Table of Contents

Investigations on a Relational Schema based Information System

This page is intented to describe the investigations which have been done for switching the current LDAP based data model of the EGEE Information System to a relational data model.

The target is to compare the both systems in terms of scalability and request handling speed.

The results will influence the future EGEE Grid Information System developments.

Transformation of the existing information

The information model for the BDII is currently (Sept. 2007) only defined for the LDAP protocol. Hence, it needs to be transformed from its hierachical structure to a 'flat' relational one. For this, each entry had to be parsed, its content mapped to tables and subsequently stored in a relational database. This mapping of the data is the most difficult part since it determines how the data is stored later on and therefore also determines the readout speed when the data is requested.

To start with this, the R-GMA configuration file (**gin.conf**) has been modified and adapted to the GLUE 1.3 schema (however, it is NOT compatible anymore with R-GMA). For the time being it is not necessary to create for each objectclass in the LDAP protocol a table. A LDIF entry is mapped to a table and its attributes mapped to its columns.

The multi valued attributes in some entries need to be stored in a seperate tables to normalise the database structure as much as possible. A **ValueTable** and **TypeTable** have been therefore introduced. The type (name) of the attribute is stored in the TypeTable, the value in the ValueTable. The ValueTable keeps a foreign key relation to the TypeTable to hold the information what type this value is. Again, it references by an ID to a parent in the other tables (GlueCE, GlueSE, etc.) From the query investigations from SlapdIndexedDB it didn't seem to be fisible to store the parent type table with the value.

Difficulties:

- hierachial structure to 'flat' relational structure
- UniqueIDs are sometimes not given
- Key referrals are not always consistent with related entries (A entry refers with a ForeignKey to another entry, but with wrong key)
-

Below you can find a **simplified** schema, which shows the ValueTable, TypeTable and parent entry relationship:

(please see a picture of the full schema attached)

The Tool

To do the transformation a perl script has been developed. Its source and documentation can be found in the glite cvs directory.

It reads a LDIF file in LDAP V3 format, tries to map the entries and attributes to tables and columns, fills the values into a hash arrays and finally inserts the data into the database. With this a **structure validation** tool is available too (however, it does **NOT** check the content, this is left to the infromation providers). For each problem a message is printed with the cause. If written to a file these messages can be summarised using

another script. An output would look like this :

```
Stats:
====================
Parsed LDIF entries  :  32376   (13.304646 seconds)
Inserted DB records  :  109892  (793.783362 seconds)
Refused LDIF entries :  2
Ignored values       :  4024

Error Classification:
====================
      1 ORA-12899: value too large for column "LCG_FELIXEHM"."GLUELOCATION"."UNIQUEID" (actual: 2
```

http://glite.cvs.cern.ch/cgi-bin/glite.cgi/glite-info-tools/ValidationAndDBInsert/

Some handmade checks

Find all SEs where the is not unique:

```
ldapsearch -x -LLL -h lcg-bdii:2170 -b o=grid objectClass=GlueSE GlueSEUniqueID | grep '^Glue*' |
```

Show, which attributes are published per objectClass in the Information System

```
# Example for a GlueSA entry:
ldapsearch -x -h lxn1185:2170 -b o=grid objectClass=GlueSA | grep '^Glue.*' |sed 's,\(.*\): .*,\1
```

Insertion

Below is an output of the tool, when processing a full dump from lcg-bdii and inserting into a local MySQL database.

```
===>  GlueCESEBindGroup : 6050 values in 3025 rows took 0.987007sec
===>  GlueSEControlProtocol : 2052 values in 342 rows took 0.158509sec
===>  GlueVOInfo : 60 values in 12 rows took 0.005876sec
===>  TypeTable : 42 values in 21 rows took 0.006758sec
===>  GlueSubCluster : 7881 values in 375 rows took 0.32402sec
===>  GlueLocation : 48606 values in 8102 rows took 4.032659sec
===>  GlueSA : 52093 values in 3480 rows took 2.353877sec
===>  GlueSEAccessProtocol : 6758 values in 978 rows took 0.485846sec
===>  GlueCluster : 1477 values in 371 rows took 0.151308sec
===>  GlueCESEBind : 20908 values in 5243 rows took 2.191324sec
===>  GlueSL : 20 values in 5 rows took 0.002251sec
===>  GlueCEVOView : 53208 values in 4771 rows took 3.031827sec
===>  GlueCE : 81774 values in 3012 rows took 2.980748sec
===>  GlueSE : 2680 values in 340 rows took 0.169503sec
===>  GlueSite : 2892 values in 263 rows took 0.155399sec
===>  GlueService : 16377 values in 1585 rows took 1.067189sec
===>  ValueTable : 246279 values in 82093 rows took 34.992386sec
Parser : Parsed 33968 entries in 46.335935 sec.
SQL    : Inserted 114018 rows in 53.096487 sec.
```

The Tests

The next step is to check the speed of the relational DB with the obove mentioned schema and to compare it with the OpenLDAP tests which have been done before. In respect of possible future use it was also interesting which implementation of the client would be the fastest. PERL, Python and a C++ (based on the Persistency framework http://pool.cern.ch/coral/) were compared. All three implementation models give us the possiblity of **changing** the underlying DB architecture without major code changes. MySQL and Oracle as the most common relational databases were used as candidates for this 'Client Implementation Speed Test'.

In the second part (Full Test) we use the cognitions from the Client test run the testsuite () against the database instances.

Client Implementation Speed Test

All three client issued the same SQL query to the database:

```
SELECT * FROM GlueSA, ValueTable, TypeTable
WHERE VALUETABLE.PARENTID=GLUESA.ID
  AND VALUETABLE.TYPEID = TYPETABLE.TYPEID
  AND TYPETABLE.NAME ='GlueSAAccessControlBaseRule'
  AND VALUETABLE.VALUE='lhcb'
```

The corresponding LDAP query:

```
ldapsearch -x -LLL -h lcg-bdii:2170 -b o=grid '(&(objectClass=GlueSA)(GlueSAAccessControlBaseRule
```

("Find the SA which provides a certain feature")

Oracle Dev DB

Requests going through Persistency take much more time than Perl or Python. This is due to the fact that Persistency loads the required libraries first. The graph shows how much time is spent if the SEAL, BOOST, Xerces and other libs are loaded from AFS in comparison if those are loaded locally from the machine. All client implementations use the **Oracle libs available from AFS**. Still, Perl and Python are around 50% faster than CORAL. The LDAP query result time in the graph cannot really be compared to the previous mentioned implementations since the Oracle DB instance is only a developer one and the LDAP query had been issued against the official load balanced BDII with high performance machines.

MySQL 5 DB

The graph obove shows the client execution time results using a MySQL (version 5.0.45) database and a native LDAP client. The OpenLDAP (version 2.2.13) and MySQL server run on the same hardware (SLC 4.5, Xeon 2.4Ghz, 1GB Memory, http://lemonweb.cern.ch/lemon-status/info.php?host=lxn1185).

The CORAL, Python and PERL client execution times are very close and make nearly not difference within each other. Still, the OpenLDAP native client is the fastest. If the same LDAP query is made through a PERL script the result is around 25 times and the Python version about 4 times slower than the native client written in C.

Full Test

This section shows the result from a full test run. In detail this means:

- 9 Clients issuing requests through **PERL DBI**
- 10 parallel requests / machine **=> 90 in total**
- relational and LDAP DB contain the same information
- burst length 60 seconds
- **Oracle Client libs loaded from AFS**
- MySQL libraries loaded locally

Please note : Previous tests used different LDAP queries. Therefore the graphs for the OpenLDAP test shown below look different than the ones show in SlapdDualCoreTest . Also the number of parallel requests have been changed from max 60 (before) to 90.

SQL Queries:

```
1. select * from GlueCluster, ValueTable where GlueCluster.uniqueid = 'obsauvergridce01.univ-bpcl
2. select * from GlueSE,ValueTable where uniqueid = 'lcgrid.dnp.fmph.uniba.sk' and ValueTable.par
3. select * from GlueService,ValueTable where uri = 'httpg://dpm.cyf-kr.edu.pl:8443/srm/managerv1
4. select * from GlueSA, ValueTable, TypeTable where ValueTable.parentid=GlueSA.ID and ValueTable
5. select * from GlueCE, ValueTable, TypeTable where GlueCE.id = ValueTable.parentid and ValueTab
6. select * from GlueCESEBind
7. select * from GlueSubCluster
8. select * from GlueService, ValueTable, TypeTable where GlueService.id = ValueTable.parentid an
9.select * from GlueCEVOView, ValueTable, TypeTable where GlueCEVOView.id = ValueTable.parentid a
```

LDAP Queries

```
1. /usr/bin/ldapsearch -x -l 15 -h lxn1185 -b o=grid (&(objectClass=GlueCluster)(GlueClusterUniqu
2. /usr/bin/ldapsearch -x -l 15 -h lxn1185 -b o=grid (&(objectClass=GlueSE)(GlueSEUniqueID=lcgrid
3. /usr/bin/ldapsearch -x -l 15 -h lxn1185 -b o=grid (&(&(objectClass=GlueService)(GlueServiceURI
4. /usr/bin/ldapsearch -x -l 15 -h lxn1185 -b o=grid (&(objectClass=GlueSA)(GlueSAAccessControlBa
5. /usr/bin/ldapsearch -x -l 15 -h lxn1185 -b o=grid (&(objectClass=GlueCE)(GlueCEAccessControlBa
6. /usr/bin/ldapsearch -x -l 15 -h lxn1185 -b o=grid (&(objectClass=GlueCESEBind))
7. /usr/bin/ldapsearch -x -l 15 -h lxn1185 -b o=grid (&(objectClass=GlueSubCluster))
8. /usr/bin/ldapsearch -x -l 15 -h lxn1185 -b o=grid (&(GlueServiceType=*)(GlueServiceAccessContr
9./usr/bin/ldapsearch -x -l 15 -h lxn1185 -b o=grid (&(&(objectClass=GlueVOView)(GlueChunkKey=Glu
```

MySQL 4.1.2

MySQL with memory stored tables

MySQL offers the possiblity to store tables in memory which has the advantage of being very fast. However, since this is limited to the system's available memory huge tables cannot be stored. Here, all tables EXCEPT the ValueTable (100K rows) were stored in memory. The disadvantage is that whenever the server is stopped the data (not the schema) is lost.

Oracle 10.2

OpenLDAP 2.2

Average Response Time Test

Below you'll find graphs which show the average response time behaviour of clients requesting data from MyLSQ, Oracle and OpenLDAP.

Technical details :

- MySQL and OpenLDAP run on a Xeon 2.4 GHz, 1GB memory
- The Oracle RAC (Real Application Cluster) instance consits of two nodes with a replicated database.

The resultset of the relational schema and the OpenLDAP schema differs in terms of returned data size.

As seen from the legends in the graphs OpenLDAP returns for each entry all descriptional information (attribute names), too. This increases the total size of the returned data by a factor of ~4 compared to the resultset of the relational database. The only way to avoid this, is by changing the requested data. But the target is to know how fast those systems return information a user requests (which is the idea of this test !) this fact should not be avoided.

Example (Test Query 1):

```
/usr/bin/ldapsearch -x -LLL -l 15 -h lxn1185:2170 -b o=grid '(&(objectClass=GlueSE)(GlueSEUniqueI

# extended LDIF
#
# LDAPv3
# base

SELECT * FROM GLUESE WHERE UNIQUEID = 'lcgrid.dnp.fmph.uniba.sk'
mysql> SELECT * FROM GLUESE WHERE UNIQUEID = 'lcgrid.dnp.fmph.uniba.sk';
+------+--------------------------+------------------+--------------+-----------+----------+-----
| ID   | UniqueID                 | Name             | Architecture | SizeTotal | SizeFree | Info
+------+--------------------------+------------------+--------------+-----------+----------+-----
| 7808 | lcgrid.dnp.fmph.uniba.sk | FMPhI-UNIBA:disk | disk         |         0 |        0 | ldap
+------+--------------------------+------------------+--------------+-----------+----------+-----
1 row in set (0.00 sec)
```

Conclusion

-- FelixNikolausEhm - 20 Sep 2007

This topic: EGEE > BDIIRelationalDBBackend
Topic revision: r2 - 2010-06-11 - PeterJones