gLite FiReMan

Krzysztof Nienartowicz, CERN
On behalf of EGEE JRA1

VLDB 2006
Phases

• What and why Oracle FiReMan was built for
• Interface driven functional requirements
• Hardware and use case driven speed requirements
  – Feasibility study
• Resulting approach and internal functionality
• Selected database techniques and features used
• Problems encountered
• Distribution
• Results
• Lessons for the future
This is reduced by online computers that filter out a few hundred “good” events per sec.

Which are recorded on disk and magnetic tape at 100-1,000 MegaBytes/sec \(\rightarrow\) \(~15\) PetaBytes per year for all four experiments.
Data Handling and Computation for Physics Analysis

- Enabling Grids for E-sciencE

- Detector

- Event filter (selection & reconstruction)

- Reconstruction

- Raw data

- Event reprocessing

- Event simulation

- Event summary data

- Batch physics analysis

- Analysis objects (extracted by physics topic)

- Processed data

- Interactive physics analysis

- Raw data

- Event reprocessing

- Event filter (selection & reconstruction)

- Simulation
FiReMan = File and Replica Management Catalog

EGEE aim to provide holistic catalog solution for Grid

- Initially based on cross of HEP and biomedical requirements
  - Logical (LFN, GUID) <-> (SURL) physical mapping space as in EDG
  - Hierarchy as in Alien, AFS, GNS or other Grid cataloging solutions

- Built on top of experiments’ and industry experience
  - Hierarchical space as
    - LDAP
    - MS Active Directory
  - Ease of Distribution

- Big and Fast fast fast
Catalog Content

- File Catalog and StorageIndex
  - Filesystem-like view on logical file names
  - Keeps track of sites where data is stored
  - Conflict resolution

- Replica Catalog
  - Keeps information at a site

- Meta Data Catalog
  - Attributes of files on the logical level
  - Boundary between generic middleware and application layer

**Metadata**

- **LFN**
- **GUID**

**Central Catalog**

- **SE ID/ SURL**
- **SURL**

**StorageIndex**
Redefinition of VLDB.

EGEE VLDB...

That is fast and scalable.

On our hardware.

Say $10^{11}$ entries and thousands of users. Simultaneously accessing database.

13. There is no magic to make MySQL originating things working faster on commercial databases, including Oracle. Formula is to use advanced, often proprietary features.

15. Generic = Slower but Proprietary $\neq$ Fast

17. Gains from hardware upgrades are very costly, and surely not linear. See TPC-C winning machines.

19. Volume of data increased order of magnitude but not that much the number of users/transactions served by the commodity hardware.
• Multiple shared types among different interfaces.
• FiReMan links LFN, Replica and Metadata worlds
• Lists of objects are both parameters and are being returned
Functionality

- 50 methods
- 16 types
- Dynamic hierarchical space
  - Create entries
  - Remove
  - Update
  - Rename
  - Read
  for LFN or replica/security/metadata dimensions
  - Nested symlink support with checking for cycles
- Administrative or on the fly creation of users – VOMS dependency
- Superusers
- Ownership and permission context part of the core system

- Bulk operations.
  - Transactionally safe
  - Stateless
    - Windowing operations for big results
Interface objects

- No constraints on ACL length
- No constraints other than system capacity on input/output parameters
  - i.e. LFN max ~32KB
    - No levels constraint
  - No constraints on metadata attributes number
- Surely not relational/dataset model

Old impedance mismatch problem
- Especially if we want to have bulk operations
- We do not want to do JOIN by ourselves – it is work for database
  - Flattening is bad either way
- Solution: Propagate complex objects down to the database and process them there naturally
Why interface (API) we define is important for persistent data modeling?

- List of complex objects
  - With complex objects embedded
  - With lists of complex objects embedded
    - With list of objects

Because it may complicate our lifes....
File access when the files are owned by a single entity in the Storage Element. The access control is enforced by the gLite I/O server.

1. open(LFN)
   
   1.1: checkPermission(LFN, read); void
       
       {service/user cred}
       
       1.2: listReplicas(LFN, false); ListEntry[]
           
           {service/user cred}
           
           1.3: TURL = prepareToGet(SURL)
               
               {service cred}
               
               1.4: get(TURL)
                 
                 {service cred}
                 
                 1.5: setStatus(SURL, Done)
                     
                     {service cred}

the gLite I/O server passes the user's credential to Fireman, but authenticates with its own service cert.

the files in the SE are owned by the gLite I/O service entity.
Data replication and retrieval usecase

1. Request replication
2. Get SURL
3. Get file
4. Get file key
5. (encrypted) File replication
6. Request file
7. Get SURL
8. Get file
9. Return (encrypted) file

User Interface

DICOM server

SRM-DICOM interface

Fireman

File ACL control

gLiteIO server

gLiteIO client

Worker Node

SRM interface

Any SE

Any file

DICOM server

Hydra Key store

Key store

gLiteIO client
Database machine
CERN diskserver

- Dual Intel(R) Xeon(TM) CPU 2.4 GHz, 512kB cache
- 2GB
- Roughly 1000GB available for database
  - RAID0 – 9 mirrored disks
  - Almost standard CERN Oracle configuration
    - 16k db_block_size preferred, with buffers db_16k_cache_size set
    - Some tuning on db_file_multiblock_read_count
  - Still space for low level DB process tuning

- Orders of $10^8$ – hierarchical entries in the catalog possible volume-wise
  - Plus related security info and replica info, min. additional single entry doubles the number $5*10^8$ to over $10^9$. 
Feasibility study

- Massive System Servicing performance evaluation
- M/M/1 queueing model for simplicity just for database backend
  - Look for limiting service figures based on experience and requirements
    - CMS document
    - Compass experience
    - Volumes of data expected
    - Guessing, intuition...
  - Only database considered as application servers easier to multiply
    - Real bottleneck in a longer term is a database not a protocol or security overhead
Average users call rate during the heaviest load: $\lambda := \frac{200}{\text{sec}}$

Average time spent on servicing DB call, assumption: bulk calls, 10-250 items per call: $x := 170\text{ms}$
Average time spent on servicing DB call, assumption: bulk calls, 10-250 items per call:

number of database processing slots \( m := 50 \)

average users call intensity during the busiest time \( \lambda := \frac{10}{\text{sec}}, \frac{100}{\text{sec}}, \frac{500}{\text{sec}} \)

\[ C(m, \lambda, x) \]

- \( x := 170\text{ms} \) for ~50-500 bulk calls
- \( x2 := 50\cdot\text{ms} \) for ~1-50 bulk calls
- \( x3 := 1\text{sec} \) heavy load, 500-..bulk calls

Queuing/Timeout Probability (%)

- Average servicing speed 170ms
- Fast servicing for small calls 1-15 items, 50ms
- Slow processing speed while DB is overloaded or v. big in/out 1
5% logic but 12K of code
Errors mapping
Distribution
Largely autogenerated stubs.

95% of FiReMan logic
~ 20K PL/SQL Lines

Database
Fireman 'PL/SQL' process/package

Object Types Interface
Full request handling.
Operation on data,
inserting
querying
deleting
traversal
heuristics
Superuser refresh
Memory structures

Fireman Admin 'PL/SQL' process

Monitoring data skew
Partitioning agent

Helper packages

GUID, date/time, callout
libraries, static permissions logic

Helper libraries

95% of FiReMan logic
~ 20K PL/SQL Lines

Database
Fireman 'PL/SQL' process/package

Object Types Interface
Full request handling.
Operation on data,
inserting
querying
deleting
traversal
heuristics
Superuser refresh
Memory structures

Fireman Admin 'PL/SQL' process

Monitoring data skew
Partitioning agent

Helper packages

GUID, date/time, callout
libraries, static permissions logic

Helper libraries

95% of FiReMan logic
~ 20K PL/SQL Lines

Database
Fireman 'PL/SQL' process/package

Object Types Interface
Full request handling.
Operation on data,
inserting
querying
deleting
traversal
heuristics
Superuser refresh
Memory structures

Fireman Admin 'PL/SQL' process

Monitoring data skew
Partitioning agent

Helper packages

GUID, date/time, callout
libraries, static permissions logic

Helper libraries

95% of FiReMan logic
~ 20K PL/SQL Lines

Database
Fireman 'PL/SQL' process/package

Object Types Interface
Full request handling.
Operation on data,
inserting
querying
deleting
traversal
heuristics
Superuser refresh
Memory structures

Fireman Admin 'PL/SQL' process

Monitoring data skew
Partitioning agent

Helper packages

GUID, date/time, callout
libraries, static permissions logic

Helper libraries
Functionality highlights

- **Unique features**
  - Internally implemented partial errors reporting - no speed loss.
    - If 3 items out of 1000 failed during one call, no rollback is necessary.
    - Remaining 997 items can be committed or read at user will.
  - Fast LFN renaming with negligible cost regardless of position in the hierarchy
    - Flat behaviour – same results with $10^6 .. 10^8$ items
      - Very big number of catalog principals – thousands feasible
      - Unlimited size of ACL
      - Unlimited size of metadata attributes
      - Unlimited number of superusers
      - Consistency of users, items and permissions
        - Fast removal of all pertaining ACLs if user/group removed
      - Easy reporting on ACL layout
      - Ability to amend default behaviour by users
        - Ownership of newly created items, home directory, dangling GUIDs handling
  - Functionality separated from protocol
    - Web Service, pure-JDBC, C++ possible
Component functionality

- Three libraries
  - AXIS-WS SOAP
    - 50 methods
  - JDBC Side
    - Web Service complex types mapped onto JDBC representation
    - 50 methods
  - PL/SQL Side
    - Web Service complex types mapped onto SQL representation
    - 50 methods
  - Client libraries, often with autogenerated custom wrappers WSDL->(Fireman XSLT)->client stub
    - Java
    - C/C++
    - Perl, Python
    - JavaScript
Hierarchy optimisation

• How to achieve these speeds and functionality
  – Hierarchy slicing
  – Hierarchy traversal
  – Hierarchy clustering

• /VO1/experiment1/subexperimentA/run2
• /VO1/experiment1/subexperimentA/run2
• ...
• /VO1/experiment1/subexperimentA/run2/user1
• ...

– Database cache utilisation/invalidation during traversal
– Clustering during DB reads
Hierarchy mapping

- Clustering
  - LFN traversal is much faster
  - Wise caching utilisation
  - Index organized table
    - Plus compression
Query, PL/SQL patterns used

- **To get functionality with minimal CPU, Disk activity**
  - `CONNECT BY`
    ```sql
    select sys_connect_by_path(name,'/') path, f.*
    FROM
    FC_FILE f
    START WITH parent_id=0 AND file_id=1
    CONNECT BY PRIOR file_id = parent_id
    AND NAME=fireman.getpathtoken(LEVEL-1,path)
    ```

- **To diminish size of joins or memory usage**
  - Functions in query
  - In-memory collections
    - `TABLE()` operand
    - Known heuristics to limit selectivity
      - `chdir` behaviour within single operation
      - *Token read clustering* so no need to traverse same path in a single operation

- **Both CPU and memory**
  - Non-relational resultsets
  - Conditional paths

- **Use bulk PL/SQL**
  - Its error reporting
Querying madness – decomposition

Enabling Grids for E-sciencE

Collecting data from previous stages to be returned. Create in-column, nonsquare collections as correlated nested subqueries based on data from previous stages.

Reacting on errors returned by creating error objects to be put on the errors output list.

Traversal of the whole path
And permissions/existence check
Possible symlink traversal or callouts by stored function

Directory LFN traversal

Intermediate Resultset – prepare directory starting Ldcs by removing duplicate directory LFNs

INFSO-RI-508833
select * from TABLE(getDirectoriesLFN(inputCollection));

- **Good choice if we don’t care about CPU stress**
  - They introduce object or procedural feeling to data flow within queries and ease code maintainability

- **They are not for free**
  - They cost a SQL<-> PL/SQL context switch
    - 5-50% slower than subqueries
    - There are Oracle bugs...
Enabling Grids for E-sciencE

Partitioning

• Thanks to Index Organized Tables data is naturally clustered
  – Partitions are based on adjacency and traversal level level
  – Partitions can be created by administrative process running in the background
    ▪ 300M entries tested
      • ~800M database entries (LFN, Replica, ACL, Metadata)

  ▪ ~10K..10M file items in one partition
    • For the hierarchical space - list partitioning by value
      o parent<->child relationship being a key
      o Clusters of keys (parent_id,file_id)
      o Some data reorganization plausible

• For GUID driven spaces
  o Hash partitioning
• **Architecture**
  
  – **Publisher**
    - Publishing code embedded in the application server
      - *Needs common transaction boundary*
  
  – **Consumer**
    - Specialized process that subscribed to the Topic
      - *Single Topic currently*

  – Based on standard JMS
  – Freedom of topologies, vendors
    - Publishers tested on GNU Joram and Oracle AQ
Enabling Grids for E-sciencE

Middleware
App Server
Tomcat

JDBC<->PL/SQL Stored procedures mapping
Java<->JDBC arguments translation and transaction logic

Database
Fireman 'PL/SQL' process

- Full request handling
- Operation on data, errors preparation
  - Inserting
  - Quering
  - Deleting
  - Traversal
  - Heuristics
  - Superuser refresh
  - Memory structures

Fireman Admin 'PL/SQL' process

- Monitoring data skew
- Partitioning agent

Helper packages
- Helper libraries
  - GUID, date/time, callout libraries, static permissions logic

Consumers
Fireman

JMS Channel

Messaging Service
JMS
(AQ, Joram, Jboss)
## Infamous ORA-600

<table>
<thead>
<tr>
<th>Problem</th>
<th>Present in</th>
<th>Bug reference</th>
<th>Fixed in (as of Jan 2006)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal errors when debugging pl/sql from JDeveloper on 9iR2</td>
<td>9.2.0.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORA-600 [VOPRVL1] IN QUERY HAVING AN LEFT OUTER JOIN USING OBJECT TYPES</td>
<td>10.1.0.3</td>
<td>4260205</td>
<td>patch 4260205</td>
</tr>
<tr>
<td>Recycle bin feature causes the database to crash the machine</td>
<td>10.1.0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORA-7445 [KOKQAGO] BEING HIT IN QUERY USING CAST/MULTISET</td>
<td>9.2.0.6</td>
<td>4288071</td>
<td>9.2.0.7</td>
</tr>
<tr>
<td>ORA-600 [KKXMCPLS2] BEING HIT DURING SELECT</td>
<td>9.2.0.6</td>
<td>4312416</td>
<td>no fix available</td>
</tr>
<tr>
<td></td>
<td>10.1.0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORA-600 [KOKEEAFI4] BEING HIT IN OBJECT ORIENTED QUERY</td>
<td>10.1.0.3</td>
<td>4327752</td>
<td>10.1.0.4</td>
</tr>
<tr>
<td>PSRC: ORA-600 [QKAFFSINDEX3] AFTER APPLYING 10.1.0.4</td>
<td>10.1.0.4</td>
<td>4457034</td>
<td>patch 4457034</td>
</tr>
<tr>
<td>WRONG RESULT WITH ALIASES IN NESTED SQL WITH A FUNCTION CALL</td>
<td>10.1.0.x</td>
<td>4695283</td>
<td>no fix available</td>
</tr>
<tr>
<td></td>
<td>10.2.0.x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: Michał Kwiatek
Wrong results 2

WRONG RESULT WITH ALIASES IN NESTED SQL WITH A FUNCTION CALL

*** 10/24/05 01:32 am ***

PROBLEM:
--------
Wrong result with aliases in nested sql with a function call.
Gives expected result in 9.2.0.7 but not on 10.1.0.4 and 10.2.0.1.
Customer needs a fix on 10.2.0.1.

DIAGNOSTIC ANALYSIS:
----------------------
Results got with query below:

SQL> select a per, a p, a p2 from (  
    2 SELECT (dummyBug2051014(1,2)) as a  
    3 FROM  
    4 (  
    5 SELECT /*+cardinality (i 20) */ *  
    6 FROM  
    7 TABLE(genLfns())  
    8 ) i);

9.2.0.7 :
--------
PER P P2
------------------ ------------------
 3 3 3
 3 3 3
 3 3 3

Execution Plan
---------------------------------------------------------------------
0 SELECT STATEMENT Optimizer=CHOOSE (Cost=11 Card=8168 Bytes=16336)
1 0 COLLECTION ITERATOR (PICKLER FETCH) OF 'GENLFNS'


Wrong results 3

10.1.0.4:

---

PER   P   P2
---   ---  ---
3     3    3

Execution Plan

-----------------------------------------------
0 SELECT STATEMENT Optimizer=CHOOSE (Cost=25 Card=8168)
1 0 COLLECTION ITERATOR (PICKLER FETCH) OF 'GENLFNS'

Idem with CHOOSE mode
(...)

WORKAROUND:
-----------
None found

RELATED BUGS:
-----------
None found

REPRODUCIBILITY:
-----------
9.2.0.7: No
10.1.0.4: Yes
10.2.0.1: Yes

TEST CASE: (....) and oracle changelog follows (no fixes to report ...☹☹☹)
Performance results

Raw database speeds without Application Server wrappers via JDBC

*LFN dependent – volume independent – same results for 1M and 250M entries*

- **getFileCatalogEntry of bulk 10 items**
  - 400 full FRCEntries/sec per client
    - ACL retrieval, permissions check, symlink traversal
  - Aggregated by 10 clients, up to 3000 items/sec

- **getFileCatalogEntry of bulk 500 items**
  - 1000 full FRCEntries/sec per client
    - ACL retrieval, permissions check, symlink traversal
  - Aggregated by 15 clients, up to 7000 items/sec

- **Insertion stable at around 100-400 creations/sec**

- **Security overhead – 10KB per call, four extra messages**
Performance

FRCCreate() performance - 2.5 replicas, 3 ACL

- Single client
- 10 clients
- 15 clients

items/s vs. items per call
Performance

getFileCatalogEntry() performance - avg 2.5 replicas, 3 ACL
Lessons

• Even Grid protocols/SOA could be expensive the real cost is in the DB, as usual

• Try to limit impedance mismatch
  – By designing flat interface, easily transformable on the way to the database
  – or go for object model while talking to DB.
    ▪ It’s a standard defined by SQL99 and supported by number of other DBs (DB2, PostgreSQL,...)
      • Not all bindings are supported – always Java, usually C++
    ▪ It saves memory and CPU on both client and DB ends
    ▪ Not always standard JDBC mapping is the best. (see jPublisher)

• Cluster persistent data
  – Use data model that takes fully what Oracle gives when it comes to clustering
    ▪ Index organized tables - IOTs
      • If suitable
    ▪ Partition data and indices
      • Not always natural partitioning solves all problems (timestamp not a access pattern key)
Lessons

• **Use selectivity knowledge**
  – Cardinality and other hints
    ▪ NESTED_LOOPS vs HASH_JOIN

• **Scalability facts**
  – Development, testing volume
    ▪ Use real volumes, even this is not enough (data skew triggered errors)
  – Test coverage
    ▪ Volume-wise, user#, DB version, data distribution characteristics

• **Demistified Oracle**
  – Tool with thousands of unique features
  – And same bunch of problems
    ▪ some very painful to track down and to allieviate with elegant workaround (on time)
  – OO and PL/SQL toolset below expectations