A DISTRIBUTED FILE CATALOG BASED ON DATABASE REPLICATION

S. Biegluk, V. Pinto Morais, A. J. Peters, P. Tissot-Daguette
CERN, Geneva, Switzerland

Abstract

The LHC experiments at CERN will collect data at a rate of several petabytes per year and produce several hundred files per second. Data has to be processed and transferred to many tier centres for distributed data analysis in different physics data formats increasing the amount of files to handle.

All these files must be accounted for, reliably and securely tracked in a GRID environment, enabling users to analyse subsets of files in a transparent way.

This paper describes a distributed file catalogue giving consideration to the distributed nature of these requirements. In a GRID environment there is on one hand a need for a centralized view of all existing files for job scheduling. On the other hand each site should be able – for performance reasons - to have autonomy to access files without the need of centralized services. The proposed solution meets the need for a local and global operation mode of a file catalogue. Commands can be executed autonomously in a local catalogue branch or heterogeneously in all of them.

The architecture is based on pure replication technology providing a real time backup. The catalogue implements a file system like view of a logical name space, user-defined meta data with schema evolution, access control lists and common POSIX user/group file permissions.

Architecture, interface functionalities, performance tests and very promising results in comparison to other existing GRID catalogues are presented.

ARCHITECTURE

Overview

Every GRID site deploys a fully functional file catalogue (FC) branch (local catalogue). Each of these FCs keeps an independent namespace connecting unambiguously the logical filename (LFN) with access permissions (ACL), GUID and physical filenames(PFN) per file entry. Authorization information is kept inside the local FC and valid only for this specific site! LFN/GUID translation and authorization for file access is executed within a site – this operation mode is called in the following 'local mode'. Through replication of all local FCs to a central location a 'global' read-only namespace overlay of all local FC namespaces is created. A global read operation mode refers to parallel queries to the local catalogue replicas and allows to gather information about all existing replica locations of a file (storage index).

The prototype implementation is based on MySQL ([5]) database back-ends and pure replication technology (MySQL Replication), offering the possibility of having a real time backup of the distributed catalogue branches at a central location.

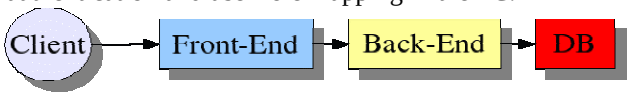
The local mode enables linear scaling of the architecture through independent site operation, the global mode satisfies the need for a storage index f.e. for scheduling of jobs according to data location,

We can distinguish three different kind of local functionalities:

- File Operations
- Access Control List Operations
- Meta Data Operations

The “global” mode provides identical commands, which are performed in all connected local catalogues for write operations or in the central replicas for read operations. To execute write operations in all local catalogue branches we replicate a command queue from the central location to each branch. In each branch a daemon executes the replicated global commands.

The catalogue database back-ends are not directly exposed in the client interface. The client talks to a catalogue front-end service, which is responsible for user authentication and user role mapping in the FC.



The front-end uses a session token mechanism. Clients authenticate once to the service using GSI authentication to obtain a session token. In concurrent calls the SOAP messages to the catalogue service are encrypted with a dynamic session token using symmetric CIPHER technology. The front-end service connects to the DB via the back-end with a single trusted user identity using standard DB authentication (password or SSL). This user owns all databases and tables. We have chosen PERL for the implementation of the catalogue back-end. The front-end service executes the back-end code.

Replication

The architecture uses two replication schemata:

- Replication N-Slaves→(N-)Master: this allows a central(global) view of all catalogue branches and a real-time backup
- Replication Master→N-Slaves: this is used for the replication of the global command queue.

The databases of each site containing the FC tables were configured as replication masters. Each MySQL master

records all insertions and modifications to the database tables binary logfiles, which are corresponding to the operation done in the file catalogue at each site. The slave databases (databases at central services) are synchronized by a MySQL slave replication daemon executing all statements from the binary log of the master database. As a result, we have a replication of all site catalogues in a central machine.

Figure 1 shows an example for a local mode operation. We consider a user running data analysis in a given site:

1. A user submits an analysis job to be executed in a computing site, where the data can be read.
   An output file containing his analysis results is registered in the local file catalogue branch of the executing site.
2. The new entry in the master database is replicated to the central database replica.

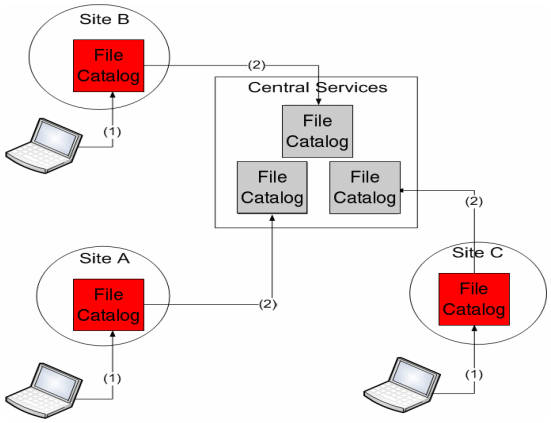


Figure 1 – Local mode operation

Figure 2 illustrates the three steps during a global write operation:

1. A command is inserted in the global command queue
2. The command is replicated to all file catalogue branches
3. The command is execution in all local file catalogue branches

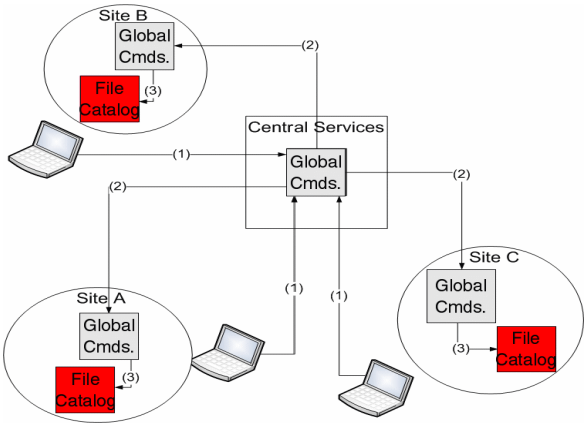


Figure 2 – Global mode write operation

A global read operation is implemented as a parallel query on the local FC branch replicas.

Local Catalogue Database Layout

In every local catalogue we use a flat name pace layout i.e. all logical file names are contained in one table. Additional information like file meta data or event meta data is stored in additional flat tables, which can be joined in queries. This very simple structure was tested to work well with 100M entries and standard DB machines. Permissions are stored for every entry identified by a unique LFN/GUID using the POSIX UID/GID schema or user defined access control lists (ACLs). The correspondence LFN↔GUID↔PERM↔TURL is stored in a primary table which contains the FC information.

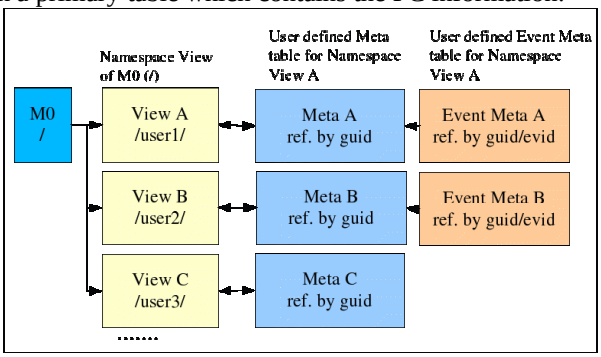


Figure 3 – Database Layout

Every user can create/update meta data tables to tag subset of registered files (implemented by creation of a view of the primary table). This schema was further extended to define for every file a list of event IDs, which can be tagged with user defined meta data on the event level. FC queries based on meta data or event meta data are single SQL statements which join corresponding tables and return all needed information together.

PERFORMANCE TESTS

We have evaluated the performance of the front- and back-end separately to understand better individual bottlenecks.

All tests were performed in a LAN environment since the majority of file and metadata operations are executed in LAN networks making one of the most important assets this approach.

Front-End Performance

We used the following test setup:

- Front-end Servers
  - 2 x Intel® Xeon™ 2.4 Ghz
- Client
  - Intel® Pentium® IV 3 GHz, 512 MB

We measured the secure RTT to send and receive a fully encrypted request from the client to the front-end server

over a GSI authenticated session connection. The request and response packets contained approx. 10-20 bytes.

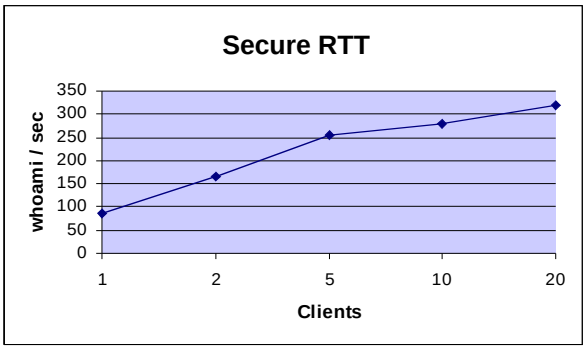


Figure4 – SecureRTT

The graph shows that we have an average RTT of 3.3 ms/call for 20 clients and around 11.7 ms/call for a single client. The measurements are stable for increasing packet sizes until we reach the limit of the network transfer speed. We can easily avoid bottlenecks in the front-end by scaling the number of front-end servers according to the performance of the underlying DB back-end.

Back-End Performance

The database FC performance was tested with the following setup:

- DB Server
  - MySQL 5.0.11 beta on 4 x Intel® Xeon™ 3.06 GHz, 4 GB.
- Client
  - Intel® Pentium® IV 3 GHz, 512 MB

The two functionalities as a pure file catalogue and as a meta data catalogue were tested separately. The number of concurrent clients was always varied between 1 and 20. During all tests the MySQL query cache was disabled.
We measured the following results for the FC:

- **Insertion File Rate**: The maximum number reached was 522 inserts/sec (Fig 5). The database server had a similar behaviour with different database size (10^4, 10^5 and 10^6). The same tests were made with bulk operation; here the maximum reached was 2200 inserts/sec (Fig 6).

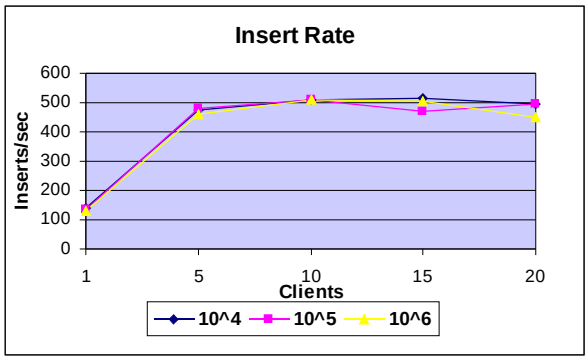


Figure5 – Insertionrateinserts/sec.

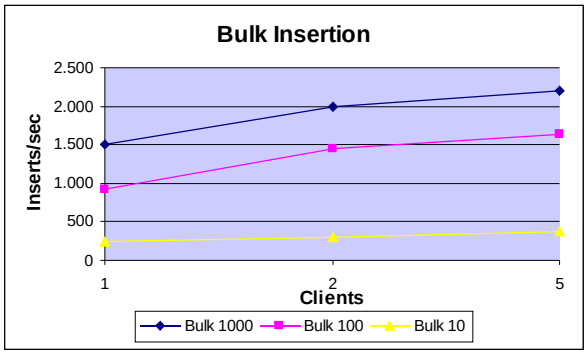


Figure6 – Bulkoperationsinserts/sec.

- **Replication Delay**: We measured the time to replicate an already populated FC database. The average time for a DB with 10^8 records was around 7000 records/sec and with 10^7 was 10000 records/sec and was mainly limited by the available network bandwidth.

- **Listing Rate:** Here we measured the listing time for a random directory with 1000 entries. The query was made 150 times in order to obtain a more accurate value. Independently of the database size the average was ≈ 1000 queries/sec.

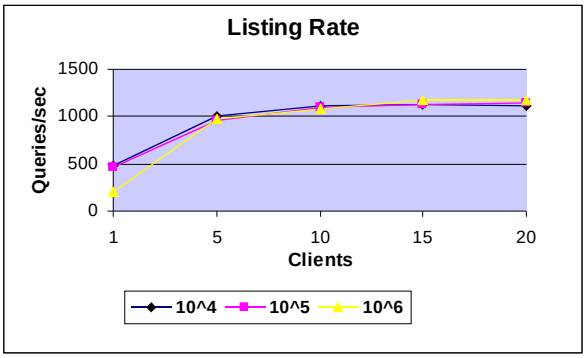


Figure7 – Listingrate queries/sec.

In the following we discuss the results for metadata operations:

- **Metadata Insertion Rate:** We measured the number of insertions per second varying the number of metadata tags per entry and the number of concurrent clients. The maximum tag insertion rate was 587/s with one tag per file and 16 clients. With an increasing number of tags the insertion rate decreases slightly. This behaviour is partially influenced by the network bandwidth limitation, which we hit for a huge number of tags and not a DB feature.

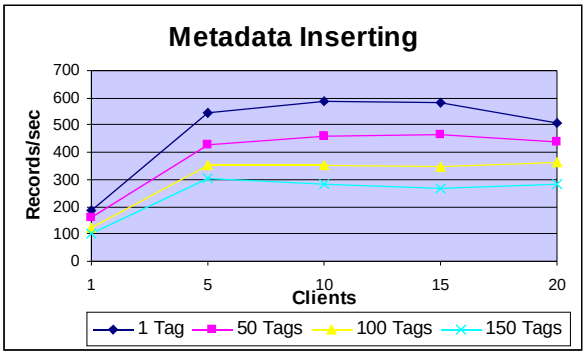


Figure 8 – Insertion Rate for a TEXT tag.

- **Metadata Query Rate:** For this test we were selecting a random float tag value range returning 5000 entries. The query was repeated 150 times in order to obtain a more accurate value. Independently of the database size the average result was ≈ 1 queries/sec returning 5000 entries.

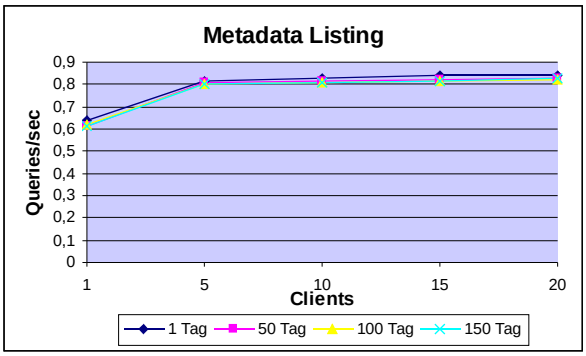


Figure 9 – Listing Rate FLOAT

COMPARISONS

There are several evaluations and performance measurements of the LFC and the FireMan file catalogues available (see [1] - [4]) . Most evident are differences in single entry operations for single clients. Our implementation outperforms the insertion and query rate of these two alternative implementations by at least one order of magnitude. The performance of a file catalogue can be compared to the theoretical insertion and query rate of the DB back-end used. For our test server this was in the order of max. 1000-2000 DB operations per second depending on the size of the query result or on the data to be inserted. Most FC functions could be be implemented with two database operations, one for the access control, one for the insertion or query statement. This is reflected in the results we obtained.

CONCLUSIONS

We have implemented a prototype of a distributed FC. The design meets the requirements of site independence and scalability of a distributed file catalogue as also the global storage index functionality. The performance of the local catalogue branches came close to what was theoretically possible with the used MySQL back-end. The performance of the global operation mode has not yet been studied in detail, although replication measurements show a quasi instantaneous response. The low-level MySQL replication is the fastest possible implementation for a file catalogue replica with MySQL back-ends. To be independent of the DB back-end one might also consider to replicate high-level catalogue statements (as already done for global commands) with some performance penalty. We have shown, that a very simple structure and database layout allowed to implement a distributed high performance catalogue for file- and (event-)metadata. Security and Performance considerations lead to the natural separation between FC front-end dealing with authentication/authorization and the back-end implementing all file catalogue functionality.

ACKNOWLEDGEMENTS

This research project was done in the ALICE ([6]) offline group at CERN ([7]). The authors are funded by CERN , byADI/FCT Portugal and by the EGEE project([8]). We would like to thank all of them for their support or funding.

REFERENCES

[1] Craig Munro and Birger Koblitz. Performance comparison of the LCG-2 and glite file catalogues. In ACAT 05, May 2005.

[2] Craig Munro, Birger Koblitz, Nuno Santos and Akram Khan. Performance Comparison of the LCG2 and gLite File catalogues.

[3] J. Baud, J. Casey, S. Lemaitre, C. Nicholson. Performance Analysis of a File catalogue for the LHC Computing Grid. In HPDC 14, 2005.

[4] J. Baud, J. Casey, S. Lemaitre, C. Nicholson, G. Stewart. LCG Data Management. From EDG to EGEE.

[5] MySQL , The world's most popular open source database, http://www.mysql.com

[6] ALICE, the ALICE experiment, http://www.cern.ch/ALICE/

[7] CERN, European Organization for Nuclear Research, http://www.cern.ch

[8] EGEE, Enabling Grids for E-Science, http://public.eu-egee.org