UNIVERSITY of GLASGOW

Department of Physics & Astronomy
Experimental Particle Physics Group
Kelvin Building, University of Glasgow,
Glasgow, G12 8QQ, Scotland
Telephone: +44 (0)141 339 8855 Fax: +44 (0)141 330 5881

GLAS-PPE/2005-01
15th February 2005

Performance Analysis of a File Catalog for the LHC Computing Grid

Jean-Philippe Baud[2], James Casey[2], Sophie Lemaitre[2], Caitriana Nicholson[1]

[1] University of Glasgow, Glasgow, G12 8QQ, Scotland

[2] CERN, European Organization for Nuclear Research, 1211 Geneva, Switzerland

Abstract

The Large Hadron Collider (LHC) at CERN, the European Organization for Nuclear Research, will produce unprecedented volumes of data when it starts operation in 2007. To provide for its computational needs, the LHC Computing Grid (LCG) will be deployed as a worldwide computational grid service, providing the middleware upon which the physics analysis for the LHC will be carried out. In 2003, versions of this middleware were deployed which were based on the middleware produced by the European Data Grid project (EDG). In 2004 the LCG-2 release, which consisted of the EDG middleware with some minor modifications, was deployed for use by the LHC experiments.

A series of data challenges by these experiments were the first real experiment production use of LCG. During the course of the data challenges, many issues and problems were exposed which had not shown up in more limited tests. The deployment, service and development teams worked closely with the experiments to understand these issues and while some of the problems were solved during the data challenges, others exposed fundamental problems with the middleware as deployed in LCG-2.

One of these fundamental problems was the performance under real load of the catalog component provided by EDG, the Replica Location Service. To solve these problems a new component was designed, the LCG File Catalog (LFC). The LFC moves away from the Replica Location Service model used in previous LCG releases, towards a hierarchical filesystem model which is more like a UNIX filesystem. It also adds missing functionality which was requested by the experiments.

This paper presents the architecture and implementation of the LFC and evaluates it in a series of performance tests, with up to forty million entries and one hundred requesting threads from multiple clients. The results show good scalability up to the limits of these tests, and compare favourably with other Grid catalog implementations.

14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)
Research Triangle Park, North Carolina, USA

1 Introduction

The 2004 series of LHC experiment data challenges were the first to use the LCG-2 set of middleware tools [13] in a realistic environment. The CMS (Compact Muon Solenoid) collaboration, for example, aimed to reach a complexity of about 25% of that predicted for the initial running of the LHC. Feedback from the experiment groups after the data challenges highlighted various problems and limitations, as well as differences between the expected and actual usage patterns.

With particular reference to the data management components of LCG-2, it became apparent that the file catalog infrastructure, which consisted of the EDG Local Replica Catalog (LRC), Replica Metadata Catalog (RMC) (which together form the Replica Location Service or RLS) and Replica Manager, was too slow both for inserts and for queries [9]. Queries involving both the LRC and RMC were particularly slow [12]. Missing functionality identified included lack of support for bulk operations and transactions. It also became clear that queries were generally based on metadata attributes and were not simple lookups of a file's physical location. On the other hand, users did not use the web services approach in the way which had been anticipated when the EDG components were developed. They were implemented such that a remote procedure call (RPC) was performed for each low-level operation; users, however, wanted to send higher-level or multiple commands in a single RPC. As this was not available, the cumulative overheads from a large number of low-level RPCs led to considerable loss of performance. Also, although a C++ API was available, command-line tools were available only in Java which led to added loss of performance due to the overhead in starting up the Java Virtual Machine with each call.

In response to this feedback, a proposal was put forward, accepted and subsequently implemented to develop the LCG File Catalog (LFC) as an immediate replacement for the EDG catalogs. Initial performance tests of the LFC were presented in [2], and this paper presents the results of full testing. The RLS framework was designed jointly by Globus and EDG [8], then two different implementations were produced; here, the LFC performance is compared to that of both the EDG RLS [10] and Globus RLS [6].

The structure of this paper is as follows. First, a description of the architecture, implementation and main features of the LFC is given. The methodology and results of the performance tests are then shown with some discussion of their significance. Some pointers to future work are then given before discussing related work, summarising and drawing conclusions.

2 LFC Architecture

The LFC has a completely different architecture from the RLS framework. Like the EDG catalog, it contains a GUID (Globally Unique Identifier) as an identifier for a logical file, but unlike the EDG catalog it stores both logical and physical mappings for the file in the same database. This speeds up operations which span both sets of mappings. It also treats all entities as files in a UNIX-like filesystem, and is thus closer in concept, logically, to the AliEn File Catalog [14] (see Section 7). The API is designed to mimic a UNIX filesystem API, with calls which are intuitive to the user, such as `creat`, `mkdir` and `chown`.

The main entities of the LFC design are shown in Figure 1. There is a global hierarchical namespace of Logical File Names (LFNs) which are mapped to the GUIDs. GUIDs are mapped to the physical locations of file replicas in storage (Storage File Names or SFNs). System attributes of the replicas (such as creation time, last access time, file size and checksum) are stored as attributes on the LFN, but user-defined metadata is restricted to one field, as the authors believe that user metadata should be stored in a separate metadata catalog. Multiple LFNs per GUID are allowed as symbolic links.

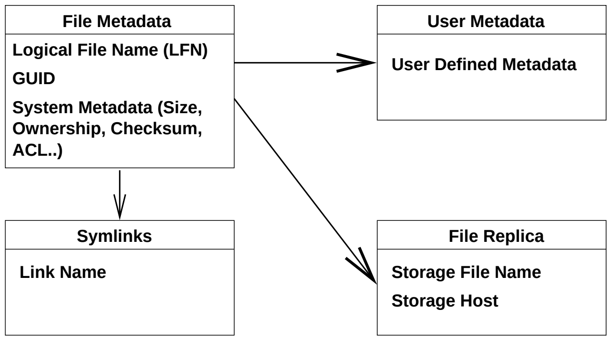

Figure 1: Components of the LFC.

Bulk operations are supported, with transactions, and cursors for handling large query results. As there is only one catalog, transactions are possible across both LFN and SFN operations, which was impossible with the EDG RLS. In case of momentary loss of connection to the catalog, timeouts and retries are supported.

Authentication is by Grid Security Infrastructure (GSI), which will allow single sign-on to the catalog with users' Grid certificates. The client domain name is mapped internally to a uid/gid pair which is then used for authorization. It is also planned to integrate VOMS (the Virtual Organisation Membership Service which was developed in EDG) as another way of authentication, mapping the VOMS roles to multiple group IDs in the LFC. The uid/gid pairs are used for authorization by means of the file ownership information which is stored in the catalog as system metadata on the LFN. Standard UNIX permissions and POSIX-compliant Access Control Lists (ACLs) on each catalog entry are supported.

3 LFC Implementation

The LFC is implemented entirely in C and is based on the CASTOR [5] Name Server code. There is no web services implementation, which allows integration with external catalog interfaces such as gLite FiReMan [7]. It runs as a multi-threaded daemon, with a relational database backend. Currently, both Oracle and MySQL are supported as database components. The client may also be multi-threaded.

Bulk operations are implemented inside transactions. The transaction API is also exposed to the user, both to allow multiple operations inside a single transaction and to allow user-controlled as well as automatic rollback.

The GSI security implementation depends on an external library from CASTOR and is still being integrated, but the permissions infrastructure described above has been implemented. VOMS integration is also pending.

Clients currently exist for GFAL, POOL and lcg_utils. GFAL is the Grid File Access Library, a library developed by LCG to give a uniform POSIX interface to local and mass storage on the Grid; POOL is the Pool of persistent Objects for LCG, a framework for the LHC experiments to navigate distributed data without knowing details of the storage technology [12]; and lcg_utils is the command-line interface and API for user access to LCG [2].

4 Performance Test Methodology

For the performance tests, requests were submitted to the LFC via multi-threaded client programs written in C. Each test program allowed the user to specify the number of operations to perform (inserts, deletes, etc), the number of threads to be used, and other options relevant to each specific test. The C programs were then wrappered by Perl scripts which ran each batch of tests. Typically, each operation was performed several thousand times in the C program and the mean time taken; this was then called several times from the Perl script and an overall mean taken. Any entries added to the LFC were removed before the next test run.

5 Performance Test Results

In this section, the results of the performance tests conducted on the LFC are presented, first with a single client and then with multiple clients connecting to the same server. The server was a dual Pentium III 1 GHz processor with 512 MB of memory, with an Oracle database back end which was running on a dual Intel Xeon 2.4 GHz machine. The server operating system was Red Hat Linux 7.3 and for the majority of these tests it was configured to run with 20 threads.

For the single client tests, the client machine was a Pentium III 853 MHz processor with 128 MB of memory, running Red Hat Linux 7.3. Server and client machines were connected by a 100 Mb/s local area network.

For the multiple client tests, 10 Pentium IIIs were used as clients. Each had 1 GHz CPU, 512 MB memory and were running CERN Scientific Linux 3.0.3. The server was as for the single client tests, and all the machines were on the same 100 Mb/s local area network.

A comparison of the hardware used for these tests with that used in the EDG and Globus RLS tests, in terms of the SPEC CINT2000 values [15] of the machines used, is shown in Table 1. Dual CPUs are indicated by a 2× in front of the CINT 2000 value for a single CPU. When comparing the results in Sections 5.1 and 5.2, the difference in values for the server in particular should be kept in mind.

	LFC	Globus RLS	EDG RLS
Server	2 × 420	2 ×810	2 × 420
Single Client	420	2 × 220	2 × 420
Multiple Clients	400	2 × 220	2 × 420

Table 1: Approximate SPEC CINT2000 values for test machines.

5.1 Single Client Tests

5.1.1 Add tests

Starting from a near-empty database, the mean time to add a single entry was measured as the number of entries in the LFC increased, up to about 40 million entries. As Figure 2 shows, the mean add time is roughly constant at about 22 ms until there are of the order of 5 million entries, when it increases slowly up to a plateau around 27 ms per entry. The EDG RLS, on the other hand, started to show significant increase in the insert time from 200,000 entries and upwards. As shown in [4], the mean insert time starts at about 20 ms per insert but by the time there are 500,000 entries in the catalog the insert time is about 40 ms per insert.

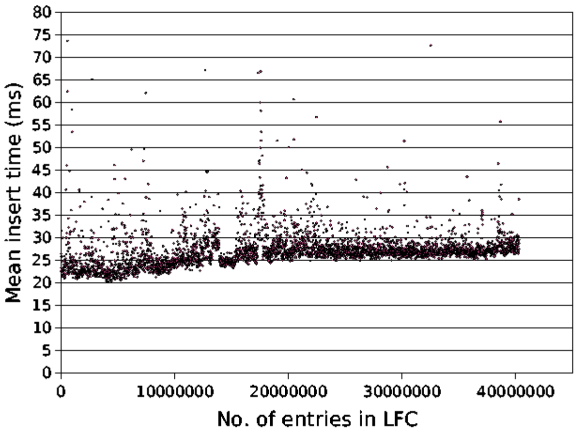

Figure 2: Mean add time with increasing catalog size.

Next, the add rate as a function of number of client threads was examined, for approximately 1 million entries in the LFC (Figure 3). This shows the add rate increasing up to approximately 200 adds per second, up to the limit of server threads (after which the rate falls again). Figure 3 also shows the add rate achieved by the Globus RLS when database flush is enabled (i.e. when transactions are written to the physical disk immediately) [6], which is about 84 adds per second independent of the number of threads. The authors consider enabling of database flush to be essen-

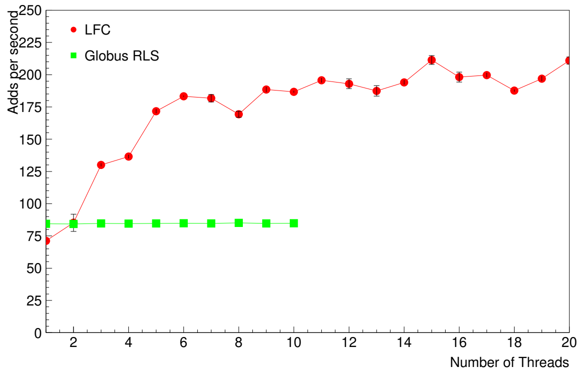

Figure 3: Add rate for increasing number of client threads.

tial for data consistency and hence do not consider the case where database flush is disabled.

With the LFC, it is possible to change into the working directory in the LFC namespace by a `chdir` operation. This leads to improved performance when there are multiple operations in the same directory, as permissions only need to be checked once (when the `chdir` is performed) rather than with every operation. The extent of this performance improvement was measured by examining the mean add time as a function of the number of subdirectories in the path to the file to be inserted. The comparative performances of doing a `chdir` then using the relative pathname and of staying in the top directory and using the absolute pathname are shown in Figure 4. It is clear from these re-

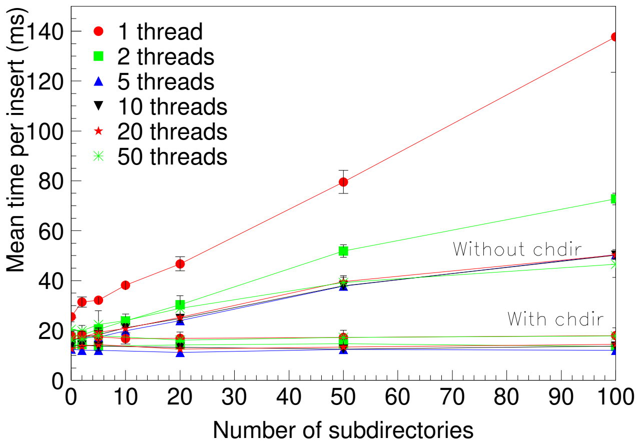

Figure 4: Mean add time with and without chdir, with increasing number of subdirectories in the path.

sults that when `chdir` is used, the add time is independent of the number of directories in the path, whereas when the absolute names are used, the time taken increases proportionally. The authors therefore recommend that application code includes a call to `chdir` if a large number of operations are envisaged in one directory.

The performance when multiple catalog operations are done inside database transactions was then investigated. The mean add time was measured as the number of adds included in a single transaction was varied, and the results are shown in Figure 5. This shows that the optimal number of

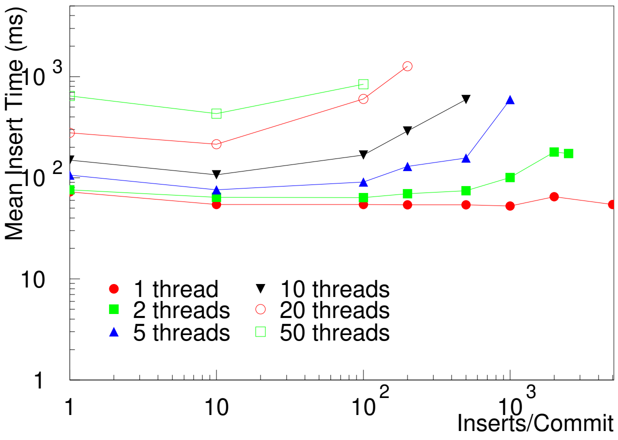

Figure 5: Mean add time when using transactions, with increasing number of adds per transaction.

operations per transaction is between 10 and 100, but that even at this optimal number, using transactions increases the insert time by at least a factor of 2. After about 100 operations per transaction, the insert time increases sharply.

To understand the reason for this loss of performance, each stage of the transaction was examined: waiting to start, actually running, and ending the transaction. Figure 6 shows the mean time taken for each of these stages, as well as for the whole transaction, as the number of operations per transaction increases.

This shows that the wait time is approximately 5-10 ms per transaction, until the number of client threads is greater than the number of server threads, when transactions spend upwards of 50 ms waiting for a thread to become available. The time for ending a transaction varies randomly between 10 and 100 ms, and when this is combined with the wait time it results in some of the observed performance loss. It is therefore clear that the extra time is not being spent in the LFC code, but comes from the database, and should be investigated further at that level.

5.1.2 Delete tests

The delete rate was also measured for a varying number of client threads, with 1 million entries in the LFC, and the results are shown in Figure 7. The delete rate is about 150 deletes per second, up to 20 threads; beyond that, the rate falls as there are fewer server threads than client threads. The case where there is 1 client thread, where the time per delete is about 16 ms, can be compared with the corresponding result for the EDG RLS in [4], where the time per delete is about 30 ms. There are no directly comparable results available with the Globus RLS.

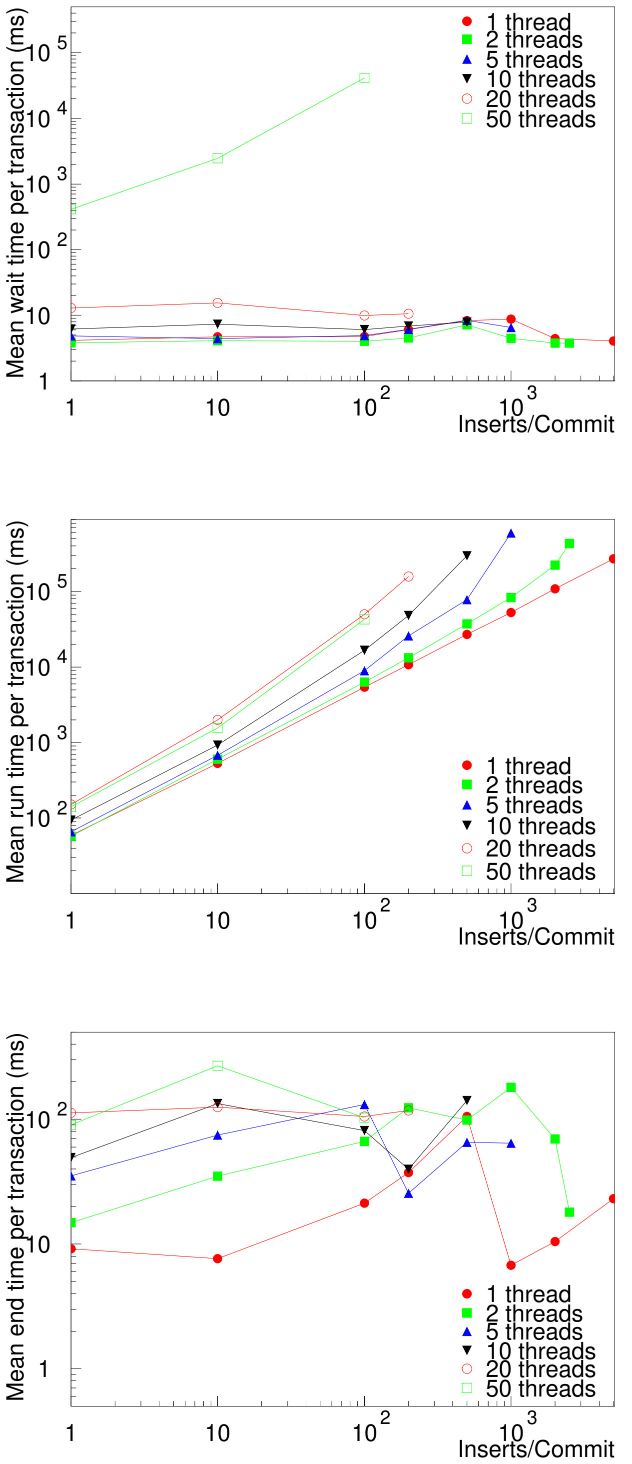

Figure 6: Mean time spent by transactions in (a) waiting, (b) running and (c) finishing.

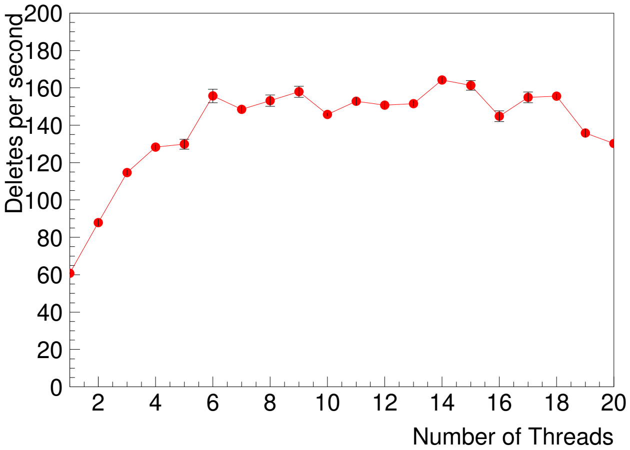

Figure 7: Delete rate for increasing number of client threads, LFC with 1 million entries.

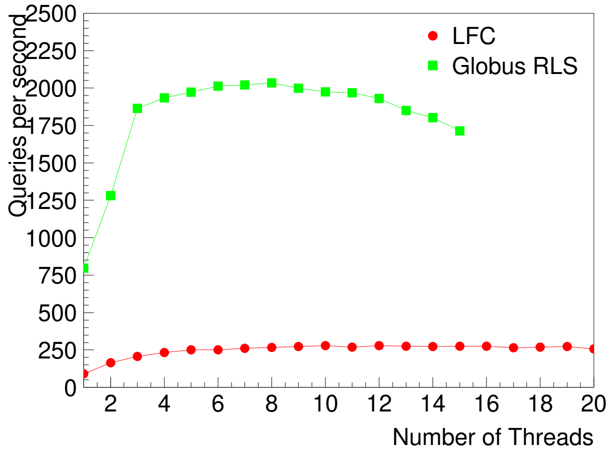

Figure 8: Query rate for increasing number of client threads, LFC with 1 million entries.

5.1.3 Query tests

The rate at which the LFC can perform queries for a single filename was measured, with 1 million entries, for an increasing number of client threads. As seen in Figure 8, the maximum rate is about 275 queries per second, for up to 20 threads. This is clearly lower than the rates achieved in [6], but it is hard to draw a direct comparison between the two, as the Globus RLS results are for a simple LFN to SFN lookup, while the LFC performs a `stat()` operation on the LFN. This returns all the associated metadata such as file size, checksum, modification time and so on, as well as checking the ACLs. The pseudo-code in Figure 9 shows the steps which are performed. Observation of actual user patterns [12] suggests that users rarely perform a simple lookup, but tend to request some items of metadata with the SFN.

```
foreach ( component of LFN pathname )
    get file metadata
    check entry permissions
if ( GUID is supplied )
    check that GUIDs match
return LFN and file metadata
```

Figure 9: Pseudo-code for a `stat()` operation

The results reported for the EDG RLS, on the other hand, give a query time (for a single client thread) of about 16 ms. This corresponds to a rate of about 63 queries per second, whereas the LFC query rate for a single thread is about 90 queries per second. It should be taken into consideration that the LFC performs an SQL `ORDER BY` statement when listing the entries in a directory. With large directories, this naturally has an impact on the performance. In the EDG RLS, there was no `ORDER BY`, which meant it was not possible to implement safe iterators or cursors.

Next, the time to read all the files in a directory is measured for an increasing number of files in the directory. Figure 10 shows that the total time taken increases linearly with the number of files in the directory, up to 20 client threads. At this point, the time spent waiting for a thread to become available dominates, and so the total time taken becomes independent of the directory size until there are sufficiently many entries (more than about 3000) in the directory for the read time to dominate. With fewer than 20 threads, the average time to perform a read and `stat()` operation on each file in the directory is between 10 and 20 ms. It is not yet clear why the wait time becomes dominant at 20 threads (when the number of client threads is exactly equal to the number of server threads) rather than at 21 threads, but this is undergoing further investigation.

The time to list and get the status information for all symlinks in a directory shows very similar behaviour to the read time described above, as Figure 11 shows, except that in this case the server was running with only 6 threads, so the wait time becomes dominant for all measurements with more than 5 client threads. The time to traverse a chain of symbolic links until the original file was reached was also

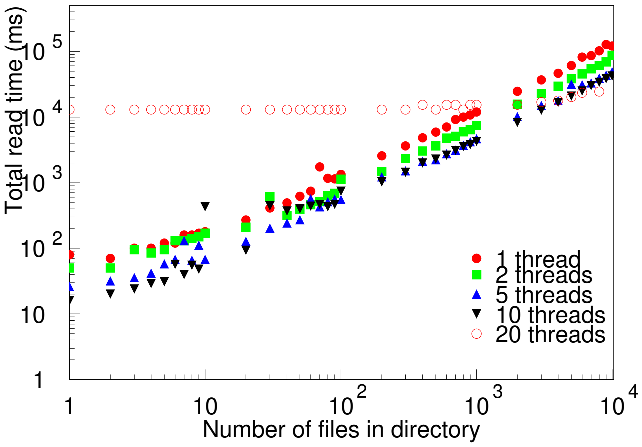

Figure 10: Total read time with increasing directory size, for varying number of client threads, 20 server threads.

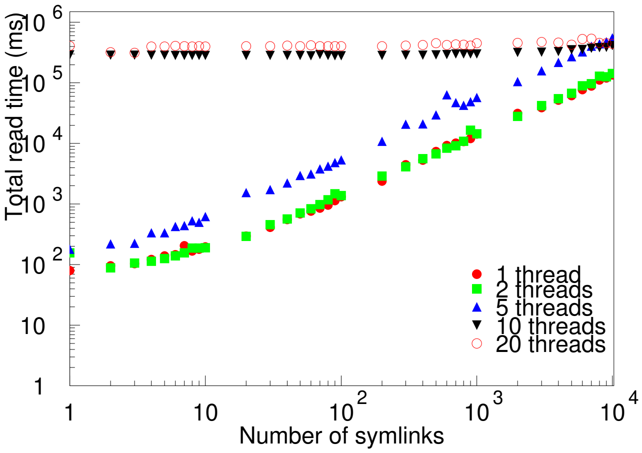

Figure 11: Total time to list and stat all symlinks in a directory, for varying number of client threads, 6 server threads.

measured, for an increasing number of links in the chain. These results are shown in Figure 12.

The time to list and `stat()` all the replicas of a file (Figure 13), for an increasing number of replicas of the file, also shows similar behaviour to the read case. In this case the server was running with 20 threads. All of these tests of the behaviour with increasing number of files, symbolic links and replicas are as expected; comparable results for the two RLS implementations are not available.

5.2 Multiple Client Tests

A subset of the tests described in Section 5.1, namely those measuring add and query rates, were run simultaneously on multiple clients, each running with 10 threads. Figure 14 shows the behaviour of the operation rates as the number of clients increases when transactions are not used, while

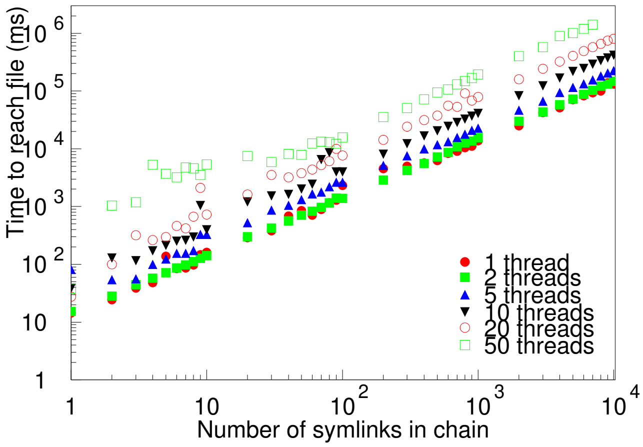

Figure 12: Total time to traverse a chain of symlinks, for increasing number of symlinks and varying number of client threads, 20 server threads.

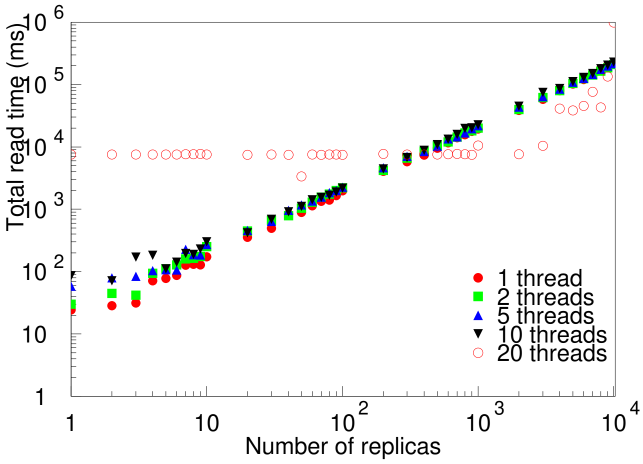

Figure 13: Total time to list and stat all replicas of a file, for varying number of client threads, 20 server threads.

Figure 15 shows the rates when transactions are used, with 100 operations per transaction.

These show that the operation rates are essentially independent of the number of clients, up to the limits of the test. It should be borne in mind that the standard error on the data points in these plots is of the order of 10-20 operations per second and hence the actual variation from point to point is less marked than the plots seem to indicate.

What is clear is that the catalog is very scalable and shows no significant reduction in operation rate as the number of clients increases. When transactions are used, the loss of performance observed in the single client case is less apparent; in fact, performance is as good, within experimental uncertainty, as that without transactions.

The authors believe that the reason for the constant operation rates with multiple clients is that the server is running at maximum load. Examination of the client-side rates shows that the load is being shared equally among the clients, and more detailed investigation is required to understand where exactly the bottlenecks lie.

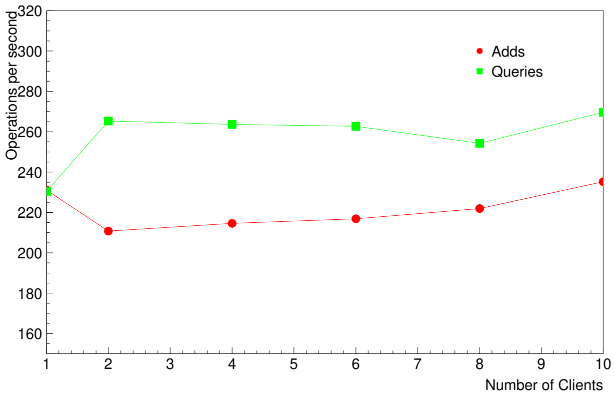

Figure 14: Operation rates with increasing number of clients, no transactions.

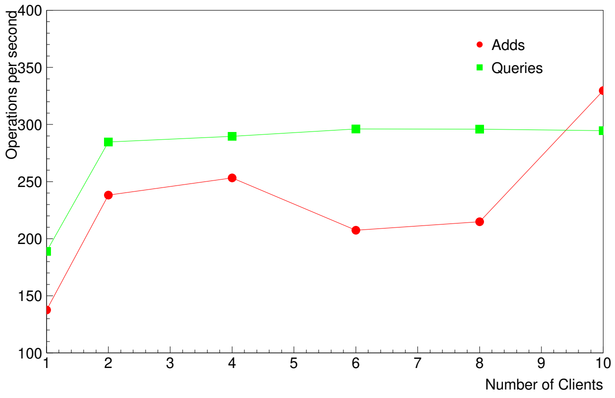

Figure 15: Operation rates with increasing number of clients, 100 operations per transaction.

6 Future Work

Immediate next steps for the LFC are the integration of GSI security and integration with VOMS, as discussed in Section 2. Future development can then aim at distributed catalogs rather than a single central server, possibly using a peer-to-peer update mechanism. The hierarchical structure also allows the possibility of partitioning the catalog on the LFN namespace rather than the simple hashing of GUIDs, which is the only partitioning possible in a flat namespace.

Another possibility is to have closer coupling of the LFC with the Disk Pool Manager (DPM) [2], which manages resources on local storage. This would make local storage catalogs directly part of the global Grid catalog, thus removing inconsistencies and problems with update propagation.

7 Related Work

There are several related Grid file management solutions. Among them are the EDG Replica Location Service [10] and Globus Replica Location Service [6], which are both implementations of the RLS Framework which was defined in collaboration by EDG and Globus [8]. A peer-to-peer RLS system based on the Globus RLS is also being researched and a prototype is presented in [3].

Somewhat more similar in concept to the LFC is the ALICE Environment (AliEn), which is a lightweight Grid framework developed by the Alice collaboration, one of the LHC experiment groups. In particular, the AliEn FileSystem (AliEnFS) [14] also offers a UNIX-like virtual file system with access via a C++ API. The gLite catalog service [7] also presents the abstraction of a global filesystem, as does Gfarm [16], which provides a POSIX-compliant secure network file system coupled to the Grid's compute resources, so that computation is always sent to the data.

The Storage Resource Broker [1] allows applications to access heterogeneous storage resources by using metadata attributes or logical filenames rather than physical filenames or locations.

Also relevant, especially to the high energy physics Grid community, is the SAM (Sequential Access via Metadata) [11] project, which handles data for the D0 and CDF projects at the Fermi National Accelerator Laboratory near Chicago.

8 Conclusions

The LCG File Catalog has been presented, outlining its implementation and features. In particular, the features required by users but unavailable in other replica management systems such as the EDG and Globus RLSs have been highlighted. These include transaction support, checksums, ACLs and a UNIX-like virtual file system. Other aspects, such as the information returned by a query, have also been implemented according to observed user patterns. The results of a series of performance tests using both single and multiple clients accessing the server have shown the robustness and scalability of the LFC up to many millions of entries and hundreds of client threads.

Acknowledgments

This work has been part funded by the European Union under contract number INFSO 508833 through the LCG project. Sophie Lemaitre would like to thank the Bundesministerium für Bildung und Forschung, Berlin, Germany and Caitriana Nicholson would like to thank the Particle Physics and Astronomy Research Council, Swindon, UK for their respective funding. Thanks to Ann Chervenak and Naveen Palavalli for data from the Globus RLS and to David Cameron for information on the EDG RLS.

References

[1] Chaitanya Baru, Reagan Moore, Arcot Rajasekar, and Michael Wan. The SDSC Storage Resource Broker. In *CASCON'98 Conference*, Toronto, Canada, November 1998.

[2] J-P. Baud and J. Casey. Evolution of LCG-2 Data Management. In *Computing in High Energy and Nuclear Physics (CHEP)*, Interlaken, Switzerland, September 2004.

[3] M. Cai, A. Chervenak, and M. Frank. A Peer-to-Peer Replica Location Service Based on a Distributed Hash Table. In *SuperComputing 2004*, Pittsburgh, PA, USA, November 2004.

[4] David G. Cameron. Replica Management and Optimisation for Data Grids, November 2004. Ph.D. Thesis, University of Glasgow.

[5] CERN Advanced Storage Manager. `http://castor.web.cern.ch/castor/`.

[6] A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and Scalability of a Replica Location Service. In *13th IEEE Symposium on High Performance Distributed Computing (HPDC-13)*, Honolulu, Hawaii, USA, June 2004.

[7] EGEE Design Team. EGEE Middleware Architecture and Planning, August 2004. CERN Document EGEE-DJRA1.1-476451-v1.0.

[8] A. Chervenak et al. Giggle: A Framework for Constructing Scalable Replica Location Services. In *SuperComputing 2002*, Baltimore, MD, USA, November 2002.

[9] A. Fanfani et al. Distributed Computing Grid Experiences in CMS DC04. In *Computing in High Energy and Nuclear Physics (CHEP)*, Interlaken, Switzerland, September 2004.

[10] D. Cameron et al. Replica Management Services in the European DataGrid Project. In *e-Science All Hands Meeting*, Nottingham, UK, September 2004.

[11] Igor Terekhov et al. Distributed Data Access and Resource Management in the D0 SAM System. In *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, San Fransisco, CA, USA, August 2001.

[12] M. Girone et al. Experience with POOL in the LCG Data Challenges of Three LHC Experiments. In *Computing in High Energy and Nuclear Physics (CHEP)*, Interlaken, Switzerland, September 2004.

[13] A. Delgado Peris, P. Méndez Lorenzo, F. Donno, A. Sciabà, S. Campana, and R. Santinelli. LCG-2 User Guide. Technical Report CERN-LCG-GDEIS-454439, CERN, Geneva, Switzerland, September 2004.

[14] Andreas-J. Peters, P. Saiz, and P. Buncic. AliEnFS - a Linux File System for the AliEn Grid Services. In *Computing in High Energy and Nuclear Physics (CHEP)*, La Jolla, CA, USA, March 2003.

[15] Standard Performance Evaluation Corporation. `http://www.spec.org/cpu2000/results/cint2000.html`.

[16] O. Tatebe, S. Sekiguchi, Y. Morita, N. Soda, and S. Matsuoka. Gfarm v2: a Grid File System that Supports High-Performance Distributed and Parallel Data Computing. In *Computing in High Energy and Nuclear Physics (CHEP)*, Interlaken, Switzerland, September 2004.