELECT "+inputColumns+
      " FROM "+table+", QueryProperties.CONTINUOUS);
      c.start(new TimeInterval(1000, Units.DAYS));
      ResultSet rs=null;

//starts a while loop, which loops while the continuous query runs
      while (c.isExecuting()) {

//decrement a timer which tracks how long has passed since the last
//job submitted by this Grid Job Creator.
         currentTimeout--;
         if (currentTimeout==0)
            {
            finishScript();
            submitJob();
            resetTimeout();
            }
         rs = c.popAll();
```

```
//a do-while loop iterates through the rows that are contained
//in the ResultSet object, and executes while the isAfterLast method of
//the ResultSet object is false.
        do
           {
              if (rs.size() !=0)
                 {
                 //invokes the memoisation function, which returns
                 //true if a row with the same input values is not
                 //currently in the active table.
                 if (memoisation(rs))
                    {
                    writeToScript(rs);
                    chunkCount++;
                    }
                 //checks if the chunking limit for the active table
                 //has been reached as a result of the invocation of
                 //the writeToScript function
                 if (chunkCount==chunkLimit)
                    {
                    finishScript();
                    submitJob();
                    resetChunkCount();
                    resetTimeout();
                    }
                 }
           rs.next();
           } while (!rs.isAfterLast());
     }
```

As an example, consider the following sequence of insertions to an active table called sequenceAlign, which has been defined as having a chunking limit of 3 and has input columns called sourceSequence and targetSequence.

```
INSERT INTO sequenceAlign (sourceSequence, targetSequence)
   values ('GACTGGGACCCTCAG','GACTGGGACAATCAC')

INSERT INTO sequenceAlign (sourceSequence, targetSequence)
   values ('AACTGGTTGCCTCAA','GACTGGACAATCAC')

INSERT INTO sequenceAlign (sourceSequence, targetSequence)
   values ('GAATAGGACACTCAG', 'GACTGGGACAATCAC')
```

The Grid Job Creator for the sequenceAlign active table builds an executable script called sequenceAlignJobScript.sh that invokes a sequence alignment application, clustalw, using the data from these insertions as input. After the first insertion into the sequenceAlign table, the Grid Job Creator will perform a memoisation check to see if there is already a row in the table with the same inputs, similar to the following:

```
SELECT count (*) FROM sequenceAlign WHERE
   sourceSequence='GACTGGGACCCTCAG' AND
   targetSequence='GACTGGGACAATCAC'
```

If the result returned by this query is zero, these input values have not already

been calculated by the active table, and will be added to an array in the executable script, which will then look like this...

```
input={"GACTGGGACCCTCAG GACTGGGACAATCAC",
```

After the second insertion, assuming that the memoisation process does not find an existing row in the table with the same inputs, the script will look like this...

```
input={"GACTGGGACCCTCAG GACTGGGACAATCAC",
      "AACTGGTTGCCTCAA GACTGGACAATCAC",
```

...and so on until the chunking limit has been reached. At this point a "for" loop is inserted into the script, which invokes (on the Grid worker node that executes the script) the executable associated with the active table (clustalw) once with each element in the array as an argument. As the chunking limit for the sequenceAlign table is 3, the executable script will look as below after the third set of inputs is added to it:

```
input={"GACTGGGACCCTCAG GACTGGGACAATCAC",
      "AACTGGTTGCCTCAA GACTGGACAATCAC",
      "GAATAGGACACTCAG GACTGGGACAATCAC"}
for i in `seq 0 3`
do
   clustalw input[i];
done
```

When the script is ready for submission to the Grid, the Grid Job Creator invokes methods in the gLite WMS API to create a job which specifies "sequenceAlignJobScript.sh" as an executable, and submits this job to the WMS. The WMS finds a resource on the Grid that will execute this script, and returns a job ID that identifies the job that is executing this particular script. The Grid Job Creator also inserts a row into a table called gLiteIDToJobSourceMappings in order to indicate the active table this job is processing data for. This information is required by the Table Updater component upon the completion of a job, as will be seen later.

The rows that are initially inserted into an active table may have a short retention period relative to the execution time for the function performed by that active table. By the time a job has completed the input rows may no longer be present in the active table. Renaming the executable script so that its title includes the unique portion of the the job ID identifying the job executing that script allows the input data to persist until it is required by the Table Updater component. For example, if the job ID is
"https://node1.cs.tcd.ie:9000/yT2G0MHTu8yUdrWHl4rLLg", the script is renamed to "yT2G0MHTu8yUdrWHl4rLLg.sh". This script will be used by the Table Updater component to obtain the input for the job when it has completed.

Grid Job Creator for SubmitDataset table

The SubmitDataset table has its own dedicated Grid Job Creator. It is coded in Java, and uses the R-GMA Java API to instantiate Consumers and issue queries on tables. The SubmitDataset Grid Job Creator uses an R-GMA Consumer to issue a continuous query on the SubmitDataset table. This query retrieves the values of the Input and Executable columns for every row inserted into this table.

In Figure 4-10, a row has been inserted into the SubmitDataset table which has "AnalyseData" as the value for the Executable column. The SQL SELECT statement in the input column of this row defines the input dataset for the job ("SELECT * FROM GenomicData"). R-GMA provides methods for obtaining metadata such

as column datatypes for resultsets. The column types of the resultset obtained when the SQL statement in the input column is executed are obtained in this manner. In this example, the resultset obtained when the SQL query "SELECT * FROM GenomicData" is executed contains a single column containing data of type INTEGER. The Grid Job Creator then searches the InfogridApplications table for a row which has the value "AnalyseData" in the Executable column. When this row is found, the Grid Job Creator looks at the value of the Input column in that row, which specifies what datatypes the input for that executable must be. In Figure 4-10, the InfogridApplications table specifies that the AnalyseData executable accepts as input a set of tuples which contain a single Integer. If the input dataset has incorrectly typed data or the wrong number of columns, the SubmitDataset table is updated so that the Status column for that job contains an error message (as is the case in the second row in the SubmitDataset table in Figure 4-10). If the SELECT statement in the row from the SubmitDataset table specifies a valid input dataset, the Grid Job Creator executes this statement, and writes the dataset returned to a uniquely named file (GenomicData1 in Figure 4-10). A job is then submitted to the Grid which uses the application in the Executable column of the row from the SubmitDataset table as an executable, and the file prepared by the Grid Job Creator as input. As with the Grid Job Creator for an active table, a row is inserted into the gLiteIDToJobSourceMappings table in order to indicate the application this job is processing data for.

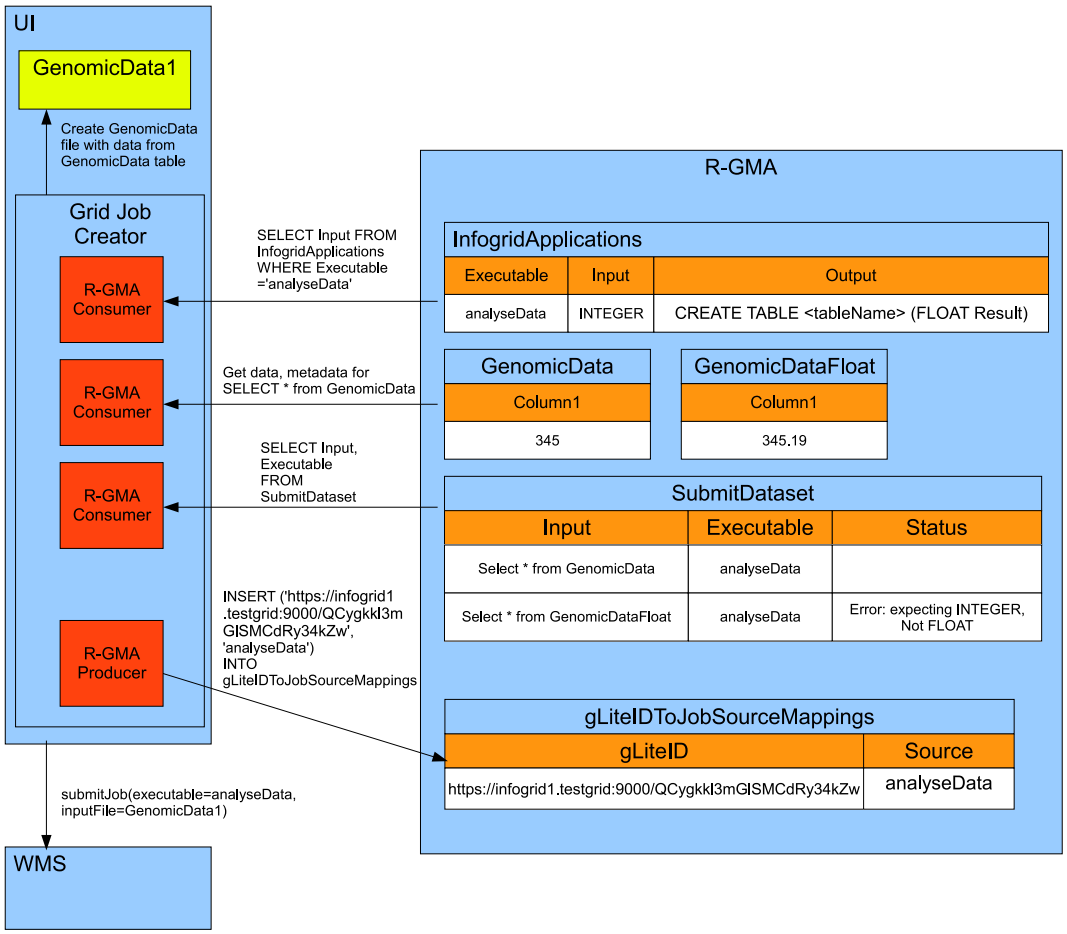


Figure 4-10: R-GMA Grid Job Creator for SubmitDataset table

Grid Job Creator for SubmitLargeDataset table

The Grid Job Creator for the SubmitLargeDataset table takes two values from each row inserted into the table, a LFN which identifies the dataset that is used as input for a job and the executable that processes that input, as shown in Figure 4-11.

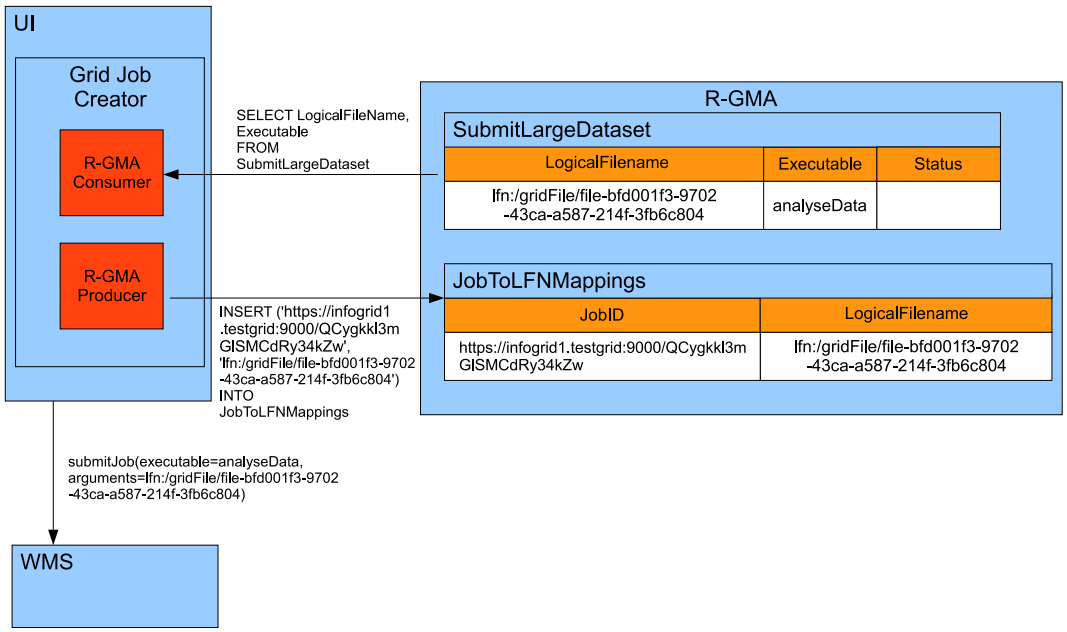


Figure 4-11: R-GMA Grid Job Creator for SubmitLargeDataset table

Before a row is inserted into this table, the input dataset must be located on a storage element, and be registered with the LHC File Catalog (LFC). The Grid Job Creator generates a JDL file using the LFN and the executable name similar to that shown below. The italicised sections of the JDL file indicate the values for the LFN and executable that are generated using the values inserted into the SubmitLargeDataset table.

```
[ Executable = "/bin/sh";
Arguments = "executeJob.sh
   lfn:/gridFile/file-bfd001f3-9702-43ca-a587-214f-3fb6c804
   analyseData ";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = "executeJob.sh";
OutputSandbox = {"std.out","std.err"};
DataRequirements =
```

```
{
   InputData =
      {"lfn:/gridFile/file-bfd001f3-9702-43ca-a587-214f-3fb6c804"};
   DataCatalogType = "lfc";
]
};
DataAccessProtocol = {"rfio","gsiftp"};
]
```

This job executes a script, executeJob.sh, which takes two input arguments, the LFN and the executable. The executeJob.sh script defined as an argument in the JDL is shown below. It defines variables required for the Grid middleware to use the LFC (LFC_HOST, LCG_GFAL_INFOSYS, and LCG_CATALOG_TYPE), uses the lcg-cp command to transfer the dataset identified by the LFN (the first argument to the script, denoted using $1) to the Grid resource executing the job, and then invokes the executable which is the second argument passed to the script (denoted using $2). The Infogrid prototype assumes that the executable reads in data from the input dataset without requiring command line arguments to be specified. The "Copy and Register" command (lcg-cr) is then used to copy the output of the job (contained in the file "std.out") to an SE, and this file is registered with the LFC, so that users can use the Grid data management tools to retrieve it. When registering the output file with the LFC, it is given the same name as the input LFN specified in the row inserted into the SubmitLargeDataset table, but with "_OUTPUT" appended.

```
#!/bin/sh
# Set the proper environment
export LFC_HOST=cagraidsvr22.cs.tcd.ie
export LCG_GFAL_INFOSYS=rgma-bdii.sa3.testgrid:2170
export LCG_CATALOG_TYPE=lfc

# Download the file from the SE to the worker node (WN) where this
job runs
# note that the LFN is passed as input to this script
lcg-cp --vo gitest $1 file:``pwd``/local_file
$2
lcg-cr --vo gitest -d se3.sa3.testgrid
   -l $1_OUTPUT file:$PWD/std.out
```

The JDL is submitted to the Grid for processing. When the job executes on the Grid, it can retrieve the large dataset it requires, process the data contained in it, and upload the output to an SE where it can be retrieved using the data management tools provided with the Grid middleware (e.g. the same lcg-cp command as was used in the above script). The Infogrid prototype did not allow users to specify an SE they wish to upload the output of jobs to. The name of the SE in the script where output was transferred to was hard-coded. Future work on the Infogrid prototype could investigate how users could specify the SE they wish output to be transferred to.

The Grid Job Creator for the SubmitLargeDataset table also inserts a row into a table called 'JobToLFNMappings'. This table is required when the Infogrid prototype is inserting rows into the SubmitLargeDataset table indicating that jobs have completed.

4.4.5 Table Updater

The Table Updater is a Java program which runs on the UI that issues a continuous query on a table which contains details of completed jobs, FinishedJobs. This table consists of rows containing a gLite job ID that uniquely identifies the job to the Grid middleware, and the source of this job. The source of this job may be an active table or the executable specified in a row inserted into either the SubmitDataset or SubmitLargeDataset tables. When a row containing a job ID is inserted into the FinishedJobs table, the Table Updater uses the gLite WMS API to retrieve the job output. What the Table Updater does after retrieving the job output depends on what the source of the job was.

Updating Active Tables

If the source of the job was an active table, when the Table Updater consults the active tables metadata table, there will be rows indicating which columns in the active table represent input and which represent output. A shortcoming of R-GMA is that it does not allow row updates, therefore output data from active table jobs must be inserted into active tables in new rows. Figure 4-12 illustrates the Table Updater inserting the output of a job into an active table.

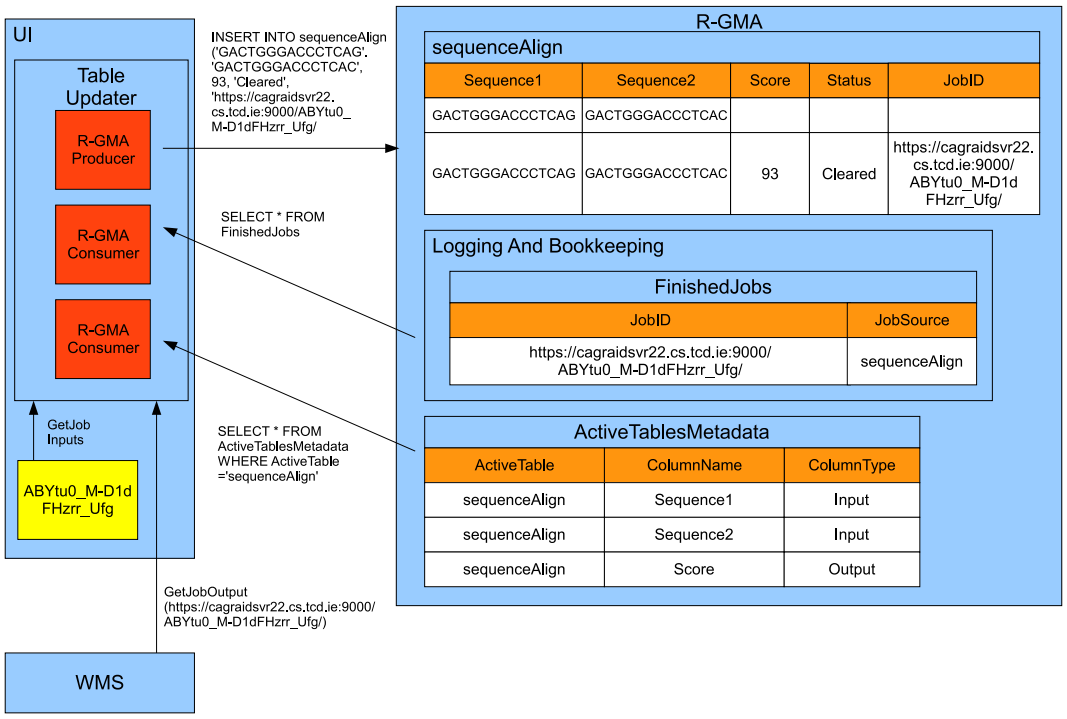


Figure 4-12: Table Updater updating Active Table.

The output from an active table job will be a text file containing the output for one row from the active table per line. The Table Updater reads the data from this file, one line at a time, and also obtains the input data for this job by parsing its executable script (which as mentioned previously, was saved by the Grid Job Creator for that active table). The output on the Nth line of the output file will be the result of applying the function performed by the active table to the Nth element in the input array in the executable script, as illustrated in Figure 4-13.

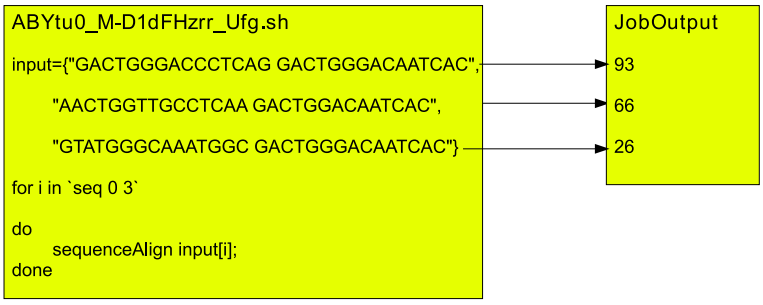


Figure 4-13: Nth element in input array in job script maps to Nth line in output file.

The Table Updater queries the active tables metadata table in order to determine

which columns in the active table are input, and which are output, by issuing an SQL statement such as the following:

```
SELECT ColumnName,Type FROM ActiveTablesMetadata
   WHERE ActiveTable='sequenceAlign'
```

Having done this, the Table Updater can generate and execute SQL statements which will insert rows containing the input and output data into R-GMA. For example, if a row that is inserted initially by a user into an active table is...

```
INSERT INTO sequenceAlign (Sequence1, Sequence2)
   VALUES ('GACTGGGACCCTCAG', 'GACTGGGACAATCAC')
```

...the statement executed by the Table Updater at the stage of job completion will be the following:

```
INSERT INTO sequenceAlign (Sequence1, Sequence2,
   Score, Status, JobID)
   VALUES (
   'CTGGGACCCTCAG', 'GACTGGGACAATCAC', 93, 'Cleared',
      'https://infogrid1.testgrid:9000/tulDGJYyZP7T1RIW92ZJeA');
```

This process continues until all the rows from the output file have been read, and an SQL INSERT statement has been generated and executed for each of these output values.

As mentioned previously, it is a shortcoming of the Infogrid prototype that R-GMA does not allow row updates. As a consequence, new rows containing the output of active table jobs must be created, instead of updating the output columns of the existing rows in the active table. The old rows will persist in the active table until they exceed the retention period of the Producer that inserted them, at which point they will be purged from the Producer's storage and disappear from the table. The screenshot in Figure 4-14 shows the contents of an active table immediately after a job that has processed 3 rows has completed. There are 6 rows in the table: the initial three rows that were inserted, and the three rows containing the output which have been inserted by the Table Updater. The value of the Status column in the last 3 rows is "Cleared" indicating that the job that processed these rows has completed and its output has been retrieved. The chunking limit for this active table has been set to 3, therefore all 3 rows have the same value in the ID column, indicating one job processed all 3 rows.

sequenceAlign				
Sequence1	Sequence2	Score	Status	JobID
GACTGGGACCCTCAG	GACTGGGACAATCAC			
GACTGGGACCCTCAG	CGCAAAGACGAACAC			
GACTGGGACCCTCAG	AAGGGGGCCATTTGA			
GACTGGGACCCTCAG	GACTGGGACAATCAC	93	Cleared	https://cagraidsvr22.cs.tcd.ie:9000/ABYtu0_M-D1dFHzrr_Ufg/
GACTGGGACCCTCAG	CGCAAAGACGAACAC	20	Cleared	https://cagraidsvr22.cs.tcd.ie:9000/ABYtu0_M-D1dFHzrr_Ufg/
GACTGGGACCCTCAG	AAGGGGGCCATTTGA	40	Cleared	https://cagraidsvr22.cs.tcd.ie:9000/ABYtu0_M-D1dFHzrr_Ufg/

Figure 4-14: State of an active table after job completion.

Updating SubmitDataset Table

If there are no entries in the active tables metadata table for the source indicated by a row in the FinishedJobs table, but there is a row in the InfogridApplications table

for an executable with the name of that source, this job is a result of an insertion into the SubmitDataset table. The Table Updater must create a table in the R-GMA schema that will contain the results of this job. In order to do so, it must query the InfogridApplications table to determine what the structure of output tables should be for jobs that invoke this executable. It then creates a uniquely named table with this structure, retrieves the output for this job from the WMS, and inserts the output into the newly created table. Figure 4-15 illustrates this procedure.

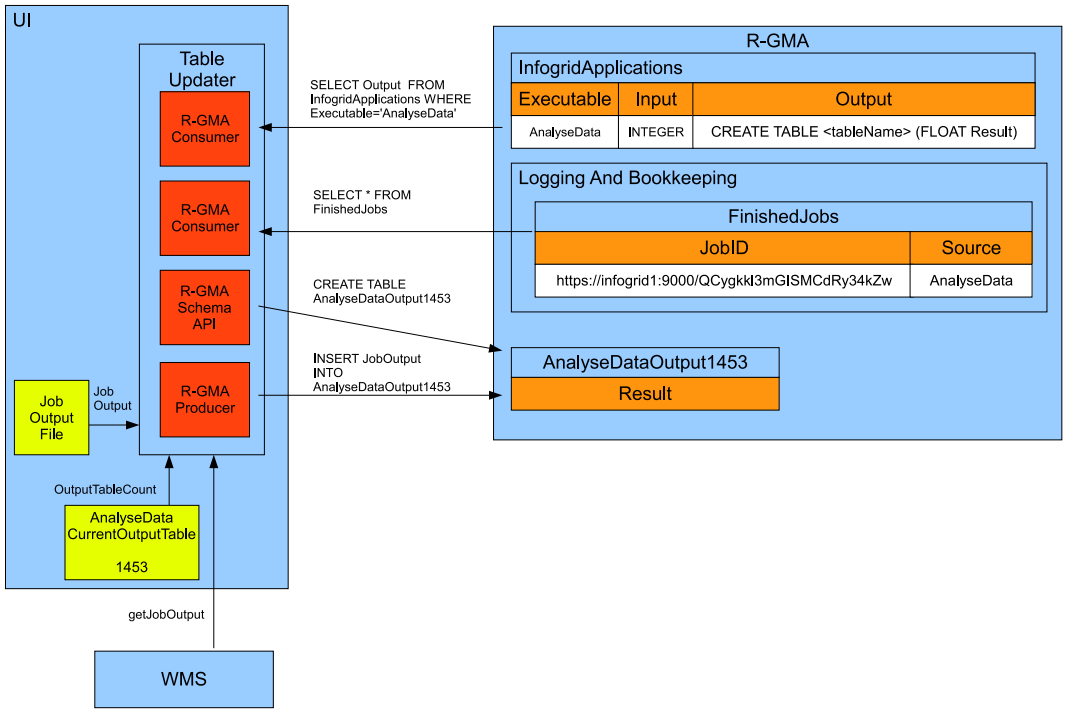


Figure 4-15: Table Updater inserting output of SubmitDataset job into RGMA.

The Table Updater issues a query on the FinishedJobs table using a Consumer. It receives a tuple indicating that a job from the source "AnalyseData" has completed. After failing to find any entries in the active tables metadata table for the AnalyseData executable, another Consumer is used to pose the following query on the InfogridApplications table, in order to determine if this job is a result of an insertion into the SubmitDataset table, and if so, what structure the output table for this executable should have:

```
SELECT Output from InfogridApplications
   WHERE Executable='AnalyseData';
```

If the recently completed job is a result of an insertion into the SubmitDataset table, this query returns an SQL CREATE TABLE statement similar to the following:

```
CREATE TABLE <tableName> (FLOAT Result);
```

Each output table for a job submitted to Infogrid via the SubmitDataset table must have a unique name. The Table Updater generates a unique name for each table by the following process. It first checks if a file which is called <executable>CurrentOutputTable exists in a particular directory on the UI, where executable is the name of the Source column in the row retrieved from the FinishedJobs table, e.g. AnalyseData. This file contains a number which is the next number to use in the name of an output table which will result in a unique name for that table.

The Table Updater opens this file, reads the number that it contains, and uses this number in the name of the output table when creating it. In Figure 4-15, this file contains the number 1453, therefore the output table is called AnalyseDataOutput1453. The Table Updater then increments the number and writes the new number to the <executable>CurrentOutputTable file, overwriting its existing contents. If the file containing the number for the next output table does not exist, this executable has not been invoked using the SubmitDataset table yet, therefore the number 0 is used in the output table name, and the Table Updater creates a count file for that executable containing the number 1.

The CREATE TABLE statement retrieved from the InfogridApplications table is

modified to use this unique table name as shown below, and executed:

```
CREATE TABLE AnalyseDataOutput1453 (FLOAT Result);
```

The Table Updater then retrieves the output of the job identified by the JobID column in the row from FinishedJobs, and reads through the output file, constructing SQL statements which insert the data in each row into the output table in R-GMA. For example, if the first 4 rows of the output file in Figure 4-15 are as follows...

```
45.32
32.12
143.02
87.99
```

...the Table Updater will read each of these rows in, and invoke the following SQL statements:

```
INSERT INTO AnalyseDataOutput1453 VALUES (45.32);
INSERT INTO AnalyseDataOutput1453 VALUES (32.12);
INSERT INTO AnalyseDataOutput1453 VALUES (143.02);
INSERT INTO AnalyseDataOutput1453 VALUES (87.99);
```

For the sake of simplicity, the Infogrid prototype assumes that the output from jobs is stored in a comma-separated-version (CSV) format. After the output table for the job has been created and populated, the Table Updater inserts a row in the SubmitDataset table indicating this job has completed and the name of the table

containing its output (as shown previously in Figure 3-8 in Section 3.3.2).

Updating SubmitLargeDataset Table

If there are no entries in either the active tables metadata table or the InfogridApplications table for the value contained in the "Source" column contained in a row from the FinishedJobs table, the job is a result of an insertion into the SubmitLargeDataset table. The Table Updater must insert a row into the SubmitLargeDataset table which informs users that this job has completed, and its output has been uploaded to an SE, as indicated in Figure 4-16. The SubmitLargeDataset table in Figure 4-16 contains two rows. The first row was inserted by a user who wished to submit a job which invoked the analyseData executable with the file identified by the value in the LFN column as input. The "Status" column for this row is empty. The second row is inserted by the Table Updater when the job has completed. The "Status" column in this row contains the value "Cleared", and the LogicalFilename column of the second row contains the LFN which identifies the output for the job. As mentioned in Section 4.4.4 in the description of the Grid Job Creator for the SubmitLargeDataset table, the LFN for the job output is the same as the LFN for the job input, but with "_OUTPUT" appended.

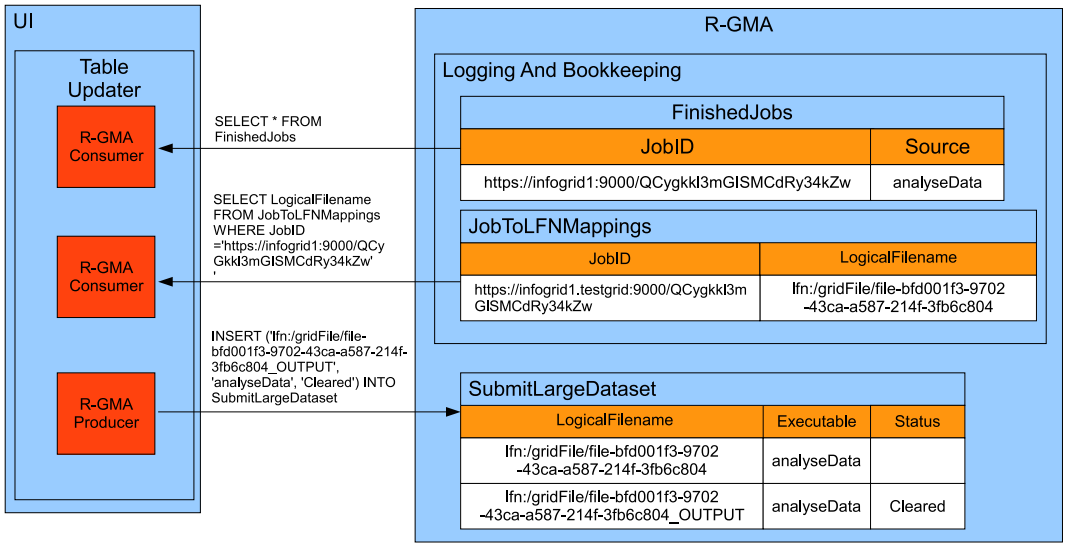


Figure 4-16: Table Updater updating SubmitLargeDataset table.

The user can then use the gLite data management tools (e.g. the lcg-cp command) to

retrieve the output dataset named in the row inserted into the SubmitLargeDataset table by the Table Updater.

4.4.6 R-GMA as a Grid Information System

R-GMA's intended purpose was to implement an information system for the gLite Grid middleware. It is designed to maintain up to date information on the status of resources on the Grid, and is intended to be queried by the gLite WMS when matching specifications for a job's hardware and software requirements to a resource on the Grid. The state of Grid resources is modelled by the gLite middleware using the Grid Laboratory Uniform Environment (GLUE) schema [73]. The GLUE schema defines the following entities:

- Site: a set of resources within a single administrative domain.
- Service: a Grid Service, which is a web service with additional features described in the Open Grid Services Infrastructure [74].
- Cluster: an aggregation of computing resources.
- Computing Element: an abstraction of a system managing computing resources.
- Storage Element: an abstraction of a storage resource.
- Storage Area: portion of storage to which a uniform set of policies applies.
- Access Protocol: protocol used to transfer files to and from an SE.
- Control Protocol: protocol used to control or manage an SE.

The GLUE schema defines a set of attributes for each of these entities. For example, some of the attributes which are defined for the Computing Element are GlueCEInfoTotalCPUs (the total number of CPUs available to the CE), GlueCEInfoLRMSType (the type of underlying Local Resource Management System for a CE), and GlueCEStateRunningJobs (the number of jobs currently in a running state on the CE).

Currently, the standard gLite WMS obtains information on Grid resources by querying the BDII server, which is an LDAP [75] database containing information on resources available on the Grid. The WMS can be configured so that instead of querying the BDII server, it queries R-GMA to get the same information. If R-GMA is to be used by the WMS for matchmaking, information describing the state of Grid resources must be inserted into R-GMA. This can be accomplished using a tool called GadgetIn (GIN) [76]. GIN takes the output from an application which provides information on a resource, and publishes it to R-GMA. The schema used by R-GMA to describe Grid resources is slightly different from the GLUE schema however. GIN uses a configuration file which contains a list of mappings between elements in the GLUE schema and their corresponding elements in the R-GMA schema. The configuration file lists these mappings for each column in each table in the R-GMA schema. The datatype associated with each column in the R-GMA schema is also specified. A selection of these mappings are shown here for one of the tables in the R-GMA schema, GlueCE.

```
table GlueCE
   primarykey:VARCHAR(128) UniqueID=GlueCEUniqueID
      column:INTEGER EstimatedResponseTime=
            GlueCEStateEstimatedResponseTime
      column:INTEGER FreeCpus=GlueCEStateFreeCPUs
      column:VARCHAR(128) GatekeeperPort=GlueCEInfoGatekeeperPort
      column:VARCHAR(128) GlueClusterUniqueID=GlueForeignKey-
      column:VARCHAR(128) HostName=GlueCEInfoHostName
      column:VARCHAR(128) InformationServiceURL
            =GlueInformationServiceURL
      column:VARCHAR(255) LRMSType=GlueCEInfoLRMSType
      column:INTEGER MaxCPUTime=GlueCEPolicyMaxCPUTime
      column:INTEGER MaxRunningJobs=GlueCEPolicyMaxRunningJobs
```

```
column:INTEGER MaxTotalJobs=GlueCEPolicyMaxTotalJobs
column:INTEGER MaxWallClockTime=GlueCEPolicyMaxWallClockTime
column:VARCHAR(255) Name=GlueCEName
column:INTEGER RunningJobs=GlueCEStateRunningJobs
column:VARCHAR(255) Status=GlueCEStateStatus
column:INTEGER TotalCPUs=GlueCEInfoTotalCPUs
column:INTEGER TotalJobs=GlueCEStateTotalJobs
```

An example of an information provider that GIN republishes output from is the lcg-info-wrapper program on CEs and SEs, which obtains information on the state of these resources, and outputs it in the LDIF [77] format. Figure 4-17 shows a screenshot displaying a portion of the LDIF output produced when the lcg-info-wrapper program is executed.

```
[lyttleto@infogrid7 gin]$ /opt/lcg/libexec/lcg-info-wrapper | more

dn: GlueSiteUniqueID=INFOGRID7,mds-vo-name=local,o=grid
objectClass: GlueTop
objectClass: GlueSite
objectClass: GlueKey
objectClass: GlueSchemaVersion
GlueSiteUniqueID: INFOGRID7
GlueSiteName: INFOGRID7
GlueSiteDescription: LCG Site
GlueSiteUserSupportContact: mailto: grid-ireland-alert@cs.tcd.ie
GlueSiteSysAdminContact: mailto: grid-ireland-alert@cs.tcd.ie
GlueSiteSecurityContact: mailto: grid-ireland-alert@cs.tcd.ie
GlueSiteLocation: DUBLIN
GlueSiteLatitude: 32
GlueSiteLongitude: 32
GlueSiteWeb: www.grid.ie
GlueSiteOtherInfo: 1
GlueSiteOtherInfo: www.grid.ie
GlueForeignKey: GlueSiteUniqueID=INFOGRID7
GlueForeignKey: GlueClusterUniqueID=infogrid7.testgrid
GlueForeignKey: GlueSEUniqueID=gridstore.testgrid
GlueForeignKey: GlueSiteUniqueID=INFOGRID7
GlueForeignKey: GlueClusterUniqueID=infogrid7.testgrid
GlueForeignKey: GlueSEUniqueID=gridstore.testgrid
GlueSchemaVersionMajor: 1
GlueSchemaVersionMinor: 2
```

Figure 4-17: LDIF output from the lcg-info-wrapper information provider.

GIN is installed on each Grid resource (such as CEs and SEs) for which status information will be published to the R-GMA information system. It runs as a service on these resources, and can be configured to publish information on that resource at regular intervals (for example, every minute). GIN parses the LDIF output of the lcg-info-wrapper application in order to retrieve the attribute-value pairs that describe the resource, and generates INSERT statements that an R-GMA Producer will execute in order to republish this information to R-GMA. The screenshot in Figure 4-18 shows GIN performing some of these INSERT statements.

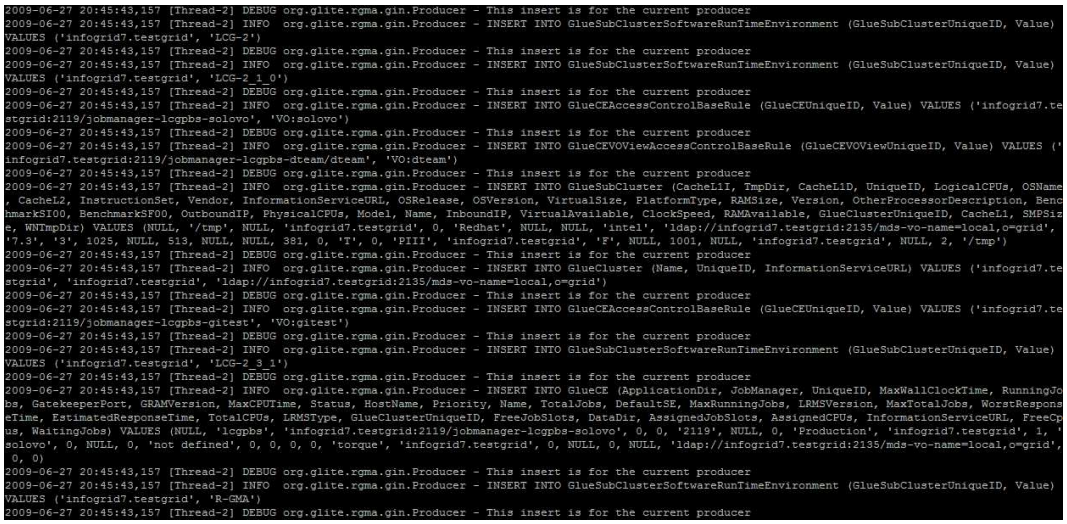


Figure 4-18: GIN publishing information to R-GMA.

Figure 4-19 shows a set of CEs and SEs which use GIN to insert information on their current state into tables in R-GMA. Clients such as the WMS can then use Consumers to query R-GMA and retrieve information on these resources. For the purpose of clarity, only two of the tables used by R-GMA to store information on the status of Grid resources are shown, GlueCE and GlueSE. The WMS has instantiated two Consumers to query both of these tables in Figure 4-19.

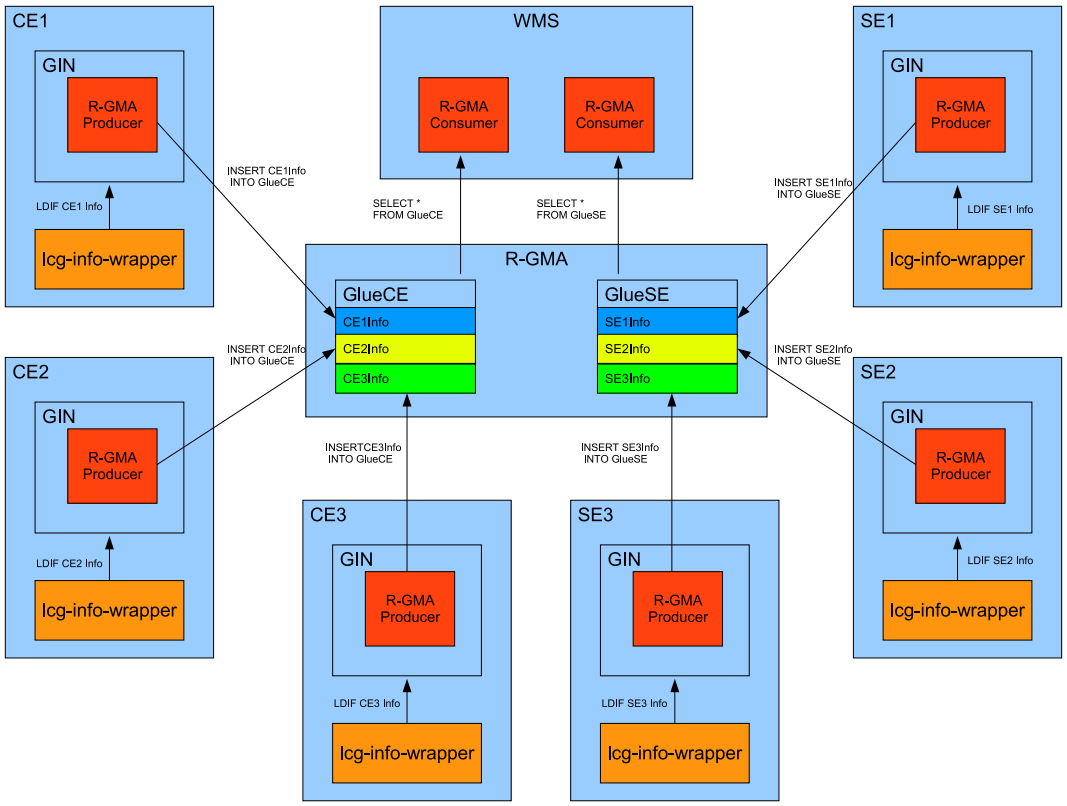


Figure 4-19: R-GMA used as an information system.

R-GMA's security mechanisms are used to ensure that only genuine information representing the state of resources on the Grid is published to the information system, and unauthorised users who are not in possession of a valid X.509 certificate issued by the administrators of the Grid cannot obtain this information. Authorised users can retrieve information on Grid resources by using the R-GMA APIs and command line tool to issue SELECT queries on the appropriate tables. Figure 4-20 illustrates the R-GMA command line utility being used to retrieve information from the GlueCE table describing a CE. There is only information about one CE in R-GMA in this example. A subset of the columns are requested in the SELECT query (UniqueID, FreeCPUs, LRMSType, EstimatedResponseTime and Status) as an entire record for a CE contains too many fields to be displayed clearly by the R-GMA command line tool.

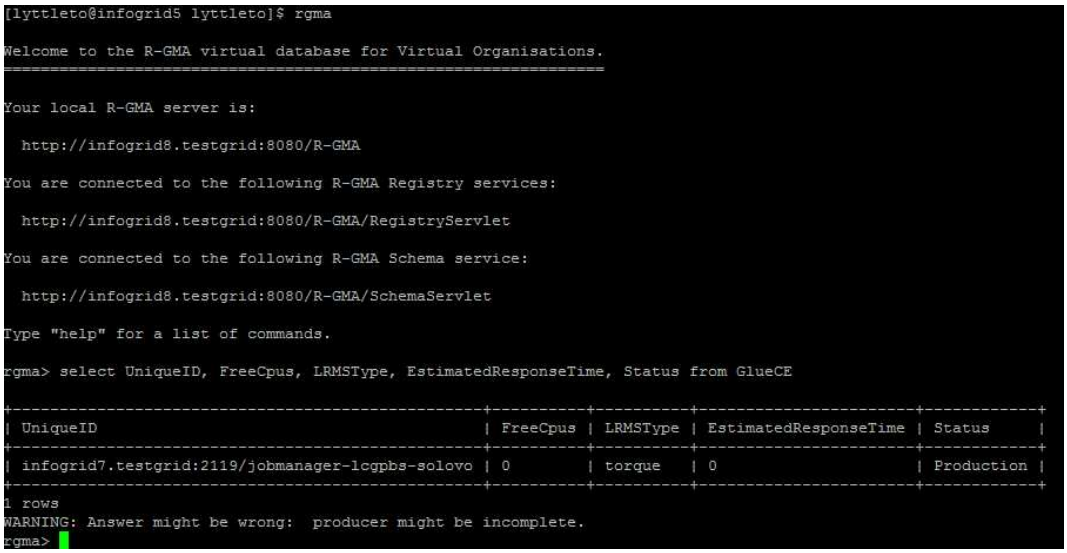


Figure 4-20: Retrieving information on Grid resources from R-GMA.

4.4.7 Using R-GMA as a Logging System

R-GMA can be used to log details of events that occur as jobs are executed by Grid middleware. This information can be used for a variety of purposes, such as:

- Troubleshooting problems that arise during the execution of jobs on the Grid.
- Analysing the past performance of Grid middleware components.
- In commercial Grids, the usage of Grid resources by individual users must be recorded in logs order to generate invoices.
- Investigating security issues, such as denial of service attacks on the Grid, unauthorised usage of Grid resources, etc.

Files containing logging information for the Grid are stored on various components of the gLite middleware. For example, on the WMS, Condor log files contain details of job submission to CEs, and the execution and completion of these jobs. There are also a number of log files on each CE that contain information on incoming jobs, authorization and authentication operations carried out by the CE, and job submissions to the LRMS.

Information in these files can be queried and accessed more effectively if it is stored in a single database, as opposed to being stored in a number of text files located on machines distributed across the Grid. The Accounting Processor for Event Logs (APEL) is an application developed by the creators of R-GMA which publishes the data contained in log files on CEs to R-GMA. R-GMA provides a means for authenticating clients performing operations on the logging system and a language for inserting and querying data. There are two main components of APEL, the parser and the publisher.

APEL Parser

The APEL parser extracts data from the various log files, and inserts it into tables in a MySQL database. The parser is a command line executable (apel-pbs-log-parser), and can be scheduled as a cron job so that it is run automatically every day. The executable takes the location of a configuration file as a command line argument, as shown below.

```
/opt/glite/bin/apel-pbs-log-parser
   -f /opt/glite/etc/glite-apel-pbs/parser-config.xml
```

The parser-config.xml configuration file contains details of which log files to parse, and details of the database into which the parsed records will be inserted. An example of a parser-config.xml configuration file is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<ApelConfiguration enableDebugLogging="yes">
<SiteName>infogrid7.testgrid</SiteName>
<DBURL>jdbc:mysql://localhost:3306/apelLogs</DBURL>
```

```
<DBUsername>root</DBUsername>
<DBPassword>password</DBPassword>
<EventLogProcessor>
   <Logs searchSubDirs="yes" reprocess="yes">
      <Dir>/var/spool/pbs/server_priv/accounting</Dir>
   </Logs>
</EventLogProcessor>
<GKLogProcessor>
   <SubmitHost>infogrid7</SubmitHost>
   <Logs searchSubDirs="yes" reprocess="yes">
      <GkLogs>
         <Dir>/var/log</Dir>
      </GkLogs>
      <MessageLogs>
         <Dir>/var/log</Dir>
      </MessageLogs>
   </Logs>
</GKLogProcessor>
</ApelConfiguration>
```

This configuration file defines the CE for which the log parser is retrieving records using the Sitename element. Details of the MySQL database into which the parsed details from logfiles will be inserted are also specified. The DBURL element specifies the hostname of the machine hosting the database, the port the database listens on for commands from clients, and the name of the database in a URL (in this example, the URL is jdbc:mysql://localhost:3306/apelLogs). The DBUsername and DBPassword elements define the username and password used to access the database. The APEL parser configuration file also contains details of processors, which parse the data contained in files in specified directories. There are two processors defined in

this configuration file, an EventLogProcessor and a GKLogProcessor. The EventLogProcessor defines the location of the directory containing log files for the LRMS on a CE (in this case, /var/spool/pbs/server_priv/accounting). The GKLogProcessor specifies directories containing log files for the Gatekeeper (the component of the CE which accepts job submissions, and performs authentication and authorisation). In the above configuration file, there are two types of log file parsed by the GKLogProcessor, GkLogs and MessageLogs, both of which are located in the /var/log directory. The parser parses the records contained in these files, and inserts them into tables in the MySQL database, as shown in Figure 4-21.

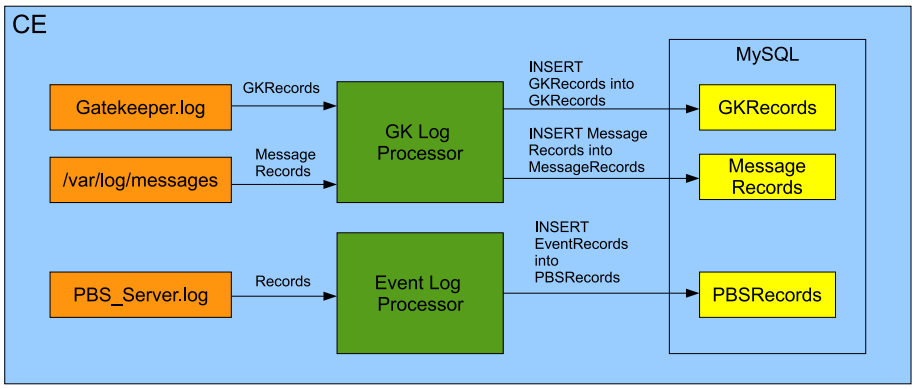


Figure 4-21: APEL parser.

The PBS log is parsed to extract the event data generated by jobs running on the CE's LRMS, which is then inserted into a table in the MySQL database called PBSRecords. The Gatekeeper log is also parsed and the data extracted from it is inserted into the GkRecords table. Data from the /var/log/messages file on the CE is extracted and published to the MessageRecords table.

APEL Publisher

The APEL publisher is a command line executable (apel-publisher), which accepts the name of an XML configuration file as a command line argument, as shown below.

```
/opt/glite/bin/apel-publisher
   -f /opt/glite/etc/glite-apel-publisher/publisher-config.xml
```

As with the parser, this executable can be run as a daily cron job. An example of the publisher-config.xml file is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<ApelConfiguration enableDebugLogging="yes">
   <SiteName>infogrid7.testgrid</SiteName>
   <DBURL>jdbc:mysql://localhost:3306/apelStuff</DBURL>
   <DBUsername>root</DBUsername>
   <DBPassword>password</DBPassword>
   <JoinProcessor publishGlobalUserName="no" >
      <Republish>missing</Republish>
   </JoinProcessor>
</ApelConfiguration>
```

The publisher-config.xml file specifies details of a MySQL database using the DBURL, DBUsername and DBPassword elements. The publisher performs a join on the PBSRecords, GkRecords and MessageRecords tables, inserting the resultant records into a single table in R-GMA, LCGRecords. The Republish element is used to specify what data is republished to R-GMA. This element can have three values:

- all: all records in the MySQL tables are republished to R-GMA
- missing: only records in the MySQL tables which have been inserted since the last time the publisher ran are republished to R-GMA

- nothing: no records are republished to R-GMA

An R-GMA Producer is used to insert the data in the tables in the MySQL database into the LCGRecords table defined in the R-GMA schema, as shown in Figure 4-22.

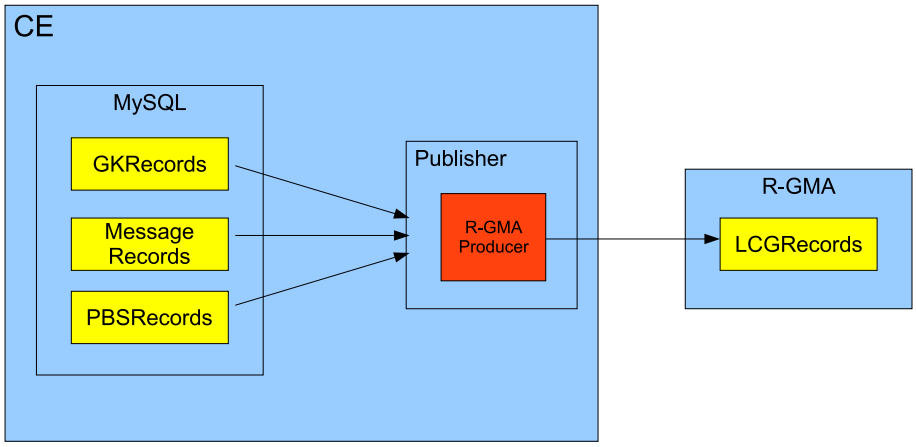


Figure 4-22: APEL publisher inserting data into R-GMA.

Infogrid-specific publishers

The Infogrid prototype also publishes other logging information generated by its components to R-GMA tables, in order that components of the prototype can access data produced by other components which may be located on remote machines. For example, the Infogrid prototype defines a table in the R-GMA schema which contains mappings of gLite job IDs to the source of those jobs, called gLiteIDToJobSourceMappings. When a Grid Job Creator submits a job to the WMS, it inserts a row into the gLiteIDToJobSourceMappings table. This row contains a gLite job ID and either the name of the active table or the value of the Executable Column in a row in the SubmitDataset/SubmitLargeDataset table, depending on how the job was submitted to Infogrid. The structure of the gLiteIDToJobSourceMappings table is as follows:

- gLiteID (VARCHAR(255)) : the gLite Id associated with a job
- Source (VARCHAR(255)) : either the name of the active table or the value

of the Executable Column in the row in the SubmitDataset or SubmitLargeDataset table associated with this job

The Infogrid prototype also maintains a table containing mappings between gLite job IDs (which are generated by the WMS) and Condor job IDs (which are generated by the LRMS on a CE when it processes a job). The CondorG[78] log file on the WMS contains details of events such as job submissions to CEs, job execution on the CEs, and job termination. An example of entries in the CondorG log file illustrating the submission of a job, the execution of a job and the termination of a job appear below.

```
000 (192.000.000) 07/05 14:17:29 Job submitted from host:
   <192.168.18.37:45310>
   (https://infogrid1.testgrid:9000/QCygkkl3mGlSMCdRy34kZw)
(UI=000003:NS=0000000003:WM=000004:BH=0000000000:
JSS=000003:LM=000000:LRMS=000000:APP=000000) (0)
...
001 (192.000.000) 07/05 14:17:32 Job executing on host:
   <192.168.18.37:45309>
...
005 (192.000.000) 07/05 14:17:33 Job terminated.
   (1) Normal termination (return value 0)
   Usr 0 00:00:00, Sys 0 00:00:00 - Run Remote Usage
   Usr 0 00:00:00, Sys 0 00:00:00 - Run Local Usage
   Usr 0 00:00:00, Sys 0 00:00:00 - Total Remote Usage
   Usr 0 00:00:00, Sys 0 00:00:00 - Total Local Usage
   0 - Run Bytes Sent By J
0 - Run Bytes Received By Job
0 - Total Bytes Sent By Job
0 - Total Bytes Received By Job
```

There are two forms of job ID in these log entries, Condor job IDs (e.g. 192.000.000) and a gLite job ID (e.g. https://infogrid1.testgrid:9000/QCygkkl3mGlSMCdRy34kZw). The Condor job ID appears in all entries in the CondorG log file related to that job. However, the gLite job ID only appears in the job submission entry. When a job termination event appears in the log file, the job is only identified by its Condor job ID. As will be seen in a later section, the process on the Infogrid prototype which monitors the status of jobs by retrieving data from the CondorG log file must be able to obtain the gLite job ID for completed jobs. In order to enable the Infogrid prototype to map Condor job IDs to their associated gLite job ID, Infogrid specifies a table in the R-GMA schema called CondorTogLiteMappings.

- CondorID (VARCHAR(255)) : the Condor job ID associated with a job
- gLiteID (VARCHAR(255)) : the gLite job ID associated with a job

The Infogrid prototype also stores logging data indicating Grid jobs which have completed, and the source of that job submission, in a table called FinishedJobs.

- JobID (VARCHAR(255)) : the gLite job ID of a job
- Source (VARCHAR(255)) : either the name of the active table or the value of the Executable column in the row in the SubmitDataset or SubmitLargeDataset table which initiated this job.

As mentioned previously, the Table Updater issues a continuous query on the FinishedJobs table, and retrieves the gLite job ID and the source for jobs as the Infogrid WMS detects that they have completed.

4.4.8 Infogrid Workload Management System

The Infogrid prototype has modified the gLite WMS so that it uses R-GMA to submit jobs to Infogrid CEs. The standard gLite WMS uses CondorG to submit jobs to CEs. CondorG is an extension to Condor that allows it to submit jobs to Globus gatekeepers using the GRAM protocol. The gLite WMS is composed of several subcomponents, as illustrated in figure 4-23.

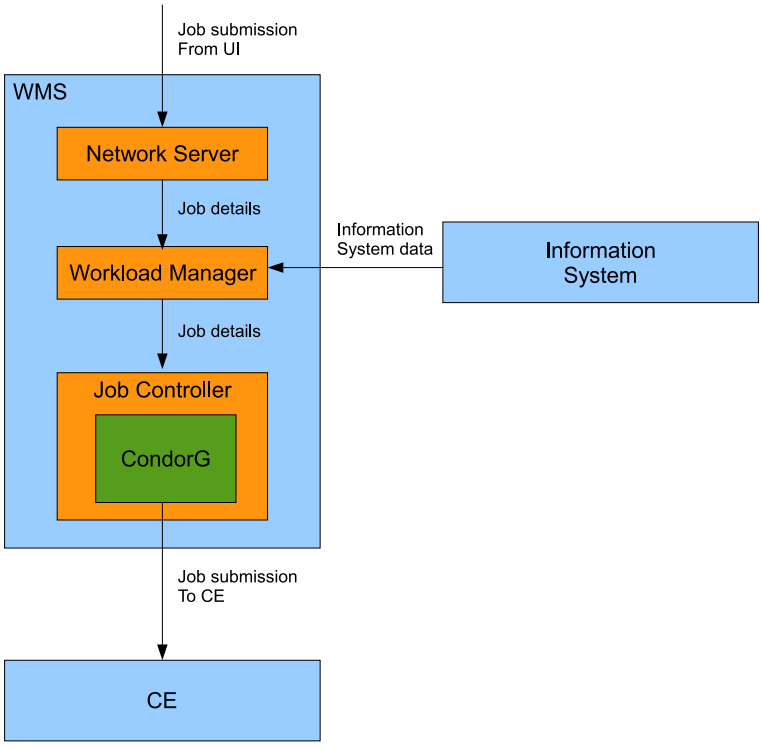


Figure 4-23: Components of the gLite WMS.

- Network Server: this listens for jobs that have been submitted to the WMS. It receives JDL files, job executables, and input files from clients.
- Workload manager: this consults the information system to find resources that are capable of executing jobs described in JDL files received by the Network Server, and selects a CE which will execute jobs.
- Job controller: this uses CondorG to submit the job to the CE that was selected by the Workload Manager.
- CondorG: an add-on for Condor which implements the GRAM protocol, allowing Condor to submit jobs to a Globus gatekeeper.

The following is an extract from the source code for the job controller. The job controller is written in C++.

```
parameters.assign( "-d " );
parameters.append( ad.submit_file() );
parameters.append( " 2>&1" );
result = CondorG::instance()->set_command
   ( CondorG::submit, parameters )->execute( info );
```

This code creates a string representing a job submission command for CondorG, and uses this string as input to a method that invokes this job submission. The parameters object is a string which contains arguments for a command line executable called condor_submit, which is used to submit jobs to a CE. The string is initially given the value "-d" by invoking the assign method of the parameters object, and the append method is then used to add the name of a file containing the job description ("ad.submit_file") and Unix file output and error redirection commands ("2>.&1"). An example of an invocation of the condor_submit command with the command line arguments defined in the parameters string is given below.

```
/opt/condor/bin/condor_submit -d
   /var/edgwl/jobcontrol/submit/AM/
   Condor.https_3a_2f_2fnode1.grid_3a9000_2fAMmhQRzhoi2ZnT1hzxgAOA.submit
   2>&1
```

As the description file is created after the WMS has found a CE for executing the job, this job does not use the matchmaking capabilities of Condor to decide where the job will be executed. The description file contains a line targeting a particular CE for its execution, similar to the following:

```
   GlobusScheduler = infogrid7.testgrid:2119/jobmanager-lcgpbs
```

This line specifies that when Condor processes this job, its matchmaking process is bypassed and the job is submitted to the Globus gatekeeper hosted on the infogrid7.testgrid machine. The URL for the GlobusScheduler also specifies a jobmanager on the CE (jobmanager-lcgpbs) that will submit the job to the CE's LRMS.

In the Infogrid prototype, the code for the job controller has been modified and recompiled, so that submission to the CE is achieved by an insertion into a table in the R-GMA database, as illustrated below.

```
// Create string containing SQL INSERT statement for job submission
   std::string SQLStatement;
   SQLStatement.append("INSERT INTO CETable (SubmitJob)
      VALUES ('"+parameters+"')");
// Use insert method of R-GMA Producer object (pp) to execute statement
in
// the SQLStatement string
   pp->insert(SQLStatement);
```

The job controller for the Infogrid WMS uses an R-GMA Producer to insert a row into the CETable table in R-GMA. This row contains the name of a directory on the WMS that stores the executable and input files (if any) for a job. This directory has the same name as the unique portion of the gLite job ID for that job. Figure 4-24 illustrates how the Infogrid CE obtains this tuple.

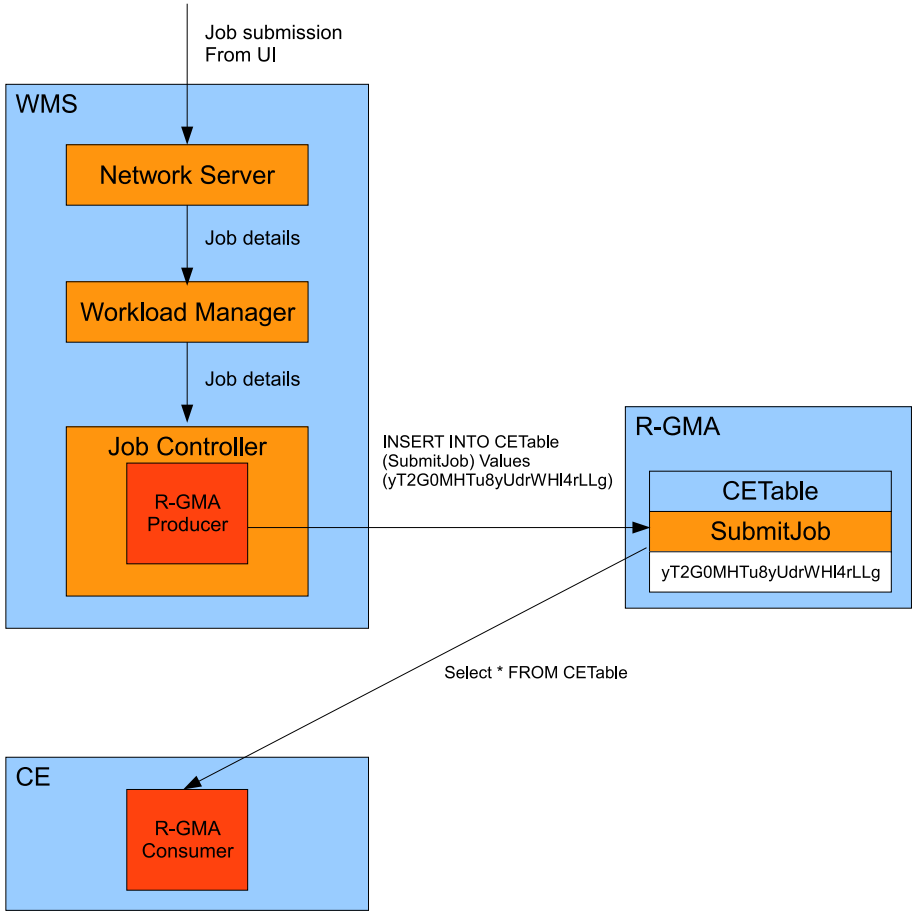


Figure 4-24: Remotely invoking jobs on CEs using R-GMA

The Infogrid CE uses an R-GMA Consumer to issue a continuous query on the CETable table. When it retrieves a tuple from the table, it transfers the job files contained in the directory on the WMS specified in the SubmitJob column to the CE, and submits the job to its LRMS.

When implementing the Infogrid prototype, the internal use of the WMS matchmaker was disabled, so that jobs could be submitted directly to the Infogrid prototype CE. The result is that the Infogrid prototype WMS cannot perform matchmaking between a job and several CEs ; it can submit to only one CE, an implementation of the Infogrid CE. This is a simplification for the purposes of experimentation, that yields an acceptable limitation in the context of the investigation into implementing components of the Grid middleware using R-GMA.

A problem that arose during the development of the Infogrid prototype was that the WMS was unable to set the status of the job to "Done" when the job had completed. The WMS continuously monitors a CondorG log file. When an entry is inserted into the CondorG log file indicating that a job has completed, the WMS updates the status of the job to "Done". However, as CondorG is not used by the Infogrid WMS to submit jobs to CEs, this information was not available to the WMS, and jobs would stay in a "Submitted" state, never reaching the "Done" state.

To work around this issue, a process on Infogrid CEs called the CondorLogMonitor continuously monitors the Condor log file produced by the Condor LRMS on the CE, and sends its contents to the WMS via R-GMA. When a record is entered into the CE's Condor log file, the CondorLogMonitor uses a Producer to insert a tuple into a table called CondorLogEntries. This tuple contains the text for the record in the Condor log file. A process called the LogUpdater, located on the WMS, issues a continuous query on this table and updates a synthetic CondorG log file on the WMS with the log entry it receives from the CE. Figure 4-25 shows the interaction between these processes.

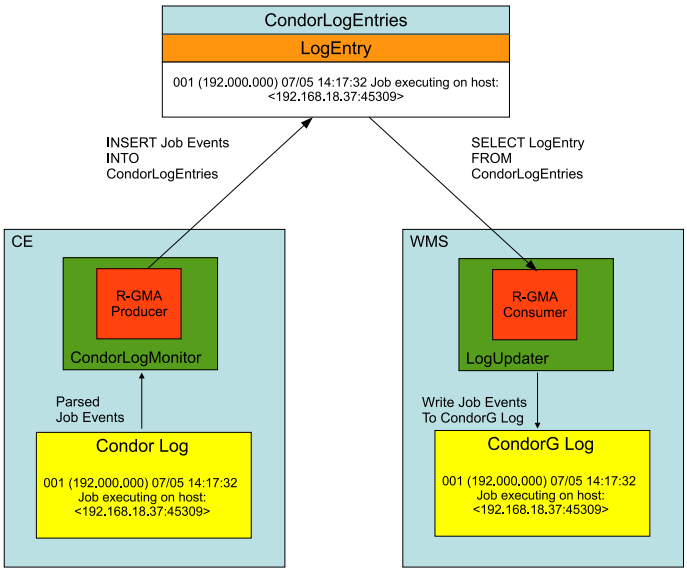


Figure 4-25: Updating CondorG log file on the WMS with CE data.

The Infogrid WMS monitors this synthetic Condor log file, using a process called the JobsMonitor. It informs the Table Updater component when a job has completed by

inserting a row into the FinishedJobs table. When the JobsMonitor detects an entry in the CondorG log file on the WMS indicating that a job has terminated (e.g. 005 (192.000.000) 07/05 14:17:33 Job terminated.) it extracts the Condor job ID from this entry (192.000.000), and issues a query on the CondorTogLiteMappings table, in order to find the gLite job ID associated with that Condor job ID, as illustrated in Figure 4-26.

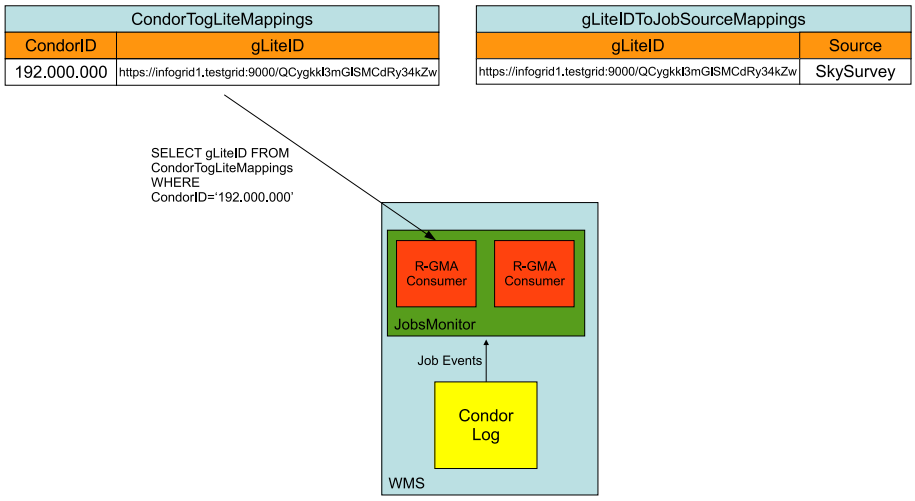


Figure 4-26: The JobsMonitor obtains the gLite ID for a terminated Condor job.

It is also necessary for the Table Updater to know what the source for a job was, which can be either an active table or an entry in either the SubmitDataset or SubmitLargeDataset tables. The JobsMonitor can discover this information by querying the gLiteIDToJobSourceMappings table, as illustrated in Figure 4-27.

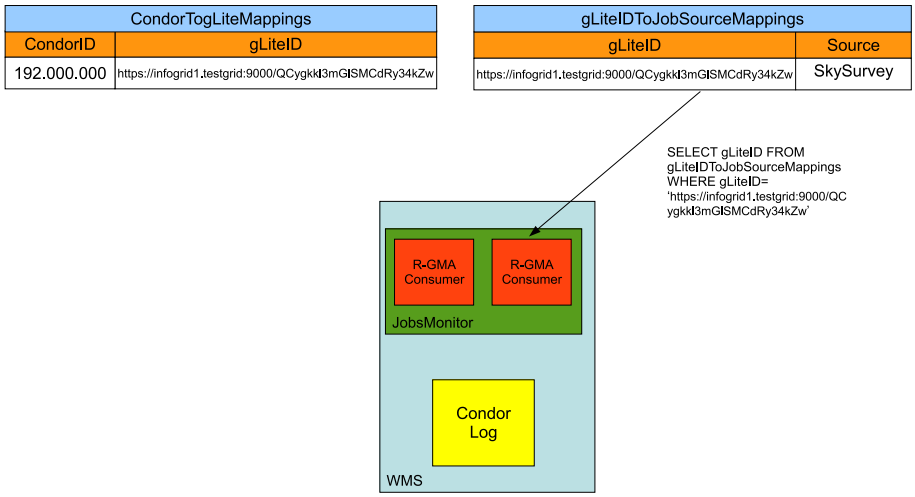


Figure 4-27: The JobsMonitor determines the source of a job.

Having obtained the gLite job ID for a Condor job and the source of that job, the JobsMonitor can send this information to the Table Updater. The JobsMonitor process runs on the WMS, while the Table Updater runs on the User Interface. The JobsMonitor must communicate across the network when informing the Table Updater that a job has completed, by inserting a row containing this information into the FinishedJobs table in R-GMA. The Table Updater uses a Consumer to issue a continuous query on this table, and can retrieve data inserted into it, as illustrated in Figure 4-28. The JobsMonitor can thus send a message to the Table Updater indicating that a job has completed, and without waiting for a response continue monitoring the CondorG Log.

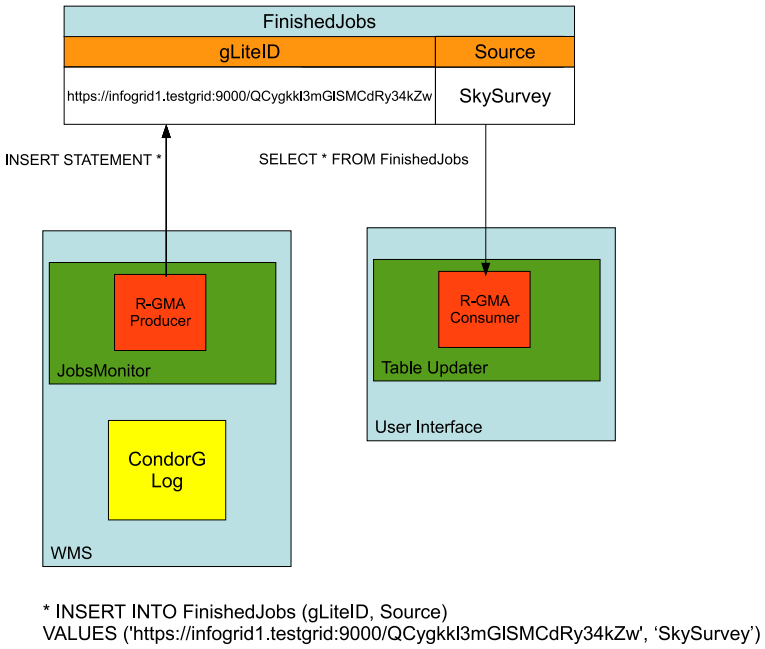


Figure 4-28: The JobsMonitor inserts a row into the FinishedJobs table.

4.5 Infogrid CE

The role of the CE is to authenticate and authorise job submissions from the WMS, and submit those jobs to a local cluster computing system, or LRMS. The standard gLite CE is composed of the following components:

- EDG gatekeeper: a modified version of the Globus gatekeeper. It listens for incoming jobs, and authenticates and authorises the user on whose behalf the WMS is submitting that job.
- Globus job manager: submits the job to the LRMS that will execute the job, and monitors its progress.
- LRMS: the cluster system used to execute jobs (e.g. Portable Batch System (PBS), LSF or Condor).

gLite jobs are submitted to a gatekeeper on the CE using the Global Resource Allocation Manager (GRAM) protocol. The GRAM protocol provides an interface to computing resources on a network. It allows heterogeneous computing resources to

be accessed via a uniform interface. Job submission from a GRAM client (such as CondorG) to a CE's gatekeeper is illustrated in Figure 4-29.

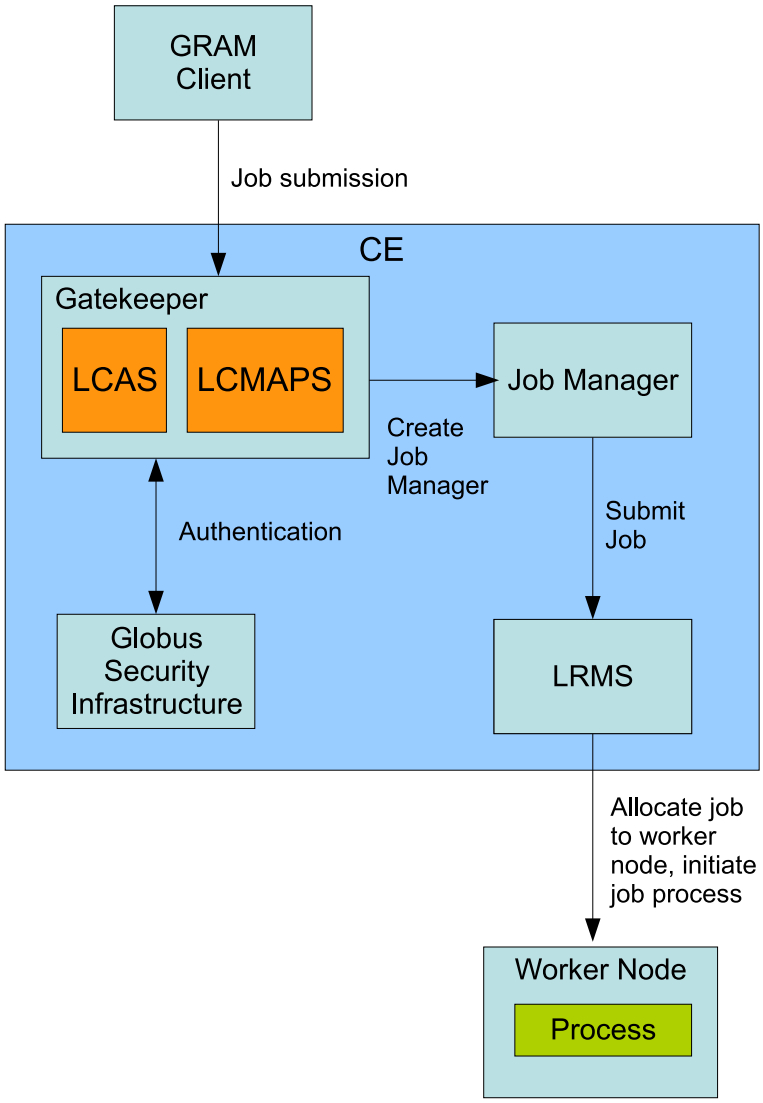


Figure 4-29: Job submission to a CE's gatekeeper using GRAM.

The GRAM client sends a job for submission to the gatekeeper. Before the CE can process a submitted job, it must authenticate the client, and check their authorizations. The gLite CE uses LCAS, the Local Centre Authorization Service [79], to authorise users over an authenticated communications channel based on their name, their VO affiliation, and the resources requested. This allows sites to maintain their autonomy by enabling them to enforce local security policies.

Users of the Grid are not required to have accounts on Grid resources in order to use those resources. The Local Credential Mapping Service (LCMAPS) [80] is used to execute jobs on the CE under one of a set of local accounts called pool accounts. There are a set of pool accounts on a CE for each VO that the CE is a member of. For example, for a VO called gitest, these pool accounts would be called:

```
gitest001, gitest002, gitest003,...,gitest050
```

When a job submission is received by the gatekeeper, it extracts the Distinguished Name (DN) from the X.509 certificate associated with the job. This certificate identifies the user who submitted the job. The gatekeeper consults a file containing mappings between DNs and VOs to see what VO the user who submitted the job is a member of. Entries in this file have a format similar to the following:

```
/C=IE/O=Grid-Ireland/OU=cs.tcd.ie/L=RA-TCD/CN=Oliver Lyttleton .gitest
```

This entry indicates that the user with the DN /C=IE/O=Grid-Ireland/OU=cs.tcd.ie/L=RA-TCD/CN=Oliver Lyttleton is a member of the gitest VO. If the CE receives a job submission from this user, LCMAPS will execute that job under a pool account assigned to the gitest VO. LCMAPS will check if the pool account gitest001 has been leased to another user. If this account has already been assigned, LCMAPS will check if gitest002, gitest003, etc. have been assigned until it finds an unleased pool account, and will lease that pool account to the incoming job submission request. The CE will then spawn a job manager process which runs under the context of that pool account. The job manager executes the job by submitting it to the CE's LRMS.

Given the extent to which R-GMA proved applicable to a number of Grid middleware functions the opportunity has been taken to explore whether it would facilitate a fresh approach to the most central component, the CE. This has yielded a uniquely "Infogrid" CE. The functionality of the gatekeeper can be carried out using R-GMA, by instantiating a Consumer which listens for insertions into a table in R-GMA representing incoming job submission requests. R-GMA's security mechanisms can be used to perform authentication and authorisation operations on the user that inserts rows into this table, and the Globus job manager replaced with code attached to the R-GMA Consumer which submits jobs to the LRMS, yielding a simplified system similar to that shown in Figure 4-30.

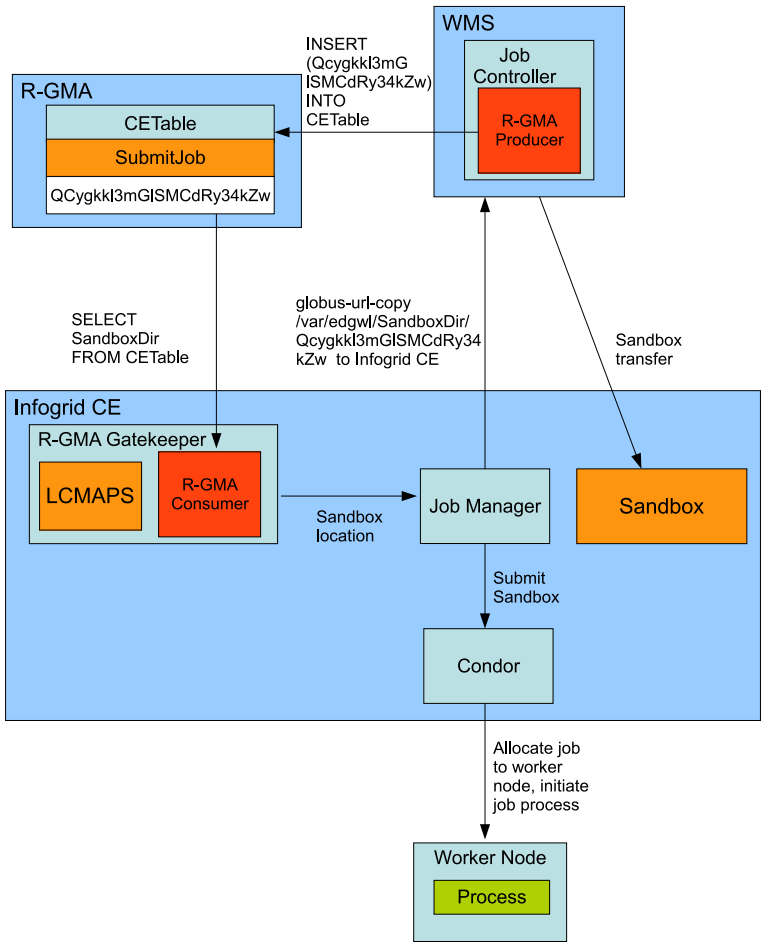


Figure 4-30: Submission from WMS to CE in Infogrid prototype.

The Infogrid WMS job controller inserts a row into a table that represents the interface to the CE, which in the case of the prototype Infogrid CE is called CETable.

This table contains one column, called SubmitJob, which contains the location of the directory on the WMS containing the input and executable files for a job. R-GMA's security mechanisms are used to authenticate and authorise job submissions from the WMS. The Infogrid CE does not currently support the kind of modularity of authorization mechanisms that LCAS provides (but in future it probably will). LCAS is not used (yet). Only R-GMA's authorisation mechanisms are used. To support authorisation the version of R-GMA used to implement Infogrid must be that available in gLite version 3.1 or greater.

The tuple inserted into the CETable table contains the name of the directory containing the input, executable, proxy certificate and Condor job description files for a job on the WMS. These files are located in subdirectories of the /var/edgwl/SandboxDir directory on the WMS. An application called globus-url-copy is used to transfer these files from the WMS to the CE. The Infogrid CE parses the Condor job description file in order to obtain the DN for the user who submitted the data, and then uses the LCMAPS API to map the job to a pool account by performing the following command:

```
std::cout <<(lcmaps_run_without_credentials(
"/C=IE/O=Grid-Ireland/OU=cs.tcd.ie/L=RA-TCD/CN=Oliver Lyttleton"))
<< std::endl;
```

With the submission authenticated and authorised, the job executable and input files in place on the CE, and account mapping performed, the Infogrid CE can submit the job to the LRMS, which in the Infogrid prototype is Condor. The Infogrid CE creates a Condor job description file similar to the following example:

```
universe = vanilla
executable =
   https_3a_2fnode1.tgui_3a9000_2f6igJ2gWEoj5_5ffqtpVD-hew.sh
output = std.out
error = std.err
log = /var/edgwl/logmonitor/CondorG.log/CondorG.1183555430.log
queue
```

A universe is a runtime environment for a job. The "vanilla" universe is used by the Infogrid CE for two reasons. Firstly, Condor is not being used to submit a job to the "Globus" universe (as in the case of the standard gLite WMS using CondorG to submit a job to the Globus gatekeeper on a standard CE). Secondly, the vanilla universe is used to submit jobs to Condor which do not need to be checkpointed or migrated. If jobs are to be submitted to Condor which use these features, they must be submitted to the "Standard" universe. The executables being invoked must also be recompiled using a compiler provided with the Condor middleware. The use of checkpointing and migration is not required by the Infogrid prototype, therefore the Infogrid CE submits jobs to Condor using the "vanilla" universe.

Entries in Condor's logfile relating to job submission to the Globus universe will differ slightly from entries relating to jobs submitted to the vanilla universe. The entry in the standard gLite WMS CondorG log file for a job submission to the Globus universe will contain a gLite job ID and other information, as illustrated in the following example.

```
000 (192.000.000) 07/05 14:17:29 Job submitted from host:
   <192.168.18.37:45310>
   (https://infogrid1.testgrid:9000/QCygkkl3mGlSMCdRy34kZw)
(UI=000003:NS=0000000003:WM=000004:BH=0000000000:
JSS=000003:LM=000000:LRMS=000000:APP=000000) (0)
```

When a job is submitted to the vanilla universe in Condor, as is the case when the Infogrid CE submits jobs to its LRMS, the entries generated in the Condor log are slightly different. The job submission entry does not contain details of a gLite job ID nor does the log contain additional information normally returned by a Globus gatekeeper. The example below shows a job submission entry in a Condor logfile for a submission to the vanilla universe.

```
000 (192.000.000) 07/05 14:17:29 Job submitted from host:
<192.168.18.37:45310>
```

The job submission entries produced by submission to the vanilla universe in Condor log files require extra lines added to them indicating the gLite ID of this job and the output normally returned by the Globus gatekeeper upon job submission (as in the CondorG log file), otherwise when the CondorLogMonitor transfers them to the WMS they will not be parsed correctly, and jobs will not enter the "Done" state. The CondorLogMonitor adds these extra lines to the log entry for job submission it sends to the WMS. In order to do this, the CondorLogMonitor must be able to establish what the gLite job ID is for this job. It does this by querying the CondorTogLiteMappings table which contains mappings between Condor and gLite

job IDs. The Infogrid CE inserts rows into this table at the time it submits a job to the LRMS. The CondorLogMonitor then passes entries from the Condor log file reflecting the status of the job (e.g. Submitted, Running, Done) via R-GMA to the WMS which writes these entries to its own local Condor log file.

4.6 Conclusions

This chapter described the implementation of the prototype Infogrid middleware described in Chapter 3. The justification for using a combination of R-GMA and gLite to implement the prototype was explained. The components of R-GMA, and how the implementation of the various components of the Infogrid prototype use R-GMA, were described.

Chapter 5

Experimental results

5.1 Introduction

The Ingrim prototype has been constructed as an experimental instrument with which to test the hypothesis introduced in chapter 3 of this thesis. The central research questions that this thesis investigates are:

1. Can a single database technology be used to implement services required by Grid middleware components that are implemented using a number of separate technologies in existing Grid middleware?

2. How does the Infogrid prototype perform as characteristics of jobs (e.g. active table size, job input size, chunk limit size, processing of data with/without memoisation, etc.) vary?

In order to test the hypothesis that a database technology can implement a number of services required by Grid middleware, the Infogrid prototype attempted to use R-GMA for the following purposes:

- a relational interface for job submission to the Grid
- an information system for the Grid
- a Computing Element (CE)

- a logging system

Experiments were conducted to determine if the Infogrid prototype could successfully use R-GMA to implement these components of the Grid middleware.

5.2 Relational Interface to the Grid

In order to test that the Infogrid prototype provided a relational job submission interface to the Grid, the Infogrid active table creator was used to create an active table for a program that performed sequence alignment, and data was inserted into this active table for processing. Experiments illustrating the effect that chunk limits and memoisation had on the time required to process data were also performed. Another experiment was conducted in which data of each of the available datatypes in R-GMA was inserted into table columns of a variety of datatypes, in order to evaluate the ability of active tables to perform type checking. Insertions were also made into the SubmitDataset table to demonstrate how this table could serve as an interface for submitting jobs to the Grid that used sets of rows from tables as input. Finally, job submissions to the Grid were made via the insertion of rows into the SubmitLargeDataset table, in order to demonstrate the effectiveness of this table as an interface for the submission of jobs.

5.2.1 Demonstrate submission of jobs to the Grid via active tables

Aim of experiment

Demonstrate that active tables implemented using R-GMA provide a relational interface through which data can be submitted for processing to the Grid.

Form of experiment

The screenshot in Figure 5-1 illustrates the Infogrid active table creator being used to create an active table which performs sequence alignment using the clustalw ap-

plication.

Figure 5-1: Using the Infogrid web interface to create an active table.

In Figure 5-1, the active table being created has the following columns:

- *SourceSequence:* the string representing a gene sequence within which clustalW will search for the TargetSequence.
- *TargetSequence:* the string representing a gene sequence which clustalW will search for in the SourceSequence.
- *AlignmentScore:* score indicating the degree to which the TargetSequence string occurs in the SourceSequence string.

The following SQL statement is used to create this active table:

```
CREATE TABLE SequenceAlign (TargetSequence VARCHAR(65535) INPUT,
   SourceSequence VARCHAR(65535) INPUT,
   AlignmentScore Double OUTPUT)
```

The Infogrid prototype assumes that the executable used to perform the function associated with an active table (in this case clustalw) is already present in a particular directory on the UI, as was the case in this experiment. The chunk limit for this active table was set to 25 (a job would be submitted to the Grid for every 25 rows that were inserted into the active table).

After creating the SequenceAlign table, it was tested experimentally by inserting 25 pairs of gene sequences into it using the Java R-GMA API. The values contained in the SourceSequence column were 50 random gene sequences containing 5000 characters generated using a random DNA generator [81]. The values contained in the TargetSequence column were a gene sequence containing 1000 characters, also created using the random DNA generator.

Results of experiment

Figure 5-2 illustrates the contents of the SequenceAlign table after 25 rows were inserted into it. The SourceSequence and TargetSequence columns are abbreviated for clarity. The data representing input to the sequence alignment operations are highlighted in blue. The AlignmentScore, Status and JobID columns are empty as the Grid job which processes the input data has not yet completed.

Input Data

SequenceAlign				
SourceSequence	TargetSequence	AlignmentScore	Status	JobID
GATTGTAG...TTTT	TCGTACAG...CACC			
CGTGTGTT...CTAA	TCGTACAG...CACC			
TCGTACAG...CACC	TCGTACAG...CACC			
TGTACCGT...AAAT	TCGTACAG...CACC			
CGGACCAT...TGAT	TCGTACAG...CACC			
GCTGGATC...CTGA	TCGTACAG...CACC			
GATCTTGA...TGGA	TCGTACAG...CACC			
GCCTGAGT...GCCG	TCGTACAG...CACC			
ATTGCCGAT...CGAT	TCGTACAG...CACC			
CCATGGTA...GGAA	TCGTACAG...CACC			
ATTGGCCA...CCGT	TCGTACAG...CACC			
TAGGCCAG...CCCT	TCGTACAG...CACC			
GACTGGAT...AATG	TCGTACAG...CACC			
ATTGGCCA...CCGT	TCGTACAG...CACC			
TAGGCCAG...CCCT	TCGTACAG...CACC			
CGAAATGG...GTAT	TCGTACAG...CACC			
TGACGTGT...AAAT	TCGTACAG...CACC			
ATGGCCAA...TATA	TCGTACAG...CACC			
GTATGGTC...CACC	TCGTACAG...CACC			
GATGGATC...TGGG	TCGTACAG...CACC			
CGAGGCCA...TGTT	TCGTACAG...CACC			
GATTGTAC...AATG	TCGTACAG...CACC			
TGTGTTCC...GATG	TCGTACAG...CACC			
CGCGGTTA...TGGA	TCGTACAG...CACC			
CATATTAC...AAAA	TCGTACAG...CACC			

Figure 5-2: SequenceAlign active table after rows have been inserted into it, but before job has completed.

The Grid Job Creator for the SequenceAlign active table created a single job invoking the clustalw executable 25 times, using the values contained in the SourceSequence and TargetSequence columns as input. After 3 minutes 17 seconds this job had completed, and the Table Updater inserted 25 rows containing both the original data inserted into the SequenceAlign table and the alignment scores for each of those pairs of sequences that were calculated by clustalw. Figure 5-3 illustrates the contents of the SequenceAlign table after the Table Updater inserted the rows containing the output scores into it. This result demonstrates that active tables as implemented in the Infogrid prototype can provide a relational Grid job submission interface, and implement interfaces to functions executed on the Grid as tables in a database technology. It also illustrates the downside of using R-GMA: the input rows persist (for the retention period) concurrently with identical data in the output rows.

Input Data Output Data

SequenceAlign				
SourceSequence	TargetSequence	Alignment Score	Status	JobID
GATTGTAG...TTTT	TCGTACAG...CACC			
CGTGTGTT...CTAA	TCGTACAG...CACC			
TCGTACAG...CACC	TCGTACAG...CACC			
TGTACCGT...AAAT	TCGTACAG...CACC			
CGGACCAT...TGAT	TCGTACAG...CACC			
GCTGGATC...CTGA	TCGTACAG...CACC			
GATCTTGA...TGGA	TCGTACAG...CACC			
GCCTGAGT...GCCG	TCGTACAG...CACC			
ATTGCCGAT...CGAT	TCGTACAG...CACC			
CCATGGTA...GGAA	TCGTACAG...CACC			
ATTGGCCA...CCGT	TCGTACAG...CACC			
TAGGCCAG...CCCT	TCGTACAG...CACC			
GACTGGAT...AATG	TCGTACAG...CACC			
ATTGGCCA...CCGT	TCGTACAG...CACC			
TAGGCCAG...CCCT	TCGTACAG...CACC			
CGAAATGG...GTAT	TCGTACAG...CACC			
TGACGTGT...AAAT	TCGTACAG...CACC			
ATGGCCAA...TATA	TCGTACAG...CACC			
GTATGGTC...CACC	TCGTACAG...CACC			
GATGGATC...TGGG	TCGTACAG...CACC			
CGAGGCCA...TGTT	TCGTACAG...CACC			
GATTGTAC...AATG	TCGTACAG...CACC			
TGTGTTCC...GATG	TCGTACAG...CACC			
CGCGGTTA...TGGA	TCGTACAG...CACC			
CATATTAC...AAAA	TCGTACAG...CACC			
GATTGTAG...TTTT	TCGTACAG...CACC	3	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
CGTGTGTT...CTAA	TCGTACAG...CACC	3	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
TCGTACAG...CACC	TCGTACAG...CACC	2	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
TGTACCGT...AAAT	TCGTACAG...CACC	4	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
CGGACCAT...TGAT	TCGTACAG...CACC	2	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
GCTGGATC...CTGA	TCGTACAG...CACC	2	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
GATCTTGA...TGGA	TCGTACAG...CACC	3	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
GCCTGAGT...GCCG	TCGTACAG...CACC	2	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
ATTGCCGAT...CGAT	TCGTACAG...CACC	3	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
CCATGGTA...GGAA	TCGTACAG...CACC	4	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
ATTGGCCA...CCGT	TCGTACAG...CACC	1	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
TAGGCCAG...CCCT	TCGTACAG...CACC	3	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
GACTGGAT...AATG	TCGTACAG...CACC	2	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
ATTGGCCA...CCGT	TCGTACAG...CACC	2	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
TAGGCCAG...CCCT	TCGTACAG...CACC	2	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
CGAAATGG...GTAT	TCGTACAG...CACC	1	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
TGACGTGT...AAAT	TCGTACAG...CACC	2	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
ATGGCCAA...TATA	TCGTACAG...CACC	2	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
GTATGGTC...CACC	TCGTACAG...CACC	3	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
GATGGATC...TGGG	TCGTACAG...CACC	3	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
CGAGGCCA...TGTT	TCGTACAG...CACC	4	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
GATTGTAC...AATG	TCGTACAG...CACC	1	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
TGTGTTCC...GATG	TCGTACAG...CACC	2	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
CGCGGTTA...TGGA	TCGTACAG...CACC	2	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q
CATATTAC...AAAA	TCGTACAG...CACC	1	Cleared	https://infogrid1.testgrid:9000/vg4BY5p2xh__IhwU1pt57Q

Figure 5-3: SequenceAlign active table after output tuples have been inserted into it on completion of a job.

5.2.2 Evaluate how processing time for datasets is affected by number of input columns in active tables

Aim of experiment

Evaluate how the processing time of datasets is affected as the number of columns in an active table increases.

Form of experiment

For this experiment, an addition executable was used which takes an arbitrary number of integers as command line arguments, and returns the sum of these integers. For example, the following invocation returns the number 15 (1+2+3+4+5).

```
Addition 1 2 3 4 5
15
```

Although this is not a typical application that is executed on the Grid, it can be invoked with varying numbers of command line arguments, and is ideal for use in an experiment determining how the number of input columns in an active table affects the performance of the Infogrid prototype. In this experiment, 4 active tables implementing the above addition function with 1, 5, 20, and 50 input columns were constructed. Each of these active tables had a chunk limit of 500. 75 chunks of 500 tuples with the appropriate number of inputs were inserted into each of these active tables, resulting in the Grid Job Creator for each of these tables creating 75 jobs. These active tables had the following structure (where N is the number of input columns):

```
Operand1 INTEGER
.................
.................
OperandN INTEGER
Result INTEGER
```

The time taken for the output to be calculated and inserted into the appropriate active table was measured for each of these jobs.

Results of experiment

The Infogrid prototype successfully processed all jobs submitted to all 4 tables. Table 5.1 illustrates a summary of the results from this experiment, showing the mean processing time and standard deviation for the 75 jobs submitted from each active table.

Table inputs	Mean processing time (mins)	Standard deviation (mins)
1	02:29	00:33
5	02:36	00:37
20	02:49	00:46
50	03:18	00:59

Table 5.1: Mean and standard deviations for time taken for active tables with various numbers of input columns to process 75 chunks of 500 tuples

Figures 5-4, 5-5, 5-6 and 5-7 chart in more detail the times taken by each active table to process these jobs. These times are broadly similar in all cases, with some outliers. There is a trend upwards in the mean time for processing chunks as the number of

input columns in an active table increases. The mean processing time for a chunk of 500 tuples in an active table with 50 inputs was 3 minutes 18 seconds, compared to 2 minutes 29 seconds for chunks of 500 tuples in an active table with 1 input, a difference of 49 seconds.

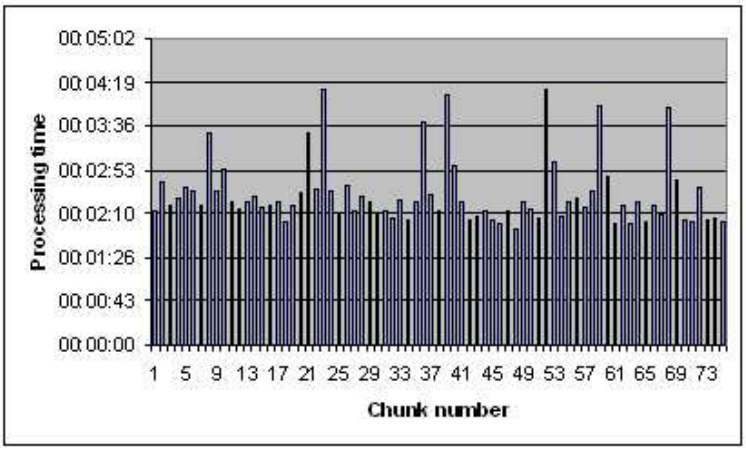


Figure 5-4: Latencies for processing 500 insertions into addition active table with 1 input column

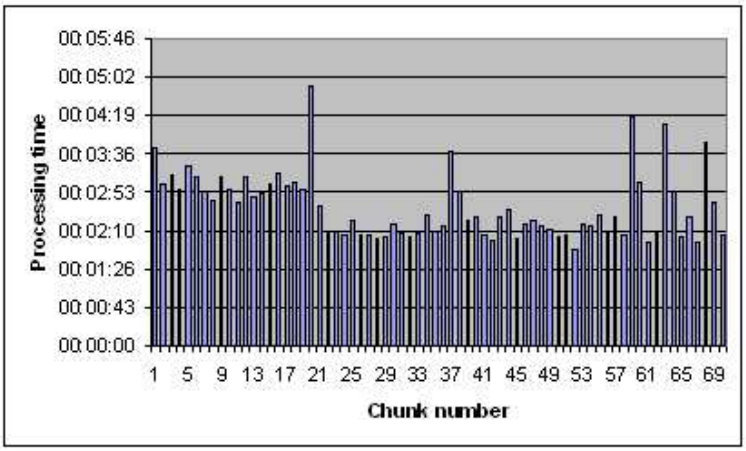


Figure 5-5: Latencies for processing 500 insertions into addition active table with 5 input columns

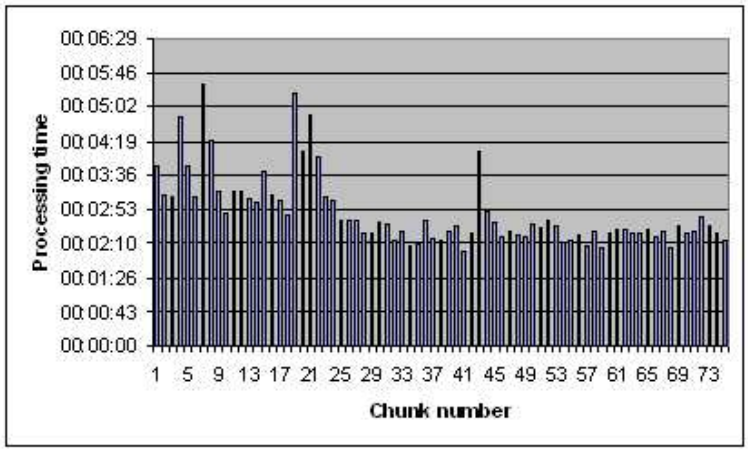


Figure 5-6: Latencies for processing 500 insertions into addition active table with 20 input columns

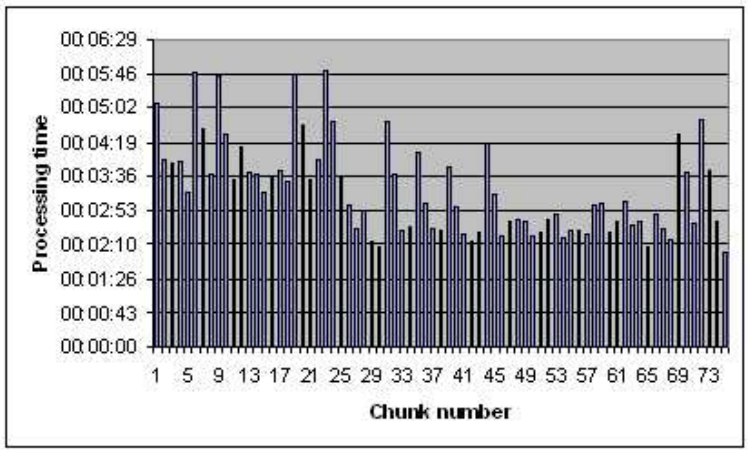


Figure 5-7: Latencies for processing 500 insertions into addition active table with 50 input columns

The trivial nature of the operation carried out in this experiment (addition of integers) does not lead to the lengthy execution times that more complex operations would. However, the experiment shows that the performance of the Infogrid prototype does not degrade to an unacceptable level as the number of columns in an active table increases to 50.

5.2.3 Evaluate how Infogrid prototype performs as increasing numbers of tuples are inserted into active tables

Aim of experiment

The aim of this experiment is to assess how the performance of Infogrid is affected by increasing the number of rows that are inserted into an active table.

Form of experiment

Rowsets containing 100, 500, 1000, 5000, and 10000 rows were inserted into an active table that performed the sequence alignment operation used in Section 5.2.1. These rows were inserted using statements such as:

```
INSERT INTO SequenceAlign (SourceSequence, TargetSequence)
   VALUES ('GTGGGCAC...GGGG','CCATGATCC...AGT')
```

The values inserted in the above statement are truncated for clarity. As in the experiment in Section 5.2.1, the values for the SourceSequence column contained gene sequences 5000 characters long, while the values for the TargetSequence column contained gene sequences 1000 characters long.

In order that a single job processed all the rows in each rowset, multiple active tables performing the sequence alignment function were created with the appropriate chunk limits (100, 500, 1000, 5000 and 10000). It was necessary to create multiple active tables with different chunk limits as it is not possible to alter the chunk limit of an active table after it has been created. The time taken to process the rowsets was measured in each case.

Results of experiment

Table 5.2 illustrates the times taken for the Infogrid prototype to process the various rowsets.

Size of rowset (number of rows)	Processing time (secs)
100	379
500	1374
1000	2523
5000	ERROR
10000	ERROR

Table 5.2: Processing time for various sizes of rowsets

Although the R-GMA Producer in this experiment could insert sets of 100, 500 and 1000 tuples into the active table (each tuple was approximately 6Kb in size), attempting to insert larger numbers of tuples into the active table caused an "out of memory" error to occur, as shown below:

```
org.glite.rgma.RGMABufferFullException:
Memory used for tuples has exceeded maximum:
23334646 bytes > 23327539 bytes
```

This error occurs when the buffer used by the Producer to store data fills up. When tuples exceed the retention period specified for that producer, they are removed from the buffer. However, if tuples are inserted into this buffer faster than they are removed (as in this experiment), the Producer will run out of memory, causing the above error. In order to modify the Infogrid prototype so that it can allow greater

amounts of data to be inserted into active tables in short periods of time, further research is required.

5.2.4 Evaluate how reuse of cached results in active tables affects processing times for datasets

Aim of experiment

This experiment aims to determine if the use of memoisation in active tables can result in improved execution times for rowsets.

Form of experiment

This experiment investigated how the use of memoisation affects the processing time for rowsets inserted into active tables when the number of duplicates in those rowsets, and the time taken to invoke the function performed by the active table for one row varies.

An active table which performed the sequence alignment operation called SequenceAlign, as in previous experiments, was used for this experiment. The longer the pair of sequences that the alignment operation is performed on, the more time is required for the operation to complete. Therefore, in order to vary the time taken for the function performed by the active table to complete, in this experiment alignments between pairs of long gene sequences and pairs of shorter gene sequences were performed. The long gene sequence pairs contained sequences 26000 and 1000 characters in length, while the short gene sequence pairs were each 100 characters in length. A sequence alignment on a pair of the long gene sequences took approximately 30 seconds, while the same operation completed virtually instantly on the short gene sequence pairs.

For both the long and short sequence pairs, rowsets containing 100 rows with 0, 25, 50, 75 and 100 duplicates were created. These rows were inserted into the SequenceAlign active table, both with and without memoisation enabled on that table. It is not possible in the Infogrid prototype to enable or disable memoisation after

an active table has been created, therefore two separate active tables performing the sequence alignment operation had to be created for this experiment, one with memoisation enabled, the other with memoisation disabled. Comparisons were performed between the time required to process these rowsets.

Results of experiment

The following is a description of the latencies associated with a job resulting from inserting 100 rows containing the longer gene sequences, consisting of 50 pairs of duplicate rows, into the SequenceAlign active table both with and without memoisation enabled. The results below show the latencies involved in preparing the job (which includes performing the memoisation check) and executing the job.

Job execution time for long gene sequence pairs (26000 and 1000 characters in length), with memoisation enabled:

Time to insert, prepare and submit data to Grid: 418 seconds

Time to execute job on Grid, obtain output and insert updated rows into active table: 1689 seconds

Total processing time: 2107 seconds

Job execution time for long gene sequence pairs (26000 and 1000 characters in length), without memoisation enabled:

Time to prepare and submit data to Grid: 21 seconds

Time to execute job on Grid, obtain output and insert updated rows into active table: 3119 seconds

Total processing time: 3140 seconds

It took 2107 seconds to process the data when memoisation was enabled, as opposed to 3140 seconds when memoisation was disabled. The memoisation check reduced the overall processing time for the set of 100 rows, because the extra time required to perform the memoisation checks during the preparation of the job was shorter than

the time saved by not executing the sequence alignment function for the duplicate rows.

The same latencies are shown below for when 100 rows containing the shorter sequence pairs, consisting of 50 pairs of duplicate rows, were inserted into the same table with and without memoisation enabled.

Job execution time for short gene sequence pairs (both 100 characters in length), with memoisation enabled:

Time to insert, prepare and submit data to Grid (including memoisation checks): 412 seconds

Time to execute job on Grid, obtain output and insert updated rows into table: 107 seconds

Total processing time: 519 seconds

Job execution time for short gene sequence pairs (both 100 characters in length), without memoisation enabled:

Time to insert, prepare and submit data to Grid: 38 seconds

Time to execute job on Grid, obtain output and insert updated rows into table: 116 seconds

Total processing time: 154 seconds

It took 519 seconds to process the data when memoisation was enabled, as opposed to 154 seconds when memoisation was disabled. The memoisation check increased the overall processing time for the set of 100 rows in this case, because the extra time required to perform the memoisation checks was greater than the time saved by not performing sequence alignment for the duplicate rows. These latencies illustrate that memoisation can result in either an increase or a decrease in total processing time for a set of rows containing duplicates, depending on the time taken to perform the active table's function on each of those rows. Figure 5-8 shows the results obtained when similar experiments were run on sets of rows containing 0, 25, 50, 75 and 100

duplicates. It can be seen that the total processing time for the rowsets remains relatively constant when memoisation is disabled. However, when memoisation is enabled, the time taken to perform the sequence alignment for one set of inputs and the number of duplicates in the rowset significantly affect the processing time. For the dataset containing longer gene sequences, processing a chunk containing 1 tuple (which occurred when all 100 tuples in the dataset were identical) took 569 seconds, whereas processing a chunk containing 100 tuples (which occurred when there were no duplicates in the dataset) took 3563 seconds, 526% times longer. The difference in processing times for datasets containing the shorter sequences was not as great. In these cases, processing a chunk containing 1 tuple took 505 seconds, while processing a chunk containing 100 tuples took 517 seconds, only 2% longer.

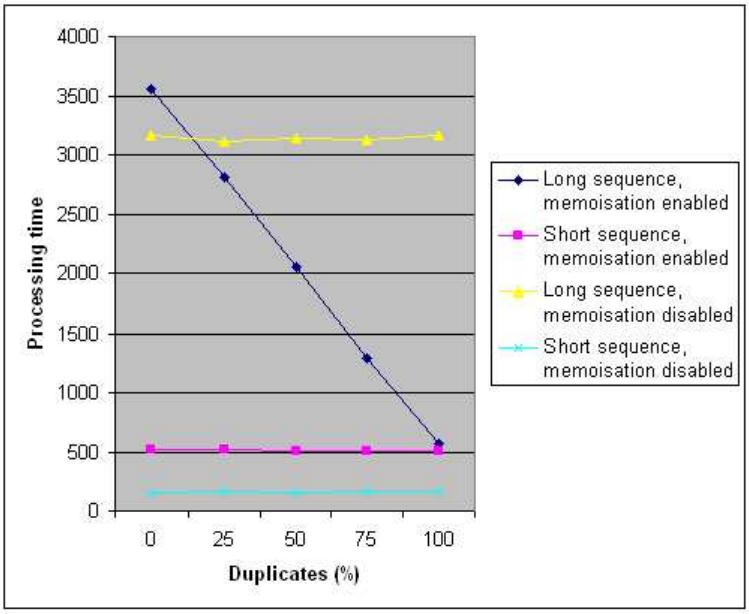


Figure 5-8: Processing times for datasets with/without memoisation.

These results show that for the rowsets containing the longer gene sequences, once a particular level of duplicates occurred, enabling memoisation resulted in a lower overall processing time for the rowset. However, for the rowsets containing the shorter sequences, the extra overhead introduced by performing the memoisation checks was greater than the reduction in overall processing times for the rowsets.

5.2.5 Evaluate how varying chunk limit on active tables affects processing times for rowsets

Aim of experiment

This experiment aims to demonstrate what effect varying the chunk limit of an active table can have on the total processing time for a series of rows.

Form of experiment

100 rows containing pairs of gene sequences 100 characters long were inserted into the SequenceAlign active table with the chunk limit for that table set to 1, 2, 5, 10, 20, 50 and 100. As chunk limits for active tables cannot be modified after the active table has been created, separate active tables had to be created for each of these chunk limits.

Results of experiment

The latency between the initial insertion of each row into the active table, and the insertion of an updated row containing output was noted. Figure 5-9 shows the time taken to process each row when the chunk limit was set to 1.

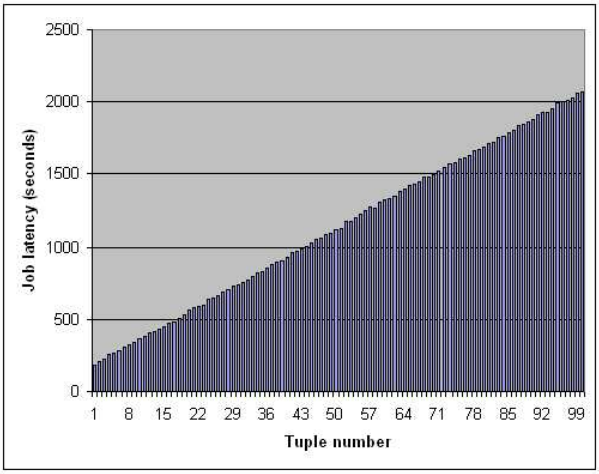


Figure 5-9: Processing time for rows with chunk limit of 1.

Figure 5-9 illustrates that the time required to process the rows increases for each consecutive row. This bottleneck occurs because the Infogrid prototype has only one CE, with one worker node. Job executions, and the processes performed by the Table Updater when inserting updated tuples into R-GMA, have to be performed in series, leading to a buildup of unfinished jobs. Figure 5-10 shows how the Infogrid prototype performed when the chunk limit on the active table was set to 2.

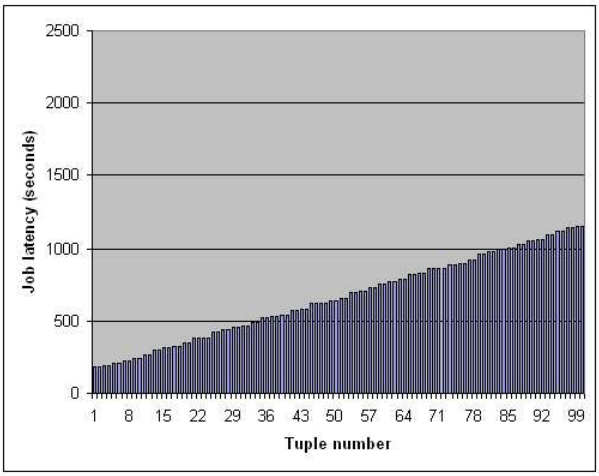


Figure 5-10: Processing time for rows with chunk limit of 2.

It can be seen that the total processing time for the data compared to the previous example almost halved when the chunking limit for the active table was set to 2. Figure 5-11 graphs the total processing times when the same rowset was inserted

into active tables with chunk limits of 1, 2, 5, 10, 20, 50 and 100. It can be seen that the total time for processing all rows decreases as the chunk limit increases. These results demonstrate that increasing the chunk limit can reduce the processing time for a set of rows. However, whether increasing the chunk limit decreases the overall processing time for a set of rows depends on factors such as the time taken to perform the active table function for one row, the number of rows inserted, and the number of worker nodes available to process data. More extensive experiments to explore these issues are proposed as future work in the Conclusions chapter of this thesis.

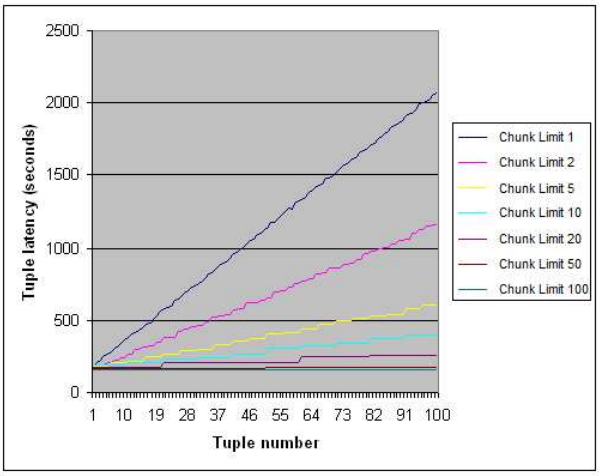


Figure 5-11: Processing time for 100 rows using active tables with various chunk limits.

5.2.6 Evaluate type checking by active tables

Aim of experiment

Determine if active tables perform type checking and enforce restrictions on structure of input for jobs.

Form of experiment

Excluding time and date related datatypes (TIMESTAMP, DATE and TIME) R-GMA supports the INTEGER, REAL, DOUBLE, CHAR and VARCHAR datatypes. A table was created in R-GMA which contained 5 columns, one of each of the

datatypes that was not related to time or date data. There are 25 combinations of datatypes that rows inserted into this table can contain. Some of these combinations will result in typing errors. 25 SQL statements that attempted to insert 5 different datatypes into each of the 5 columns were executed. The following CREATE TABLE statement was used to create this table:

```
CREATE TABLE typeCheck (an_int INTEGER PRIMARY KEY,
   a_real REAL,
   a_double DOUBLE,
   a_char CHAR(1),
   a_varchar VARCHAR(255))
```

In addition, as a practical example of type checking by an active table, the R-GMA table representing the SkySurvey operation from chapter 3.3.1 was created, using the following CREATE TABLE statement in the active table creator:

```
CREATE TABLE SkySurvey (
   Radio REAL INPUT PRIMARY KEY,
   OptR REAL INPUT PRIMARY KEY,
   OptI REAL INPUT PRIMARY KEY,
   OptZ REAL INPUT PRIMARY KEY,
   Infrared REAL INPUT PRIMARY KEY,
   QuasarCandidate REAL OUTPUT,
   Transient REAL OUTPUT)
```

The columns in this table are all of a numeric datatype (REAL). The input columns are defined as primary keys, therefore when inserting rows into this table values

must be specified for them. Rows containing invalid values for input columns (e.g. characters, symbols) or which did not specify values for input columns were inserted into this table, and the results observed.

Results of experiment

An example of one of the 25 SQL statements that inserted incorrectly typed data into the typeCheck table is shown below.

```
rgma> insert into typeCheck (an_int, a_real, a_double, a_char, a_varchar)
      values (1, 1.0, 1.0, a, 'aaa')
      ERROR: Value 'a' for column "a_char" does not match the
      defined type:  STRING(1) <> CHAR
```

In this example, the value for the "a_char" column in the SQL statement is not enclosed in quotes, therefore causing an error to occur. Table 5.3 illustrates the outcomes when various types of data were inserted into each of the columns in the typeCheck table. This table shows that:

- character based data (CHAR, VARCHAR) cannot be inserted into columns of a numeric data type (INTEGER, REAL, DOUBLE).
- numeric data is converted if it is inserted into a column of a different numeric type (e.g. REAL/DOUBLE data is rounded down if inserted into an INTEGER column, INTEGERS are converted to the appropriate datatype if inserted into a REAL/DOUBLE column). Numeric data cannot be inserted into a CHAR or VARCHAR column (unless it is enclosed in quotes).
- CHAR data can be inserted into a VARCHAR column, but if VARCHAR data is greater than one character in length, it cannot be inserted into a CHAR column

	Integer	Real	Double	Char	Varchar
Integer		Converted	Converted	Error	Error
Real	Rounded down		Converted	Error	Error
Double	Rounded down	Converted		Error	Error
Char	Error	Error	Error		Converted
Varchar	Error	Error	Error	Error$_1$	

Table 5.3: Datatype compatibility in R-GMA $_1$ if length of VARCHAR is greater than 1

Inserting a row into the SkySurvey active table containing floating point or integer data for the 5 input columns (Radio, OptR, OptI, OptZ and Infrared) was allowed by R-GMA, as shown below.

```
rgma> insert into SkySurvey (Radio, OptR, OptI, OptZ, Infrared)
      VALUES (359.5463, 16, 17.4532, 19.4265, 21.4533)
Inserted 1 row into SkySurvey
```

Inserting character data for any of the 5 input columns (Radio, OptR, OptI, OptZ or Infrared) resulted in R-GMA reporting an error, as shown below. The value of the "Radio" column in this SQL INSERT statement is a character string as it is enclosed in quotes ('359.5463')

```
rgma> insert into SkySurvey (Radio, OptR, OptI, OptZ, Infrared)
         VALUES ('359.5463', 16.4235, 17.4532, 19.4265, 21.4533)
      ERROR: Value '359.5463' for column "Radio" does not match the
      defined type:  STRING(8) <> REAL
```

Inserting non-alphanumeric data into any of the 5 input columns in the SkySurvey also resulted in R-GMA reporting an error, as shown below. In this SQL INSERT statement, an invalid non-alphanumeric character ($) is specified in the value of the "Radio" column.

```
rgma> insert into SkySurvey (Radio, OptR, OptI, OptZ, Infrared)
        VALUES ($359.5463, 16.4235, 17.532, 19.4265, 21.4533)
      ERROR: Error parsing INSERT statement:
           INSERT into SkySurvey (Radio, OptR, OptI, OptZ, Infrared)
           VALUES ($359.5463, 16.4235, 17.532, 19.4265, 21.4533)
      Caused by:  org.edg.info.sqlutil.parsql.ParseException:
      Expected numeric constant but found:  $359.5463
```

When an SQL INSERT statement which specified values for only 4 of the 5 input columns of the SkySurvey table was executed, R-GMA displayed an error indicating that the "Infrared" column could not be null, as shown below.

```
rgma> insert into SkySurvey (Radio, OptR, OptI, OptZ)
        VALUES (359.5463, 16, 17.4532, 19.4265)
      ERROR: Could not insert tuple into database.  Caused by:
      Cannot execute query:
           INSERT INTO SkySurvey (Radio,OptR,OptI,OptZ)
           VALUES (359.5463,16,17.4532,19.4265);
      Caused by:  java.sql.SQLException:Column 'Infrared' cannot be null
```

This demonstrates the ability of active tables to perform type checking on input data and enforce constraints on the number of command line arguments specified for jobs.

5.2.7 Submission of jobs to the Grid via SubmitDataset table

Aim of experiment

Demonstrate that insertion of rows into the SubmitDataset table provides a relational interface through which datasets composed of entire tables or sets of rows from tables can be submitted for processing to the Infogrid prototype.

Form of experiment

Four rows were inserted into the SubmitDataset table. These rows specified sets of rows from a table called "genomicData" as input. This table contained 1000 rows, and had the following structure:

- RowNo: INTEGER
- Sequence1: VARCHAR(1000)
- Sequence2: VARCHAR(65535)

Rows in this table were identified by the RowNo column, the values of which ranged from 1 to 1000. Input rowsets of varying size were specified for jobs using the following SQL statements:

```
INSERT INTO SubmitDataset (Input, Executable) VALUES
   ('SELECT * FROM genomicdata WHERE RowNo<=10', 'sequenceAlign')
INSERT INTO SubmitDataset (Input, Executable) VALUES
   ('SELECT * FROM genomicdata WHERE RowNo<=100', 'sequenceAlign')
INSERT INTO SubmitDataset (Input, Executable) VALUES
   ('SELECT * FROM genomicdata WHERE RowNo<=500', 'sequenceAlign')
INSERT INTO SubmitDataset (Input, Executable) VALUES
   ('SELECT * FROM genomicdata WHERE RowNo<=1000', 'sequenceAlign')
```

These rows instructed the Infogrid prototype to submit a job which uses an executable called sequenceAlign to process data contained in the resultset obtained when the SQL statements in the Input column are executed. The four SQL statements retrieved 10, 100, 500 and 1000 rows respectively. In all 4 cases, the value for Sequence1 in every row was 1000 characters long, while the value for Sequence2 in every row was 5000 characters long. The sequenceAlign executable calculated and output a similarity score for each pair of rows in the input rowset. This executable was already present on the Infogrid UI.

Results of experiment

After the insertion of the four rows, the SubmitDataset table looked as in Figure 5-12.

SubmitDataset				
Input	Executable	Status	JobID	Output
Select * from genomicdata Where rowNo<10	sequenceAlign			
Select * from genomicdata Where rowNo<100	sequenceAlign			
Select * from genomicdata Where rowNo<500	sequenceAlign			
Select * from genomicdata Where rowNo<1000	sequenceAlign			

Figure 5-12: Contents of SubmitDataset table before jobs have completed.

After the jobs had completed, the SubmitDataset table looked as in Figure 5-13.

SubmitDataset				
Input	Executable	Status	JobID	Output
Select * from genomicdata Where rowNo<10	sequenceAlign			
Select * from genomicdata Where rowNo<100	sequenceAlign			
Select * from genomicdata Where rowNo<500	sequenceAlign			
Select * from genomicdata Where rowNo<1000	sequenceAlign			
Select * from genomicdata Where rowNo<10	sequenceAlign	Cleared	sJmjcpJ1VEmDOkClazZnA	sequenceAlignOutput37
Select * from genomicdata Where rowNo<100	sequenceAlign	Cleared	Kp-F_QneqecI-GwRKKQfw	sequenceAlignOutput38
Select * from genomicdata Where rowNo<500	sequenceAlign	Cleared	zTWEM8gqhTCafDwf0uP7w	sequenceAlignOutput39
Select * from genomicdata Where rowNo<1000	sequenceAlign	Cleared	Dcsaf6bPVIdD44FIoEgbqg	sequenceAlignOutput40

Figure 5-13: Contents of SubmitDataset table after jobs have completed.

The table below shows the processing time for each job.

Number of rows	10	100	500	1000
Processing time (secs)	170	395	1323	2475

Table 5.4: Processing times for sets of jobs submitted via SubmitDataset table

In addition, when an input dataset was specified which had incorrectly typed data, the SubmitDataset table looked as in Figure 5-14.

SubmitDataset				
Input	Executable	Status	JobID	Output
Select * from wrong genomicdata	sequenceAlign			
Select * from wrong genomicdata	sequenceAlign	Error: expecting INTEGER, VARCHAR(1000), VARCHAR(65535), Not INTEGER, INTEGER, INTEGER		

Figure 5-14: Contents of SubmitDataset table after incorrectly typed data is submitted.

These results demonstrate the use of the SubmitDataset table to provide a relational interface through which datasets composed of entire tables or sets of rows from tables can be submitted for processing to the Grid.

5.2.8 Submission of jobs to the Grid via SubmitLargeDataset table

Aim of experiment

Demonstrate that insertion of rows into the SubmitLargeDataset table provides a relational interface through which datasets stored on SEs can be submitted for processing to the Infogrid prototype.

Form of experiment

The same executable as was used in the previous experiment, sequenceAlign, was used in this experiment. However, for this experiment jobs processing larger datasets containing 1000, 2000, 5000 and 10000 pairs of gene sequences were submitted to Infogrid. Four rows were inserted into the SubmitLargeDataset table, specifying LFNs which identified input files for the sequenceAlign application. The following SQL statements performed these operations:

```
INSERT INTO SubmitLargeDataset (LogicalFilename, Executable) VALUES
   ('lfn:sequences1000', 'sequenceAlign')
INSERT INTO SubmitLargeDataset (LogicalFilename, Executable) VALUES
   ('lfn:sequences2000', 'sequenceAlign')
```

```
INSERT INTO SubmitLargeDataset (LogicalFilename, Executable) VALUES
   ('lfn:sequences5000', 'sequenceAlign')
INSERT INTO SubmitLargeDataset (LogicalFilename, Executable) VALUES
   ('lfn:sequences10000', 'sequenceAlign')
```

These rows instructed the Infogrid prototype to submit a job which used an executable called sequenceAlign to process data contained in the files identified by the LFNs in the "LogicalFilename" column.

Results of experiment

After the insertion of the four tuples, the SubmitLargeDataset table looked as in Figure 5-15.

SubmitLargeDataset		
LogicalFilename	Executable	Status
lfn:sequences1000	sequenceAlign	
lfn:sequences2000	sequenceAlign	
lfn:sequences5000	sequenceAlign	
lfn:sequences10000	sequenceAlign	

Figure 5-15: Contents of SubmitLargeDataset table before jobs have completed.

After the jobs had completed, the SubmitLargeDataset table looked as in Figure 5-16. The output datasets shown in Figure 5-16 were successfully retrieved from the SE using the lcg-cp command. These results illustrate the use of the SubmitLargeDataset table to provide a relational interface through which datasets stored on SEs can be submitted for processing to the Infogrid prototype.

SubmitLargeDataset		
LogicalFilename	Executable	Status
lfn:sequences1000	sequenceAlign	
lfn:sequences2000	sequenceAlign	
lfn:sequences5000	sequenceAlign	
lfn:sequences10000	sequenceAlign	
lfn:sequences1000_OUTPUT	sequenceAlign	Cleared
lfn:sequences2000_OUTPUT	sequenceAlign	Cleared
lfn:sequences5000_OUTPUT	sequenceAlign	Cleared
lfn:sequences10000_OUTPUT	sequenceAlign	Cleared

Figure 5-16: Contents of SubmitLargeDataset table after jobs have completed.

5.3 Information System

R-GMA was originally designed for use as an information system for the gLite Grid middleware, and has previously been used by the WMS when performing matchmaking within the EDG Datagrid project. While the Infogrid prototype does not consult the information system when submitting a job to the Grid, the Infogrid CE does attempt to republish information stored on the Infogrid CE relating to its current state.

5.3.1 Demonstrate publication of information on Infogrid CE to R-GMA

Aim of experiment

Illustrate that R-GMA can be used by the Infogrid prototype to maintain current view of resources on Grid.

Form of experiment

Data produced by an information provider on the Infogrid CE was republished to R-GMA using the GIN application. The contents of the R-GMA tables containing

information related to the state of CEs on the Grid was compared to the output of the ldapsearch function executed on the Infogrid prototype.

Results of experiment

An extract of the results returned by the ldapsearch utility related to the CE when executed on the machine hosting the Infogrid WMS is shown below.

```
GlueCEStateEstimatedResponseTime:  0
GlueCEStateRunningJobs:  0
GlueCEStateStatus:  Production
GlueCEStateTotalJobs:  0
GlueCEStateWaitingJobs:  0
GlueCEStateWorstResponseTime:  0
GlueCEStateFreeJobSlots:  0
GlueCEPolicyMaxCPUTime:  0
GlueCEPolicyMaxObtainableCPUTime:  0
GlueCEPolicyMaxRunningJobs:  0
GlueCEPolicyMaxWaitingJobs:  0
GlueCEAccessControlBaseRule:  VO:gitest
```

GIN was configured on the Infogrid CE to insert data reflecting the current state of the CE every minute. When the GIN log on the CE (stored in the file /var/log/glite/rgma-gin.log) was examined over a period of time, a number of entries were written to the log indicating the execution of SQL statements that inserted data into various tables at one minute intervals. The following is an example of a logging entry indicating the execution of an SQL statement which inserts data into the GlueCE table:

```
2010-01-02 17:40:49,310 [Thread-0] DEBUG
org.glite.rgma.gin.ResilientProducer - Executing:
INSERT INTO GlueCE (ApplicationDir, JobManager, UniqueID,
MaxWallClockTime, RunningJobs, GatekeeperPort, GRAMVersion, MaxCPUTime,
Status, HostName, Priority, Name, TotalJobs, DefaultSE, MaxRunningJobs,
LRMSVersion, MaxTotalJobs, WorstResponseTime, EstimatedResponseTime,
TotalCPUs, LRMSType, GlueClusterUniqueID, FreeJobSlots, DataDir,
AssignedJobSlots, AssignedCPUs,InformationServiceURL, FreeCpus, WaitingJobs)
VALUES
(NULL, 'lcgpbs', 'infogrid7.testgrid:2119/jobmanager-lcgpbs-solovo', 0,
0,
'2119', NULL, 0, 'Production', 'infogrid7.testgrid', 1, 'solovo', 0, NULL,
0, 'not defined', 0, 0, 0, 0, 'torque', 'infogrid7.testgrid', 0, NULL,
0,
NULL, 'ldap://infogrid7.testgrid:2135/mds-vo-name=local,o=grid', 0, 0)
```

When a continuous query selecting all tuples published to the GlueCE table was executed on the machine hosting the Infogrid WMS, every minute a tuple indicating the current state of the Infogrid CE was retrieved. An example of some of the information contained in these tuples is shown below.

```
GlueCEUniqueID
UniqueID infogrid7.testgrid:2119/jobmanager-lcgpbs-gitest
Name gitest
GlueClusterUniqueID infogrid7.testgrid
TotalCPUs 0
```

```
GRAMVersion NULL
HostName infogrid7.testgrid
RunningJobs 0
TotalJobs 0
Status Production
FreeCpus 0
Priority 1
```

Examination of the GIN log file showed that GIN published rows to the following 10 tables at 60 second intervals:

- GlueCEAccessControlBaseRule
- GlueCEVOViewAccessControlBaseRule
- GlueCEVOView
- GlueCE
- GlueSubClusterSoftwareRunTimeEnvironment
- GlueCEContactString
- GlueCESEBind
- GlueSite
- GlueSubCluster
- GlueCluster

The Infogrid prototype is not configured to use data contained in R-GMA when performing matchmaking. However, the results obtained from this experiment illustrate that it is possible to republish data generated by an information provider on the Infogrid CE to R-GMA. This data stored in R-GMA could be used by a Grid scheduler when allocating jobs to resources on the Grid.

5.4 Computing Element

The Infogrid CE consists of an R-GMA consumer that accepts job invocation requests from the WMS, retrieves the input sandbox for a job from the WMS, and submits the job to a Condor cluster. This replaces the Globus gatekeeper and the Globus jobmanager components that are used by the gLite CE. LCMAPS pool account mapping is performed by the Infogrid CE, but at present the Infogrid CE does not use LCAS to perform authentication.

5.4.1 Demonstrate functionality of Infogrid CE

Aim of experiment

This experiment aimed to show that the Infogrid CE can perform the main functions performed by the regular gLite CE. The Infogrid CE uses R-GMA to listen for job submissions from the WMS and authenticate these submissions. It submits the job to a Condor cluster, and updates the job submission log on the WMS as the status of the job changes, which allows the standard gLite tools for monitoring the status of jobs and retrieving job output to be used for jobs executing on the Infogrid CE.

Form of experiment

An attempt was made to submit a job to the Infogrid CE without a valid credential that is required by R-GMA's security mechanisms. A bash script was also written which submitted 500 jobs to the Infogrid prototype using the standard glite-wms-job-submit command line tool. Using the glite-wms-job-get-logging-info tool, the time elapsed between submission on the UI and the job reaching the 'Done' state was measured for each of these jobs.

Results of experiment

The Infogrid CE used R-GMA's security mechanisms to restrict access to it. The R-GMA server was run in secure mode. Only authenticated users, possessing valid Grid Ireland certificates, could insert rows into or read data from the tables in the

R-GMA schema used by Infogrid. Attempting to submit jobs to the Infogrid CE without a valid X.509 credential resulted in R-GMA refusing to allow the operation and an error message being displayed, as shown below:

```
rgma> insert into CETable (SubmitJob) VALUES ("Qcygkkl3mGISMCdRy34kZw")
ERROR: Secure connection failed:  SSL_CTX_use_PrivateKey_file error
```

The Consumer on the Infogrid CE successfully retrieved all 500 rows inserted by the WMS into the CETable table. It retrieved the files required by each job from the WMS, and executed each of the jobs on a Condor cluster. The Infogrid CE also successfully notified the WMS of the changing state of the jobs as they proceeded through various stages of execution. After running the experiment, the log file used by the WMS to store information about jobs submitted to CEs showed events relating to the submission, execution and completion of all 500 jobs that were submitted to the Infogrid prototype. Figure 5-17 illustrates the time taken for each of the 500 jobs to complete.

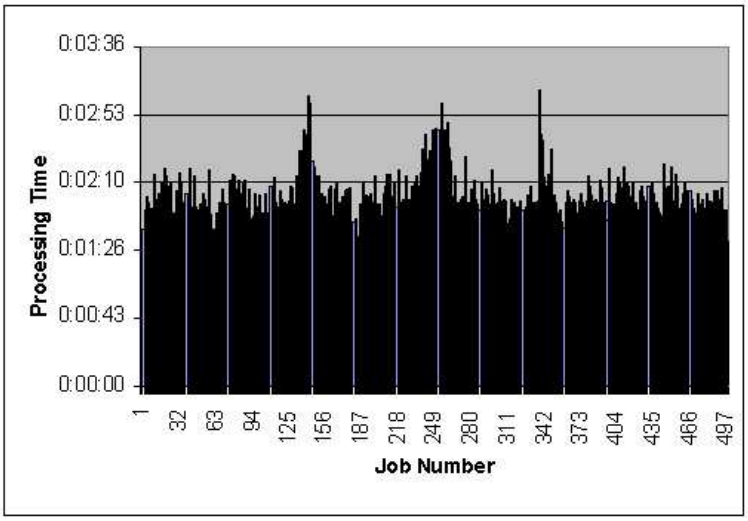


Figure 5-17: Time for jobs to complete on Infogrid CE

On average, each job took 1 minute 59.14 seconds to complete. The standard deviation of the series was 15.03 seconds. It took 2 hours 22 minutes and 50 seconds

for all 500 jobs to be processed, giving a throughput of approximately 3.496 jobs per minute.

5.5 Logging System

The logging system used by Grid middleware contains data related to events that occur on Grid components during their usage. APEL is an application that parses the data from the CE and worker node log files and publishes it to R-GMA. The Infogrid prototype attempted to use APEL to republish logging information on the Infogrid CE to R-GMA.

5.5.1 Demonstrate publication of logging information on CE to R-GMA using APEL

Aim of experiment

Determine if APEL can be used to insert logging information produced by the Infogrid CE into R-GMA.

Form of experiment

A configuration file for the APEL parser on the Infogrid CE was created which specified the URL, username and password for a MySQL database, and the various directories which contained logging files on the CE. A cron job was set up to execute the APEL parser daily using this configuration file. After the parser had executed, a query was executed on the tables in the MySQL database which contained the parsed data from the log files on the Infogrid CE to verify that the parse operation had successfully completed.

A configuration file was also created for the APEL publisher on the Infogrid CE which specified the URL, username and password for the MySQL database that contained the parsed logging data. A cron job was created which executed the APEL publisher daily after the parser had executed. A query was executed on the

LCGRecords table in R-GMA to verify that the APEL publisher had republished the data from the MySQL database.

Results of experiment

The following tables in the MySQL table should contain parsed logging data from the Infogrid CE:

- GKRecords - contains the parsed data from the gatekeeper logs (stored in files in the /var/log directory beginning with "globus.gatekeeper.log").
- MessageRecords - contains the parsed data from the CE message logs (stored in files in the /var/log directory).
- EventRecords - contains the parsed data from the LRMS logs (stored in /var/spool/pbs/server_logs/).

The table below illustrates the results obtained when the contents of these tables were retrieved.

Query	Results
SELECT * FROM GkRecords	4347 records returned
SELECT * FROM MessageRecords	0 records returned
SELECT * FROM EventRecords	1728 records returned

Table 5.5: Number of records stored in MySQL tables used by APEL parser

On a regular gLite CE, the gatekeeper writes records to the globus.gatekeeper and message logs in the /var/log directory. APEL also assumes that the LRMS used by a CE is either LSF or PBS. However, the Infogrid CE did not use the Globus gatekeeper and used Condor as its LRMS. The result was that when the APEL parser was run

on the Infogrid CE, out of date records from the globus.gatekeeper and PBS logs that were still present from when the Infogrid CE was a regular CE were parsed, and the messages log file did not contain records that would normally be inserted by the gatekeeper (hence the absence of data in the MessageRecords table). The APEL publisher attempted to perform a join on the GkRecords, MessageRecords and EventRecord tables and insert the result into the LCGRecords table in R-GMA. However, when the publisher executed the join query on the Infogrid CE no rows were returned due to the absence of records in the MessageRecords table. As a result, the LCGRecords table in R-GMA was also empty when it was queried.

APEL has been deployed successfully on existing Grid sites, however in order to work successfully when deployed on the Infogrid prototype, the Infogrid CE requires further work so that it writes the logging data that APEL requires to the /var/log/messages and /var/log/globus.gatekeeper files, and generates logs for the Condor LRMS that APEL can parse.

Chapter 6

Conclusions and Future Work

6.1 Original contributions of Infogrid prototype

The original contribution of the Infogrid prototype is using a single database technology to provide services implemented using a number of different applications in current Grid middleware. These services include

- Storing and retrieving data.
- Security (preventing unauthorised use of Grid software).
- Remote process invocation.

There are a number of Grid middlewares, each of which use a set of individual programs to implement these functions. The Infogrid prototype used a single database technology to perform these functions. Other technologies such as web services or security APIs were considered for the implementation of the Infogrid prototype. However, databases could be used to implement a wider range of the functions required by Grid middleware components than these. A database technology can allow data to be stored and retrieved, and provides security mechanisms to prevent unauthorised access to data. Database technology could also be used to provide a remote process invocation mechanism whereby applications "subscribe" to tables that serve as interfaces, and are automatically notified when remote clients insert rows into these tables.

A database technology could also be used to implement an interface to the Grid which performed type checking of input data for jobs as they are submitted. Existing Grid middleware does not perform this type checking, and there may be a period of time between job submission and a run time error occurring due to incorrectly typed data. A relational interface could report these errors to the user at the time the job is submitted.

A prototype of this Infogrid middleware was implemented. An existing Grid middleware, gLite, was modified by using a database technology to implement a number of the generic services it required. It was decided to use a combination of the gLite middleware and R-GMA, a component of the gLite middleware, to implement the Infogrid prototype for the following reasons:

- The gLite and R-GMA software is free to obtain.
- R-GMA is compatible with the gLite middleware, having been designed as one of its components, removing the risk of errors occurring during the creation and operation of the prototype due to software compatibility issues which may have occurred if separately developed database and Grid technologies were used.
- R-GMA provides publish-subscribe mechanisms which allow applications to automatically retrieve tuples when they are inserted into a table.
- R-GMA provides security mechanisms which are compliant with the International Grid Trust Federation's policies and guidelines on deploying Grids. These mechanisms can be used to restrict operations on tables to authorised and authenticated users.

GridDB and XG, described earlier in this thesis, are two other research projects which investigated implementing an interface to the Grid using database technology. However, they do not seek to implement other components of the Grid middleware using database technology. This thesis recognises further potential uses of database technology. These include implementing logging and information services, remote process invocation, etc., as demonstrated by the Infogrid prototype. The Infogrid

prototype also uses freely available, mature and well documented Grid and database technologies, unlike GridDB, which uses a database technology and query language unique to the project, and XG which uses a Grid middleware uniquely deployed within that project. Unlike the Infogrid prototype, neither GridDB nor XG provided security mechanisms to restrict access to the underlying Grid.

6.2 Evaluation of Infogrid prototype

The table below illustrates the existing components of the gLite middleware that the Infogrid prototype implemented using R-GMA.

Functionality : using R-GMA as...	Achieved	Component Replaced
Information System for the Grid	Yes	MDS
Logging System	Yes	Log files
RPC Protocol	Yes	GRAM
Gatekeeper on CE	Yes	Globus
UI interface	Yes	UI

Table 6.1: Grid Functionality using R-GMA

6.2.1 Using R-GMA as an Information System

R-GMA was originally designed as an information system for the gLite Grid middleware. R-GMA has previously been used as the information system by the WMS within the EDG Datagrid project. There are a number of publications detailing its functionality and performance [82], [83], [84]. Although the Infogrid prototype successfully used R-GMA to store information published by the Infogrid CE, its WMS did not consult R-GMA in order to perform matchmaking. While the WMS is capable of accessing information published to R-GMA using GIN, the schema representing the

information system used by the WMS (the GLUE schema) differs slightly from the schema used by R-GMA. The Infogrid WMS does not perform translations between the two schemas, therefore matchmaking was disabled in the Infogrid prototype.

6.2.2 Using R-GMA as a Logging System

Logging data is generated by events that occur during the operation of Grid middleware. In addition to using software written by the producers of R-GMA (APEL) to republish logging information to R-GMA, the Infogrid prototype published logging data to a number of tables in the R-GMA schema. A consequence of replacing the Gatekeeper component of the Infogrid CE with an R-GMA Consumer however was that the APEL program no longer worked successfully, as the CE's gatekeeper no longer produced entries in logging files that were required for APEL to function. Modifying the Infogrid CE so that it either publishes data to particular files as APEL expects or that it publishes logging data directly to R-GMA is a possible avenue for future work on the Infogrid prototype.

6.2.3 Using R-GMA to implement Remote Procedure Calls

The Infogrid prototype used R-GMA as a mechanism which could be used by Grid middleware components to interact with each other, by invoking operations remotely and exchanging data. Examples of this include:

- Remote clients submitting data to the Grid for processing by inserting it into an active table.
- The Infogrid WMS submitting a job to the Infogrid CE by inserting data into the CETable table.
- The Infogrid CE sending logging information regarding the progress of jobs it executes to the WMS by inserting data into the CondorLogEntries table.

R-GMA queries were also used to retrieve information stored on remote machines. For example, although the Infogrid WMS did not query R-GMA in order to perform

matchmaking, Section 5.3.1 of this thesis describes how in order to verify that the Infogrid CE was successfully publishing information to R-GMA, an R-GMA client was instantiated on the WMS of the Infogrid prototype and used to retrieve the information that was published by the Infogrid CE.

6.2.4 Using R-GMA as a Gatekeeper on the Infogrid CE

The Infogrid prototype had a uniquely "Infogrid" CE which used R-GMA to listen for incoming jobs, instead of the Globus gatekeeper used by the standard gLite CE. R-GMA's security mechanisms prevented unauthorised submissions to the Infogrid CE, as shown in Section 5.4.1 of this thesis. Although the Infogrid CE performs the major tasks that the regular CE does (listen for jobs, authenticate submissions, map users to local accounts, invoke jobs on local resources, etc.) further work is required to ensure that all functionality of the standard CE is implemented. An example of such work is fixing the problems encountered when trying to republish logging information on the CE to R-GMA using APEL, as described previously.

6.2.5 Using R-GMA as a relational interface for job submission

The Infogrid prototype allowed jobs to be submitted for processing by inserting tuples into the following tables:

- Active tables
- SubmitDataset table
- SubmitLargeDataset table

As well as demonstrating that R-GMA can be used as an interface for job submission, the first two types of tables allowed type checking to be performed on input data for jobs as they were submitted. However, these two tables assumed a particular structure for the input data, whereas the SubmitLargeDataset table allows jobs to be submitted where the application deals with the structure of the input data. In

addition, the regular job submission tools for the gLite middleware also work with the Infogrid prototype.

Examples of Grid applications which would be suitable for execution on Infogrid were presented in Section 3.3.1. These included sequence alignment operations performed on gene sequences and analysis of astronomy data gathered from a sky survey. However, some Grid applications would not be suitable for execution on the Infogrid prototype. The experiments performed in Section 5.2.3 highlighted problems that occurred when large numbers of tuples were inserted into active tables. This problem will occur when large amounts of data are inserted into a table in R-GMA using a single Producer.

6.3 Experiments on Infogrid prototype

The experiments conducted on the Infogrid prototype sought to verify that it implemented the functionality outlined in the hypothetical Infogrid described in Chapter 3 of this thesis. The experiments verified that the Infogrid prototype provided a number of services using R-GMA that are performed using separate applications in the existing Grid middleware. The effectiveness of using R-GMA to perform type checking was demonstrated. Section 5.4.1 illustrated that the Infogrid CE was both reliable (there were no failures out of 500 jobs) and performed well (the average processing time over the sample of 500 jobs was similar to the regular gLite CE). I The experiments highlighted that the use of memoisation could reduce the overall processing time for a set of data inserted into an active table, if the time saved by not executing recurring tuples that had been previously executed was greater than the time it took the Infogrid application to perform a history query to establish if the input values in each tuple had been previously processed. The experiments also verified that the submission of data from active tables in chunks could reduce the overall processing time for a dataset.

6.4 Possible future work

The Infogrid middleware that resulted from this thesis was a "proof-of-concept" prototype that was intended to illustrate the viability of using a single database technology to perform multiple functions that are implemented using several separate applications in existing Grid middlewares. Were a production quality version of the Infogrid prototype to be constructed, a number of shortcomings in the prototype would need to be addressed.

6.4.1 Investigate and address shortcomings of R-GMA

The conceptual Infogrid introduced in chapter 3 envisions tables where rows containing input data are updated to include output data when this output is available. R-GMA does not allow row updates, which results in entire new rows having to be inserted in some tables which only differ from an existing row in that output columns have different values. This approach leads to input data in tables being duplicated. As well as making the contents of tables potentially confusing, the extra storage space required to store this duplicated data may be significant. This is especially likely given the large amounts of data Grids are typically used to process.

R-GMA also has a retention period associated with rows inserted into it. In some cases, it is desirable that rows inserted into tables persist indefinitely (for example, rows inserted into the active tables metadata table). While retention periods can be set to a large value for rows, it would be preferable that some rows inserted into tables could persist indefinitely without requiring the user to specify a large retention period. Investigating whether these shortcomings could be overcome, either by modifying R-GMA or using a different database technology could be an avenue for future work on the Infogrid prototype.

6.4.2 How are job submission tables for CEs defined?

When constructing the Infogrid prototype, a table was defined in the R-GMA schema (CETable) which served as an interface to the Infogrid CE. If further CEs were to

be added to the prototype, additional tables would have to be added to the R-GMA schema to represent interfaces to these CEs. Administrator tools which allowed users to create such tables would have to be created in future.

6.4.3 Enable all features of existing middleware components in Infogrid prototype

In the course of modifying existing Grid components to create the Infogrid prototype, some of their features were disabled. For example, as mentioned previously the Infogrid CE does not produce all the logging information that the regular CE produces, which led to problems occurring when APEL was run on the Infogrid CE. In addition, the Infogrid CE does not use LCAS. The current implementation of LCAS requires the CE to provide it with an object created by the GSS API, which is used by the regular CE to authenticate submitted jobs. As the Infogrid CE does not use the GSS API to perform this role, it cannot invoke the functionality of LCAS to enforce local policies. Another example of a disabled feature in an Infogrid component is matchmaking in the Infogrid WMS. Modifying the Infogrid WMS so that the matchmaking process is carried out using information published to R-GMA by Grid resources is feasible, but perhaps should be investigated in the context of replacing the WMS entirely.

The job submission interfaces offered by the Infogrid prototype do not offer all the functionality offered by current Grid middleware job submission interfaces. For example, requirements that the Grid resources that will execute a job must meet (such as available memory, software installed, etc.) cannot be specified when submitting jobs to the Infogrid prototype by inserting rows into either an active table or the SubmitDataset/SubmitLargeDataset tables. Environment variables cannot be specified for jobs. Neither can jobs be submitted which require both command line arguments and files as input. Extending the Infogrid job submission interface's database tables in order that they provide a fuller range of functionality is another area for future work, and could possibly be made familiar to SQL users via its "WHERE" syntax.

6.4.4 Implement further functionality of gLite middleware using R-GMA

The use of a database technology to implement services required by Grid middleware could be applied more generally than in the Infogrid prototype. The prototype could be extended so that R-GMA is used to implement additional applications. An interface to the SE component could have also been implemented using R-GMA. In theory, the GFAL API used to provide an interface to the SE could have been replaced with the R-GMA API, with operations on storage elements and the File and Replica Catalog being performed by inserting rows into tables in the R-GMA schema. Although time did not permit this, the success in using R-GMA as a mechanism for invoking the functionality of other Grid components in the Infogrid prototype illustrates the viability of invoking the functionality of the SE and File and Replica Catalog in the same manner. Another example of how database technology could be used more generally than in the Infogrid prototype is implementing a metadata catalog (such as AMGA [85]) in the Infogrid prototype using R-GMA. Separate research has also investigated the use of R-GMA to implement a file system [86]. Further work could investigate integrating this file system with the Infogrid prototype.

6.4.5 Enable executables to be transferred to UI

At present, the Infogrid prototype assumes that when active tables are created as interfaces to executables, or when rows are inserted into the SubmitDataset/SubmitLargeDataset tables that invoke a particular executable, the executable is present in a particular directory on the UI. Future work could investigate how to allow users to upload executables to this directory, or to invoke executables located on other machines.

6.4.6 Improve error reporting of relational job submission interfaces

At present, apart from data typing errors, errors caused by issues such as the executable required by a job not being available on the UI or a Grid component malfunc-

tioning are not reported by the job submission interfaces offered by Infogrid. While it is possible to use a command line tool such as glite-job-get-status on the Infogrid prototype to discover errors that have occurred in the execution of these jobs, future work on the Infogrid prototype could involve modifying the job submission interfaces so they can also display these errors.

6.4.7 Modify Table Updater to handle name clashes between active tables and other executables

When the Table Updater is notified that a job has completed, it checks if the job is a result of an insertion into an active table, the SubmitDataset table, or the SubmitLargeDataset table. This assumes that there are no cases where an active table has the same name as an executable invoked by jobs submitted via the SubmitDataset/SubmitLargeDataset tables, or the same executable is invoked by separate rows inserted into the SubmitDataset and SubmitLargeDataset tables. The Infogrid prototype would need further modification in order to be able to process the output of jobs in these cases.

6.4.8 Unify the submission tables

For prototyping simplicity the three input use cases presented in the "ideal" Infogrid architecture in Chapter 3 of this thesis were handled separately via active, SubmitDataset and SubmitLargeDataset tables. Now the concepts are proven, these submission tables should be unified as active tables, with a single enhanced active tables metadata table. This would require addition of "IN_Dataset", "OUT_Dataset", "IN_LargeDataset", "OUT_LargeDataset" column tags to extend the valid input and output column content of an active table. If this were done, it would allow mixed input and output data cases (e.g. "IN_LargeDataset" in and "OUTPUT" out), common submission webservices, APIs, etc, and common support for data streams.'

6.4.9 Dynamically vary chunk parameters on active tables

As was remarked upon when describing the experiments performed on the Infogrid prototype, it is not possible to alter the chunking and chunk timeout limits of active tables after they have been created. This required multiple active tables performing the same function to be created when running experiments using different chunk limits/timeouts. The Infogrid prototype requires further modification in order that these settings can be altered for existing active tables after their creation. Rules could be defined that would enable users, who know their data best, to optimize when they are used.

6.4.10 Dynamically enable memoisation

As with chunk limits/timeouts, in the Infogrid prototype it was not possible to enable/disable memoisation on an active table after the time of its creation. It is conceivable that support for the dynamic enabling of the memoisation feature of active tables could be added to the Infogrid prototype, where the memoisation could be dynamically enabled by users (perhaps via SQL statements). Alternatively user-defined rules could specify under what circumstances memoisation is to be enabled on an active table. This would enable users to optimize data processing by specifying when memoisation is to be used.

6.4.11 Improve performance of R-GMA history queries used by Infogrid

When running experiments involving memoisation checks, it was found that R-GMA history queries required a minimum of 1 second to run. When instantiating a Consumer issuing a history query, it is necessary to specify the length of time the Consumer should spend trying to retrieve tuples matching the query. The code below illustrates such an example.

```
Consumer c = factory.createConsumer(ti, "SELECT * FROM ActiveTable
```

```
where Input=999",QueryProperties.HISTORY);
c.start(new TimeInterval(1, Units.SECONDS));
```

The first line of the code creates a Consumer issuing a query on an active table. The second line of the code instructs the Consumer to execute that query for 1 second. Although the R-GMA Java API allows time intervals to be specified in milliseconds, specifying a time interval of less than 1000 milliseconds will cause a run time error to occur. Using a version of R-GMA which allowed history queries to execute quicker would improve the performance of the Infogrid prototype.

6.4.12 Integrate Infogrid CE with regular WMS

Another avenue for future investigation is modifying the component of the regular gLite WMS that enables submission to CEs so that a queue for the Infogrid CE is added. This would enable Grids to be constructed that consist of both regular and Infogrid CEs.

6.4.13 Enable checkpointing of jobs on Infogrid CE

The CREAM CE[87], which is another type of CE, allows jobs to be suspended and resumed. This feature, called checkpointing, could also be implemented by the Infogrid CE, as its underlying LRMS is Condor, which allows users to issue commands which perform checkpointing on jobs. Providing a means for users to pause the execution of a job running on an Infogrid CE in this fashion, and to resume it at a later stage, is another area of potential research.

6.4.14 Investigate influencing factors on effectiveness of chunking in active tables

The degree to which chunking data from active tables effects the overall processing time for a set of rows is affected by the following factors:

- Time to execute active table function for one row.

- Number of worker nodes on Grid.
- Total number of rows inserted into an active table.
- Rate at which rows are inserted into an active table.
- Chunk size.

Further research could more thoroughly investigate the effect that a wider range of values for these variables would have on the processing time for data than was done in this thesis.

6.5 Summary

The experiments show that the Infogrid prototype described in Chapter 4 implements the functionality of the hypothetical Infogrid outlined in chapter 3. However, there are a number of issues in the implementation of the prototype that need to be resolved. Some of the functionality in components of the Infogrid prototype needs to be enabled (for example, matchmaking in the WMS, producing logging entries in the CE, allowing job requirements to be specified and enabling error reporting in the tables used for job submission). The inability of R-GMA to perform row updates, and the problems encountered when the tuple storage used by Producers became full when large amounts of data were inserted into tables also require further work to resolve. Nevertheless, R-GMA was successfully used to replace a number of protocols in the gLite middleware, reducing the size of the software stack required to implement a Grid. The results were promising and lend credence to the central hypothesis of this thesis.

Bibliography

[1] *Condor Instruction Manual.* http://www.cs.wisc.edu/condor/manual/v7.0/3_1Introduction.html.

[2] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. *A Resource Management Architecture for Metacomputing Systems*, volume 1459. Lecture Notes in Computer Science, 1998.

[3] Radu Sion, Ramesh Natarajan, Inderpal Narang, and Thomas Phan. *XG: A Grid-Enabled Query Processing Engine*, volume 3896. Lecture Notes in Computer Science, 2006.

[4] David Liu and Michael Franklin. *GridDB: a data-centric overlay for scientific grids*, volume 30. Proceedings of the Thirtieth international conference on Very large data bases, 2004.

[5] *R-GMA Architecture Diagram.* http://www.r-gma.org/arch-virtual.html.

[6] *R-GMA Producer Diagram.* http://www.r-gma.org/arch-producers.html.

[7] *R-GMA Consumer Diagram.* http://www.r-gma.org/arch-consumers.html.

[8] Heinz Stockinger. *Defining the Grid: a snapshot on the current view.* The Journal of Supercomputing, 2007.

[9] *Web Service Definition Language.* www.w3.org/TR/wsdl, 2001.

[10] Andreas Fuchsberger. *GSS-API*, volume 2. Information Security Technical Report, 2001.

[11] *Java Message Service.* http://java.sun.com/products/jms/.

[12] *Java Security API.* http://java.sun.com/javase/technologies/security/.

[13] *Bouncy Castle Crypto APIs.* http://www.bouncycastle.org/.

[14] *Particle Physics Data Grid.* http://www.griphyn.org/, 2008.

[15] *International Virtual Data Grid Laboratory.* http://www.ivdgl.org/, 2008.

[16] *The TeraGrid Project.* http://www.teragrid.org/, 2008.

[17] *Distributed European Infrastructure for Supercomputing Applications.* http://www.deisa.org/, 2008.

[18] K. Miura. *Overview of Japanese Science Grid Project NAREGI*, volume 3. Progress in Informatics, 2006.

[19] *D-Grid Initiative.* http://www.dgrid.de/, 2008.

[20] D. Britton. *GridPP: A Project Overview.* Proceedings of UK e-Science All Hands Conference, 2003.

[21] *Grid Ireland.* http://www.grid.ie, 2008.

[22] *Enabling Grids for eScience in Europe.* http://www.eu-egee.org, 2008.

[23] *gLite: Lightweight Middleware for Grid Computing.* http://glite.web.cern.ch/glite, 2008.

[24] *The BABAR Collaboration*, volume 479. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 2002.

[25] Arun Krishnan. *GridBLAST: a Globus-based high-throughput implementation of BLAST in a Grid computing framework*, volume 17. Concurrency and Computation: Practice & Experience, 2005.

[26] Chao-Tung Yang, Tsu-Fen Han, and Heng-Chuan Kan. *G-BLAST: a Grid-based solution for mpiBLAST on computational Grids*, volume 21. Concurrency and Computation: Practice & Experience, 2009.

[27] G. Trombetti, I. Merelli, A. Orro, and L. Milanesi. *BGBlast: A BLAST Grid Implementation with Database Self-Updating and Adaptive Replication*, volume 126. Stud Health Technol Inform, 2007.

[28] S. Althschul, W. Gish, W. Miller, E. Myers, and D.J. Lipman. *Basic Local Alignment Search Tool*, volume 215. Journal of Molecular Biology, 1990.

[29] Joseph Jacob, Daniel Katz, Craig Miller, Harshpreet Walia, Roy Williams, S. George Djorgovski, Matthew Graham, Ashish Mahabal, Jogesh Babu, Daniel Vanden Berk, and Robert Nichol. *GRIST: Grid-based Data Mining for Astronomy*, volume 347. Astronomical Data Analysis Software and Systems, 2004.

[30] S.G. Djorgovski, C. Baltay, A.A. Mahabal, A.J. Drake, R.Williams, D. Rabinowitz, M.J. Graham, C. Donalek, E. Glikman, A. Bauer, R. Scalzo, N. Ellman, and J. Jerke. *The Palomar-Quest Digital Synoptic Sky Survey*, volume 329. Astronomical Notes, 2008.

[31] Douglas Thain, Todd Tannenbaum, and Miron Livny. *Distributed Computing in Practice: The Condor Experience*, volume 17. Concurrency and Computation: Practice and Experience, 2005.

[32] Ian Foster and Carl Kesselman. *Globus: A Metacomputing Infrastructure Toolkit*, volume 11. International Journal of Supercomputer Applications, 1997.

[33] Rajesh Raman, Miron Livny, and Marvin Solomon. *Matchmaking: Distributed resource management for high throughput computing.* Proceedings of the seventh IEE IHPDC, 2008.

[34] Steven Fitzgerald, Ian Foster, Carl Kesselman, Gregor Von Laszewski, Warren Smith, and Steven Tuecke. *A directory service for configuring high-performance distributed computations.* Proc.HPDC6, 1997.

[35] *Replica Location Service.* http://www.globus.org/toolkit/data/rls/, 2008.

[36] W. Allcock. *GridFTP: Protocol extensions to FTP for the Grid.* Global Grid ForumGFD-R-P.020, 2003.

[37] Karl Czajkowski, Ian Foster, Nicholas Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. *A Resource Management Architecture for Metacomputing Systems.* IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[38] Von Welch Jason Novotny, Steven Tuecke. *An Online Credential Repository for the Grid: MyProxy.* 10th IEEE International Symposium on High Performance Distributed Computing, 2001.

[39] *SimpleCA security tool.* http://www.vpnc.org/SimpleCA/.

[40] *Resource Specification Language.* http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html/.

[41] *LHC Computing Grid.* http://lcg.web.cern.ch/LCG/.

[42] *The GriPhyN Virtual Data Toolkit.* http://www.lsc-group.phys.uwm.edu/vdt, 2008.

[43] *Global Grid Forum.* http://www.ggf.org, 2008.

[44] Ian Foster, Carl Kesselman, Jeffrey Nick, and Steven Tuecke. *Grid Services for Distributed System Integration*, volume 35. IEEE Computer, 2002.

[45] P. Saiz, L. Aphecetche, P.Bun, R. Piska, J.E. Revsbech, and V. Sego. *AliEn - ALICE Environment on the Grid.* Proceedings of the 8th Int.Workshop on Advanced Computing and Analysis Techniques in Physics Research, 2002.

[46] J. Casey J.P. Baud. *Evolution of LCG-2 Data Management.* Computing in High Energy and Nuclear Physics, 2004.

[47] *Berkeley Database Information Index.* http://twiki.cern.ch/twiki//bin/view/EGEE/BDII, 2008.

[48] *Job Submission Description Language (JSDL) Specification v1.0.* http://www.ogf.org/documents/GFD.136.pdf.

[49] R. Barbera. *The GENIUS Grid Portal.* Computing in High Energy and Nuclear Physics, 2003.

[50] Miroslaw Kupczyk, Rafal Lichwala, Bartek Palak, Marcin, Plociennik, Pawel Wolniewicz, and Norbert Meyer. *Roaming Access and Migrating Desktop.* Proceedings of The 2nd Cracow Grid Workshop, 2002.

[51] *GridSphere.* http://www.gridsphere.org/gridsphere/gridsphere, 2008.

[52] Gareth Lewis, Gergely Sipos, Florian Urmetzer, Vassil Alexandrov, and Peter Kacsuk. *The collaborative p-grade grid portal.* Computational Science - ICCS 2005, 2005.

[53] Mario Antonioletti, Malcolm Atkinson, Rob Baxter, Andrew Borley, Neil P. Chue Hong, Brian Collins, Neil Hardman, Alastair C. Hume, Alan Knox, Mike Jackson, Amy Krause, Simon Laws, James Magowan, Norman W. Paton, Dave Pearson, Tom Sugden, Paul Watson, and Martin Westhead. *The design and implementation of grid database services in OGSA-DAI*, volume 17. Concurrency and Computation: Practice and Experience, 2005.

[54] Edgardo Ambrosi, Antonia Ghiselli, and Giuliano Taffoni. *GDSE: A new data source oriented computing element for grid.* Proc. Parallel and Distributed Computing and Networks, 2006.

[55] H. Enke, M. Steinmetz, T. Radke, A. Reiser, T. Roblitz, and M. Hogqvist. *AstroGrid-D: Enhancing Astronomic Science with Grid Technology.* Proc. of the German e-Science Conference, 2007.

[56] Whitfield Diffie and Martin Hellman. *New Directions in Cryptography*, volume 6. IEEE Transactions on Information Theory, 1976.

[57] *Information Technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks.* http://www.itu.int/rec/T-REC-X.509-200508-I, 2005.

[58] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. *A Security Architecture for Computational Grids.* Proc. 5th ACM Conference on Computer and Communications Security Conference, 1998.

[59] *International Grid Trust Federation, Profile for Traditional X.509 Public Key Certification Authorities with secured infrastructure (Version 4.0).* http://www.eugridpma.org/guidelines/IGTF-AP-classic-20050930-4-0.html, 2005.

[60] *European Policy Management Authority for Grid Authentication.* http://www.eugridpma.org/, 2008.

[61] T. Genovese. *Profile for Short Lived Credential Services X.509 Public Key Certification Authorities with secured infrastructure.* http://www.tagpma.org/files/IGTF-AP-SLCS-20051115-1-1.pdf, 2008.

[62] *Asia Pacific Grid Policy Management Authority.* http://www.apgridpma.org/, 2008.

[63] Andrew W. Cooke, Alasdair J. G. Gray, Werner Nutt, James Magowan, Manfred Oevers, Paul Taylor, Roney Cordenonsi, Rob Byrom, Linda Cornwall, Abdeslem Djaoui, Laurence Field, Steve Fisher, Steve Hicks, Jason Leake, Robin Middleton, Antony J. Wilson, Xiaomei Zhu, Norbert Podhorszki, Brian A. Coghlan, Stuart Kenny, David O'Callaghan, and John Ryan. The relational grid monitoring architecture: Mediating information about the grid. *J. Grid Comput.*, 2(4):323–339, 2004.

[64] P.Th. Eugster, P.A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe, 2003.

[65] Fabrizio Gagliardi, Bob Jones, Mario Reale, and Stephen Burke. *European DataGrid Project: Experiences of Deploying a Large Scale Testbed for E-science Applications*, volume LNCS 2459. Performance Evaluation of Complex Systems: Techniques and Tools, 2002.

[66] Edgar Codd. *A Relational Model of Data for Large Shared Data Banks*, volume 13. Communications of the ACM, 1970.

[67] Brian Tierney, Ruth Aydt, Dan Gunter, Warren Smith, Valerie Taylor, Rich Wolski, and Martin Swany. *A Grid Monitoring Architecture.* GGF, 2001.

[68] *Open Grid Forum.* http://www.ogf.org/, 2008.

[69] *StreamBase: StreamSQL online documentation.* http://streambase.com/developers/ docs/latest/streamsql/index.html.

[70] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: Semantic foundations and query execution. Technical report, VLDB Journal, 2003.

[71] Hamish Taylor Lisha Ma, Werner Nutt. Condensative stream query language for data streams. In *ADC '07: Proceedings of the eighteenth conference on Australasian database*, pages 113–122, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[72] Rob Byrom, Brian Coghlan, Andy Cooke, Roney Cordenonsi, Linda Cornwall, Martin Craig, Abdeslem Djaoui, Alastair Duncan, Steve Fisher, Alasdair Grayand Steve Hicks, Stuart Kenny, Jason Leake, Oliver Lyttleton, James Magowan, Robin Middleton, Werner Nutt, David O.Callaghan, Norbert Podhorszki, Paul Taylor, John Walk, and Antony Wilson. *Fault Tolerance in the R-GMA Information and Monitoring System.* Lecture Notes in Computer Science, 2005.

[73] Andreozzi S, Burke S, Donno F, Field L, Fisher S, Jensen J, Konya B, Litmaath M, Mambelli M, Schopf J, Viljoen M, Wilson A, and Zappi R. 2007 glue schema specification - version 1.3 - draft 3, 2007.

[74] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, Http://www. Ggf. Org/ogsi wg K. Czajkowski, P. Vanderbilt (eds.), I. Foster Anl, J. Frey Ibm, C. Kesselman Usc/isi, D. Snelling, Fujitsu Labs, and P. Vanderbilt Nasa. Open grid services infrastructure (ogsi), 2003.

[75] S. Kille M. Wahl, T. Howes. *Lightweight Directory Access Protocol.* IETF RFC 2251, http://rfc-editor.org/rfc/rfc2251.txt, 1997.

[76] Rob Byrom, Brian Coghlan, Andrew W Cooke, Roney Cordenonsi, Linda Cornwall, Abdeslem Djaoui, Laurence Field, Steve Fisher, Steve Hicks, Stuart Kenny, Jason Leake, James Magowan, Werner Nutt, David O'Callaghan, Norbert Podhorszki, John Ryan, Manish Soni, Paul Taylor, and Antony J Wilson. *Relational Grid Monitoring Architecture (R-GMA).* UK e-Science All Hands Conference, Nottingham, 2003.

[77] The ldap data interchange format (ldif)- technical specification.

[78] *CondorG.* http://www.cs.wisc.edu/condor/condorg/, 2008.

[79] *Site authorisation and enforcement services: LCAS, LCMAPS and gLExec.* http://www.nikhef.nl/grid/lcaslcmaps/, 2008.

[80] Site authorisation, LCMAPS enforcement services: LCAS, and gLExec. http://www.nikhef.nl/grid/lcaslcmaps/, 2008.

[81] *Random DNA Generator.* http://www.faculty.ucr.edu/ mmaduro/random.htm.

[82] R. Byrom, D. Colling, S. M. Fisher, C. Grandi, P. R. Hobson, P. Kyberd, B. MacEvoy, J. J. Nebrensky, and S. Traylen. Performance of r-gma for monitoring grid jobs for cms data production. *Nuclear Science Symposium Conference Record*, 2:860–864, 2005.

[83] D. Bonacorsi, D. Colling, L. Field, S. Fisher, C. Grandi, P. R. Hobson, P. Kyberd, B. MacEvoy, J. J. Nebrensky, H. Tallini, and S. Traylen. Scalability test of r-gma based grid job monitoring system for cms monte carlo data production. *IEEE Transactions on Nuclear Science*, 51:3026–3029, 2004.

[84] P Bhatti, A Duncan, S M Fisher, M Jiang, A O Kuseju, A Paventhan, and A J Wilson. Building a robust distributed system: some lessons from r-gma. *Journal of Physics: Conference Series*, 119(6):062016, 2008.

[85] Nuno Santos Nuno. Distributed metadata with the amga metadata catalog. In *In Workshop on NextGeneration Distributed Data Management - HPDC-15*, 2006.

[86] Soha Maad, Brian Coghlan, Geoff Quigley, John Ryan, Eamonn Kenny, and David O'Callaghan. Towards a complete grid filesystem functionality. *Future Gener. Comput. Syst.*, 23(1):123–131, 2007.

[87] Cristina Aiftimiei, Paolo Andreetto, Sara Bertocco, Simone Dalla Fina, Alvise Dorigo, Eric Frizziero, Alessio Gianelle, Moreno Marzolla, Mirco Mazzucato, Massimo Sgaravatto, Sergio Traldi, and Luigi Zangrando. Design and implementation of the glite cream job management service. *Future Generation Computer Systems*, 26(4):654 – 667, 2010.