A pseudo object database model and its applications on a highly complex distributed architecture

Peter Colclough, Gilles Mathieu
STFC, UK
biton@compuserve.com, Gilles.Mathieu@stfc.ac.uk

Abstract

Developing, maintaining and accessing complex information repositories are some of the main concerns of today's information society. In huge distributed systems, guaranteeing good performances and data integrity is a real challenge when using standard relational databases. This paper presents a database conception methodology that addresses these concerns under the form of an "RDBMS independent" database designed in an object fashion, while allowing for standard SQL access for those applications that require it. After presenting the general concept and model and showing that it can be used and implemented on any database system, we will describe it in detail and focus on examples of current uses both for commercial and academic applications. The paper will highlight the benefits of using this model in one particular highly complex, distributed database environment in describing its application to the central authoritative information repository of the EGEE and WLCG worldwide computing Grids.

1. Introduction

1.1 General Scope

The model described in this paper is a "methodology", and as such will fit with any current or future relational database, protecting one's investment in database technology; it is also non application specific, so will work with any project - past, present or future. It has already been implemented against Informix/DB2, Oracle, MySQL and PostgresSQL. The design of this methodology allows for any size or distribution requirements of the database.

1.2 Application context

As described in section 4, the model has been successfully implemented in different contexts. The main use case described in this paper is, however, only at a development stage at the time of writing.

It consists of a central, authoritative database known as GOCDB [1] which contains topology information about two worldwide Grid computing projects: Enabling Grids for E-science (EGEE) [2] and the Worldwide LCG Computing Grid (WLCG) [3]. GOCDB is hosted at RAL-STFC, UK.

It stores information about regions, countries, sites, nodes, services and users, and links this information together in a logical way. This service consists of two parts: a database and a web portal, known as GOC portal [4].

GOCDB is used by many other operational tools, notably monitoring, availability calculation, contacts definition and accounting.

1.3 Needs and motivations

Maintaining highly complex relational databases in a distributed environment proves to be a very hard task. Increasing the number of constraints induces an obvious lack of flexibility and a very low scalability. Increasing the size of the database as well as trying to distribute it leads to proportional increase in model complexity with direct consequence on performances and maintainability.

In our context, the need of having a highly flexible model arises when working on distributing GOCDB to follow EGEE operational needs (see 4.2).

The model description in this paper has been adapted to fit in with the GOCDB schemas, which will be installed over multiple, geographically diverse servers.

2. Related work

The underlying requirement for GOCDB is to allow for a uniform data access across multiple grids, as well as the ability to have local applications stored within the same environment (see 4.2). These local applications may then be required to be accessed by a wider audience, so migration 'across the Grid' is also required.

In order to examine why this model has been proposed, we will look at the other options, bearing in mind cost and usability.

2.1 Standard Relational Model

Relational models [5] are very difficult to maintain across grids, as data would need to be updated over a wide geographic area. Constraints as to table/database naming would be required to be maintained, as would a set of standard data access routines. This would become unmanageable in a heterogeneous grid.

2.2 Object Databases

The normal method used in most Object Databases [6] is to store the information in a 'serialized' string. This requires all searches to split the string prior to actually determining if the tuple is required by the query. The time factor would kill this type of application, as it would not make use of the databases' query optimizer.

2.3 True Object Databases

This would require each grid to install the latest same version of the database. Not all Object databases are OS independent, and therefore it may be that a given Grid could not run the same DB as the rest of the participating sites.

This would also require that all applications are written using a compatible development language, which again cannot be guaranteed over a heterogeneous grid.

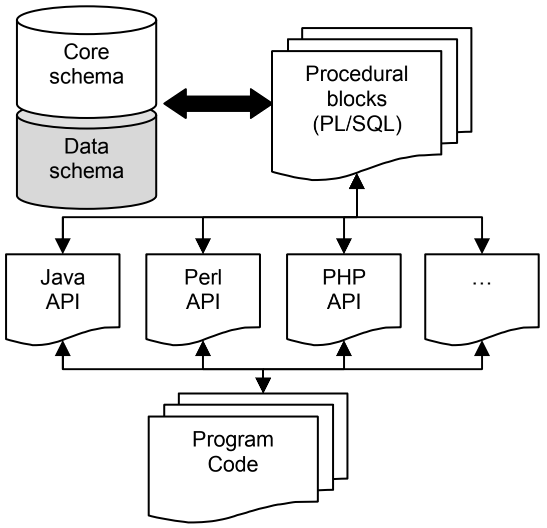
2.4 Google Big Table

Big Table is essentially a large table split over many sites [7]. The data is indexed on a row/column/timestamp basis, with the actual data being serialized in a single column. The data can be stored in memory, or on disk. The basic concept is that the data is held locally to the application, and the application makes the decision where to find its data.

This is insufficient for our needs on two counts. Firstly the serialized data will be an issue (see 2.3), and secondly most of the data is not local, but will be required on a global scale.

3. The model

The central methodology of this model consists of a core set of database tables, and a set of API calls, which can either be client side program code, or DBMS Stored Procedures (See Fig 1), which act over the project data tables.



(Fig. 1 General Schema)

The main idea behind this design is the concept of removing the physical aspects of any database into 'meta-data'. For example, table names are stored separately, as are object ids/classes, and the links between object instances. By doing this it makes it possible to maintain links and table information outside of the projects data tables, allowing this data to be accessed in a uniform way, and changed with minimal changes to the actual design. It is also possible to 'stripe' or split tables of data across databases, and even servers, with no changes to the access code. This allows for faster deployment, standard data access routines, and the ability to grow the system without the need to redesign or re-implement the actual database.

Due to the nature of the design, standard data access routines can be built in any language to access any table. It is also possible to write these as Stored Procedures.

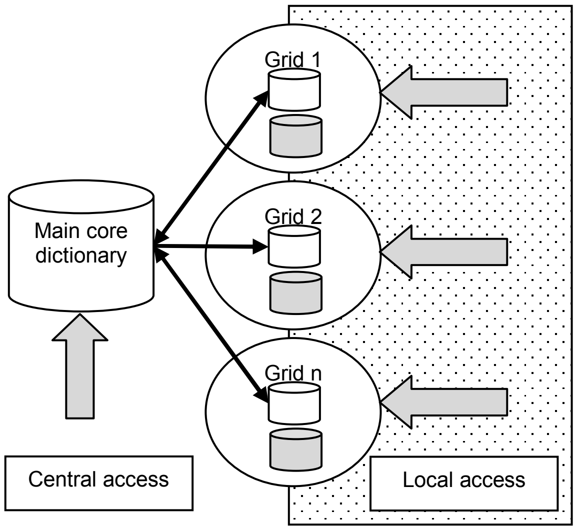
In this way data can be retrieved, and stored/updated, using exactly the same routines, removing the need to write new routines for each table that is created. This enables the DB system to use the same execution paths, resulting in a much faster and more efficient use of the database.

Due to the 'object' nature of the data items, the links are maintained outside of the actual data tables.

This allows for an item to be linked to multiple different entities; the links can be turned 'on' or 'off', without the need for lookup or cross-referencing tables, saving space and the potential for deadlock.

3.1 Basic Concepts

Each data item (tuple) in the project data will be assigned a unique ID. This is database independent, but can either be a numeric, or a generated unique ID. The model allows Ids to be uniquely generated within each class if the size of the system requires this, which is the case in the GOCDB.



(Fig. 2 Application of the model in a Grid Deployment context)

3.2 The Core Tables

The Core tables that make up the system in a grid deployment context (see Fig.2) are:

3.2.1 tObjectClass. A list of object classes, owner grids, their table names, servers, and valid dates, to act as 'switches'. By adding a Grid_ID item to the system, we are able to distribute the model across all servers that are a part of the Grid, and query either the whole Grid, or a part of it.

Similarly, local server systems will be able to add their own distinct tables to the system, masking them from external access. The GridId and ObjectIDs will be grouped, so that the core data and local data are all kept separate. Each Grid/Cluster will have its own ID. Object Ids will conform to the following:

- 0-99 will be for Core Objects.
- 100 –9999 will be for core data tables
- 10000 – n will be for local use.

While specifying actual ranges is against the 'object' principle, it is a small price to pay for the flexibility the system allows.

3.2.2 tLinkTypes. A list of valid, and invalid, links between object classes. This is in effect the integrity checking rules. Similarly to the ObjectClass table, we will have predefined ranges for the Links:

- 1 –99 for Core Link types
- 100 – 9999 for core data
- 10000 – n for Local applications.

3.2.3 tObject. A list of actual object instances, their class type, and dates on and off.

3.2.4 tObjectLinks. A list of actual Object links, with the date on /off switches.

3.2.5 tIDs. An optional list of the last Object ID used. This can provide a faster way to obtain an object ID, if the ID is numeric, as opposed to a random number generator.

3.2.6 tGridDetails. A list of participating grids, and their last used Id for objects.

1. Produces the next unique object id per grid.
2. Checks and flags if a grid is available or not.
3. Potentially should hold a list of server details per grid as well, so that a 'round robin' process could check on the availability of a given grid/server.

All project data will have a 'objectid/gridId' column pair, which will also act as the primary index. This will result in fast data access across the systems.

3.3 The API Routines

The following routines are used to manipulate the data in the main Core tables, and the object data tables that they reference. These are:

NewObject()
Creates a new object id of a given class.

GetObjClass()
Gets the Object class and Grid ID of a given Object.

GetTypeFromName()
Gets Object Class from table name

GetTable()
Gets full table name from Object Class.

NewLink()

Adds a New Link of Objects

SelectObject()
Selects the Data for a given Object

GetObjects()
Returns an array of values for a given Object Type

InsertObject()
Insert Object of Given Type

UpdateObject()
Update Given Object

DeleteObject()
Deletes a given Object. In reality should set the 'Dateoff' to before the DateOn so deletion can take place at a later date.

GetRelations()
Gets a list of Parents or Children for the given Object.

GetLinks()
Gets a list of links to the given object

FindObjects()
Finds Objects of a given type that match a condition

IsLink()
Checks on a Link being present

As these generate standard SQL, the full use of the database optimization can be used. More importantly, as all the relevant data is held in either the Core tables, or the data dictionary of the DBMS, these routines will work over current or future tables, so no further SQL code will be required by the clients.

4. Applications

4.1. Current Uses

This methodology was first implemented at STFC in 2007 for the Grid storage accounting system [8], using MySQL and PHP. By using this methodology we were able to display full storage information of the whole EGEE grid, due to the speed of access, as well as allow for multiple grids to be set up, and maintained by the user. A single system may also appear in multiple grids, using the same data, and the same data access systems.

The methodology was first invented in 1997, and installed at Britannic Assurance [9] in Birmingham, saving 7 hours processing per day over the previous Relational model. It was later used at Allied Domecq Spirits and Wine [10] for an MIS system spread over 54 countries, running under Oracle 7/8. More recently it has been installed at a number of Web Design agencies, and used as the basic backbone of a custom CMS system, allowing for very fast deployment of new web sites for clients.

4.2 GOCDB: the EGEE-WLCG topology authoritative database

4.2.1 Context. GOCDB stores data for around 450 sites, 2400 grid nodes, 4000 service endpoints and 1300 users. This represents around 52000 tuples and 10 MB of data spread between 40 tables within current relational model, which makes it a rather complex structure compared to the relatively low number of records it stores. It is currently hosted on an Oracle11g cluster.

4.2.2 Challenges for the future. The global operational model in the 3rd phase of the EGEE project is about to change dramatically from how it was in EGEE-I and EGEE-II [11]. The global evolution of the whole EGEE infrastructure has deep impacts on the tools that operate it, both in terms of organization and structure. Ideas for a general evolution in the years to come are discussed in [12].

The main evolution requirement in the context of this paper lies in distributing the information across the 11 current EGEE federations to match the requirements of the EGI design study [13]. The evolution of the operational model towards distribution implies that the operation tools, including GOCDB, follow this distribution and are maintained and operated under the responsibility of each region while remaining interoperable at a central level.

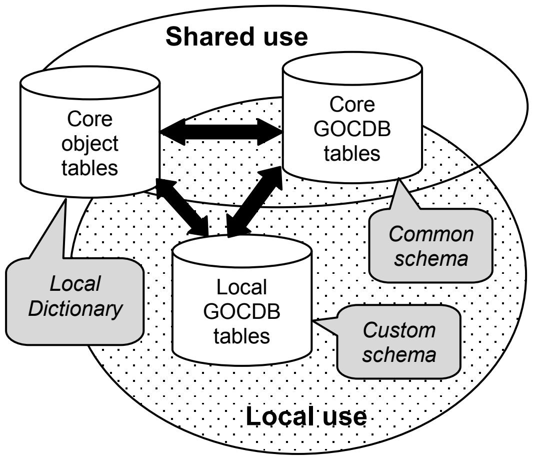
As a consequence, GOCDB has to evolve from a central database with a central interface to a distributed model. The resulting architecture should allow all these regional instances to be both fully interoperable and customizable to fulfil local requirements.

4.2.3 Applying the model to GOCDB. GOCDB current version, known as GOCDB3, runs on top of a classic relational model as described in [14] and is hosted on an Oracle11g cluster. Distributing this model by splitting it into 11 instances is not a solution that would allow easy interoperations between regions.

The DB design will be required to be a central resource, with the ability to be installed in multiple locations, to cope with multiple grids. It is also understood that each Grid may need or want to vary the design to fit specific needs for that area.

Our object design allows for this, but will also need to deal with data extraction from those grids/areas that provide a different data structure. The approach taken follows these points:

- Distribute the model into as many instances, or "Grids", as needed (see Fig.2)
- Have each of these instances split into 3 functional areas: core object tables (meta-data), core GOCDB tables (project level data) and add-on tables (regional Grid level data) (See Fig. 3)
- Provide functions, procedures and APIs to access these data
- Have a main core dictionary that will mirror all the distributed core object DBs (meta-data only) as shown on Fig.2
- Provide a controller that will read and manipulate data from local Grids that don't wish to conform to this model



(Fig. 3 Model applied to GOCDB)

4.2.4 Benefits. The deadline to tackle the distribution issue and come up with a fully distributed model is very tight and coincides with the end of EGEE phase III in April 2010. Achieving such a huge task in such a short time will be rather difficult, especially in an environment where so many people and tools are involved.

One of the benefits of using the described model is its easy deployment: installing and configuring a GOCDB regional instance should be as straight forward as possible.

Another benefit comes when thinking of possible future evolutions. As stated above, distribution will start with the current 11 EGEE federations but is likely to expand to many more instances if we think in terms of individual countries. Building the new architecture on top of a scalable model is then crucial.

As highlighted earlier in this document, this object DB design is adaptable to any system, in any context. Having GOCDB using it, with actual proof of efficiency and usability, may result in other major EGEE-WLCG operation tools adopting a similar design. The immediate benefit is to address the tool convergence issue described in [12], allowing for better modelling of the whole operations information system of the European Grid Infrastructure.

4.2.5 Working plan. GOCDB distribution has to be achieved before April 2010. The designed roadmap includes working on specific use cases before producing a generic package to be deployed by any region.

Our very first use case is to work in tight cooperation with a very well established national grid infrastructure, the NGS [15][16] in the UK. Definition of NGS specific needs already allowed validating the customization potential of the model as well as initiating valuable discussions on possible improvements.

Discussions already started with another well established National Grid, Grid-Ireland [17][18], as well as with different representatives of the national grid infrastructure in Spain [19] and Poland [20]. In parallel, a common work and study initiated with the developers of the Hierarchical Grid Site Management (HGSM) database developed as part of the SEE-Grid project [21] will validate the fact that our model can cope with external systems with traditional model.

Working closely with as many partners as possible will allow for a better integration of the model at such a large scale.

5. Future work

5.1 On the model itself

The model will be useable across applications, within the same schema. As local applications are proven, and required by other Grids, we see the migration of the data to be simplified by the use of this methodology. The core tables will already exist, requiring only the transfer of the local tables, and the setting up of the Class Ids within each Grids system. It will also be possible to make the data 'Grid-wide' if required, enabling a faster deployment of Grid specific applications.

By supplying the same APIs in multiple 'common-use' client side development languages, it will be possible to use this schema with the widest range of development environments. It is envisaged that educational and research establishments will benefit from this, and the portability of the data once complete.

In some instances, it may be beneficial to migrate some of the Core data, for example object classes, and link details. To facilitate this we will build a 'data exchange' using XML to pass the data from one system to the next.

We will also look at a heterogeneous link between databases, allowing the schema to be spread across differing database flavours.

5.2 Possible other applications

One of the areas of improvements for Grid operation tools within EGEE and WLCG is to work toward convergence in order to reduce the number of used tools and ease distribution. In this context, some discussions are ongoing about a possible integration between GOCDB and the EGEE Operations Portal, known as CIC Portal [22]. One of the proposals is to share GOCDB back-end model to facilitate service integration.

Discussions are also initiated about using this model to build the main topology information repository of the E-science Grid Facility for Europe and Latin America (EELA) [23].

Applying our model in this context could result in an even more challenging work by expanding again the already large scale of the current scope.

6. References

[1] *GOCDB homepage*, URL: http://www.grid-support.ac.uk/content/view/406/290/

[2] *EGEE web portal*, URL: http://www.eu-egee.org/

[3] *LCG web portal*, URL: http://lcg.web.cern.ch/LCG/

[4] *GOCDB web portal*, URL: https://goc.gridops.org

[5] E.F. Codd, "Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks", *IBM Research Report*, 1969

[6] *Object DB resource Portal,* http://www.odbms.org/

[7] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber, "Bigtable: A Distributed Storage System for Structured Data", *OSDI'06: Seventh Symposium on Operating System Design and Implementation*, Seattle, WA, Nov. 2006.

[8] *STFC Grid Storage Accounting System*, URL: http://stoac.grid-support.ac.uk/

[9] *Britannic Assurance P.L.C*, URL: www.britannic.co.uk

[10] *Allied Domecq*, URL: http://www.allieddomecq.com/

[11] H. Cordier, G. Mathieu, F. Schaer, J. Novak, P. Nyczyk, M. Schulz, M.H. Tsai, "Grid Operations: The evolution of the operational model of the first year", *Proceedings of the CHEP06 (Computing in High Energy and Nuclear Physics) conference*, Mumbai, India, Feb.2006.

[12] J. Casey et al, "Operations automation strategy MSA1.1", 2008, https://edms.cern.ch/document/927171.

[13] *European Grid Infrastructure Design Study*, URL: http://web.eu-egi.eu/

[14] *GOCDB3 development, wiki and documentation*, URL: http://goc.grid.sinica.edu.tw/gocwiki/GOCDB3_development

[15] *UK National Grid Service*, URL: http://www.ngs.ac.uk/

[16] L. Wang, W. Jie, J. Chen, "Chapter 9: The UK National Grid Service", *Grid Computing: Infrastructure, Service, and Applications*, CRC, 2009

[17] *Grid-Ireland portal*, URL: http://www.grid.ie/

[18] B.A. Coghlan, J. Walsh, and D. O'Callaghan, "Grid-Ireland Deployment Architecture". In Peter M.A. Sloot, Alfons G. Hoekstra, Thierry Priol, Alexander Reinefeld, and Marian Bubak, editors, *Advances in Grid Computing* - EGC 2005, LNCS3470, Amsterdam, The Netherlands, 2005

[19] *Grid Infrastructure for Advanced Research at the Spanish National Research Council*, http://www.grid.csic.es/

[20] R. Gokieli, K. Nawrocki, A. Padee, D. Stojda, K. Wawrzyniak, W. Wislicki, "Polish grid infrastructure for science and research", *Proceeedings of IEEE Eurocon 2007*, Warsaw, Poland, Sep. 2007, p.446

[21] *South Eastern European Grid-Enabled e-infrastructure development*, URL: http://www.see-grid.org/

[22] O. Aidel, A. Cavalli, H. Cordier, C. L'Orphelin, G. Mathieu, A. Pagano, S. Reynaud, "CIC portal: a Collaborative and Scalable Integration Platform for High Availability Grid Operations", *Proceedings of The 8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, Austin TX, US, Sept. 2007

[23] *EELA portal*, URL: http://www.eu-eela.eu/