Performance studies of the StoRM Storage Resource Manager

A. Carbone[†], L. dell'Agnello[†], A. Forti[†], A. Ghiselli[†], E. Lanciotti[‡], L. Magnoni[†], M. Mazzucato[†], R. Santinelli[‡], V. Sapunenko[†], V. Vagnoni[§] and R. Zappi[†]

[†] *INFN CNAF, viale Berti-Pichat 6/2, I-40127 Bologna, Italy*
[‡] *CERN, European Organization for Nuclear Research, CH-1211, Geneve, Switzerland*
[§] *INFN Sezione di Bologna, via Irnerio 46, I-40126 Bologna, Italy*

Abstract

High performance disk-storage solutions based on parallel file systems are becoming increasingly important to fulfill the large I/O throughput required by High-Energy Physics applications. Storage Area Networks (SAN) are commonly employed at the Large Hadron Collider data centres, and SAN-oriented parallel file systems such as GPFS and Lustre provide high scalability and availability by aggregating many data volumes served by multiple disk-servers into a single POSIX file system hierarchy. Since these file systems do not come with a Storage Resource Manager (SRM) interface, necessary to access and manage the data volumes in a Grid environment, a specific project called StoRM has been developed for providing them with the necessary SRM capabilities. In this paper we describe the deployment of a StoRM instance, configured to manage a GPFS file system. A software suite has been realized in order to perform stress tests of functionality and throughput on StoRM. We present the results of these tests.

1 Introduction

Dependable high throughput data services for storage and file access are the heart of Grid computing, in particular for the High-Energy Physics (HEP) community facing the challenges of the new scientific program starting at the Large Hadron Collider (LHC) [10]. In particular, the I/O load and the large data size produced by the LHC experiments, amounting to several PB/year, together with the need to exchange data with CERN and with all the other computing centres around the world at a steady rate of several Gb/s, pose very stringent requirements for all the data centres involved.

The data management is a fundamental task requiring a coordinated effort, and is realized through common tools and protocols allowing to start, control and end the transfers between the different Storage Elements (SE) at the different sites. The SEs must be fully manageable from remote users, such as experiment production managers, in order to perform all the basic operations such as space management and data access. In addition, the data stored on the SEs must be protected against unauthorized access, and the access rules have to be defined as required by the experiments.

One of the main goals of the Grid consists in providing a way to share geographically distributed heterogeneous storage resources, with an effective and common interface to users, regardless of the type of the back-end system being used. In a data Grid, in which more emphasis is given on data intensive applications that produce and read large volumes of data, the need to provide an efficient management of the storage resources becomes mandatory.

Given the above requirements, the HEP Grid community has adopted and implemented the Storage Resource Manager (SRM) interface [15, 27], allowing for a fully transparent management and access to underlying storage resources, hiding the details of the back-end implementation to the user applications. The Grid Storage Resource Manager (StoRM) [24, 17] has been developed with the specific aim of providing parallel file systems like GPFS [6, 26] and Lustre [12, 20], but also standard POSIX file systems, through a SRM interface.

In this paper we present the results of a series of tests performed on the StoRM SRM implementation at CNAF [1], i.e. the central computing facility of the Italian National Institute for Nuclear Physics (INFN). Some basics of the SRM specifications will be introduced, also briefly mentioning the SRM implementations presently available. Later, some details on the StoRM architecture and its main features will be given. Then, after having introduced the StoRM test-bed put in place for this study, the tests performed and the rele-

[1] CNAF, Centro Nazionale per la Ricerca e Sviluppo nelle Tecnologie Informatiche e Telematiche, `http://www.cnaf.infn.it`.

vant results obtained will be described.

2 Storage resource management

A storage service interface must be flexible enough to be used to access storage systems based on different storage technologies. At the same time, its usage should be simple enough so that clients do not need to have any knowledge of the underlying storage back-end. This is the main goal of the SRM interface, i.e. a middleware service aiming to provide the dynamic management of a storage resource engaged in a distributed computing system. A detailed description of SRM and its specifications is given in references [15, 16].

SRM allows to access a file-based storage system. Files are addressed as *site URL* (SURL) i.e. logical entities providing an abstraction of the file name-space, and as *transfer URL* (TURL) i.e. physical data items.

The SRM can have the files stored in several physical locations, or can bring them from tape to disk for direct access by a client. Once the file is available for I/O, a TURL is returned for a temporary access to the file controlled by the pinning lifetime. A similar capability exists when a client wishes to put a new file into the SRM.

To access a file through a SRM service, a SURL is provided as a parameter embedded in the specific SRM request, such as the *srmPrepareToPut* method for preparing the SRM to accept new data and the *srmPrepareToGet* method to get read access to the wished data resource. The SRM interface provides two classes of methods: asynchronous and synchronous ones. Asynchronous methods return a token corresponding to the request, and the corresponding action is performed by the SRM asynchronously. The client can retrieve at any time the status of the request by addressing it through such a token. This is the case for data access functionalities. Synchronous requests return at completion giving back the control to the client (*blocking calls*). This is the case of directory and file management (e.g. *srmLs*, *srmMkdir*, *srmRm* and *srmRmdir*) and space management functions (e.g. *srmReserveSpace*).

2.1 SRM implementations

In this section a brief overview of the most widely used SRM implementations is given.

The Berkeley Storage Manager (BeStMan) [1] is a Java-based SRM implementation from LBNL [2]. It has a modular design and it is currently used for managing disk-servers and the HPSS Mass Storage System (MSS) [8].

The CERN Advanced Storage System (CASTOR) [2] is designed to work with a tape back-end to provide efficient data transfer to tape and reliable data access to the front-end disk cache.

The dCache [25] system is again a Java-based SRM and can be used as pure disk SE or can be interfaced to a MSS to be used in conjunction with tapes. It is realized by a collaboration between FNAL [3] and DESY [4], and it is characterized by a highly scalable architecture.

The Disk Pool Manager (DPM) [4] project, developed at CERN, is designed to manage a disk-storage system. It has a dedicated daemon to provide the SRM interface.

Finally, the StoRM project has been developed by a collaboration between INFN and the Abdus Salam ICTP institute in Trieste, the latter operating in the framework of the EGRID [19] project. The main idea behind StoRM consists in decoupling the file system from the SRM service, in the sense that the data-access functionality can be provided by any POSIX file system, while StoRM specifically provides just the SRM functionality. In the case that any advanced functionality peculiar to a given file system were available via specific APIs, they could be included into a StoRM driver bound to the file system and plugged in into StoRM.

StoRM can be easily configured to run on the GPFS [6, 26] or Lustre [12, 20] parallel file systems, which provide high-performance POSIX access to the data and are widely employed commercial products.

In the following section we will give a more extended description of the StoRM architecture.

3 The StoRM Storage Resource Manager

StoRM [24, 17]is an implementation for disk-based storage of the version 2.2 of the SRM interface especially designed to satisfy the requirements coming from the HEP experiments .

The main idea behind StoRM is to provide a SRM access to high performance parallel file systems in a Grid environment. It provides users and applications with the standard SRM functionality combined with the capability to perform secure and local access to the shared storage resources (e.g. through GPFS).

StoRM provides the advanced SRM functionalities to dynamically manage space and files according to the user requirements, to specify the desired lifetime and to allow for advance space reservation and a different quality of service provided by the underlying storage system. Since it generalizes the file system access to the Grid, StoRM also takes advantage from the file system security mechanisms to ensure an authenticated and authorized access to the data.

[2] LBNL, Lawrence Berkeley National Laboratory, `http://www.lbl.gov`.

[3] FNAL, Fermi National Accelerator Laboratory, `http://www.fnal.gov`.

[4] DESY, Deutsches Elektronen-Synchrotron, `http://www.desy.de`.

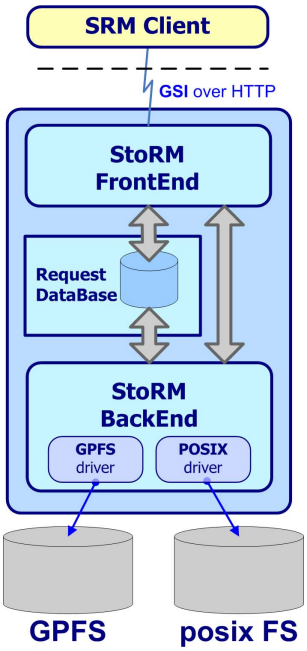

Figure 1. Schematic representation of the StoRM architecture.

StoRM provides a layered and flexible security framework to satisfy the requirements coming from the different scenarios involved. User authorization is based on certificates, authorization decisions can be performed interacting with different external Grid services and, to provide a local secure access to data, POSIX ACLs are enforced on the desired files and directories.

3.1 StoRM architecture and features

StoRM has a multi-layer architecture made by two main components: the *front-end* (written in C/C++), and the *back-end* (written in Java), with a database system used to store SRM requests and the StoRM metadata. The front-end exposes the SRM web service interface, manages user authentication and stores the data of the SRM requests into the database. The back-end is the core of the StoRM service; it executes all synchronous and asynchronous SRM functionalities. It takes care of file and space metadata management, enforces authorization permissions on files and interacts with external Grid services. Support for the different file systems is provided in StoRM by a driver mechanism. The back-end logic is decoupled from this wrapper component and the specific driver can take advantage from proprietary functionalities provided by certain file systems (e.g. block preallocation in GPFS). A sketch of the architecture is shown in Fig. 1.

The driver implements a common internal interface for the underlying file system in use, allowing a StoRM instance to work on different file systems at the same time. This solution provides a Grid site with the capability of using different services for different functionalities at the same time, i.e. StoRM exposes the SRM functionalities, and different storage systems for managing and aggregating the storage resources, such as GPFS [6], Lustre [12] or XFS [18] can be used. At present there are two specific drivers for the GPFS and XFS file systems, that rely on proprietary APIs for the management of advanced functionalities, and in addition a generic POSIX driver which works for every file system with POSIX semantics, such as Lustre or others.

An important feature for a SRM service is the naming capability. In a Grid environment it allows users to refer to specific data resources in a physical storage system using a high level logical identifier (SURL). This logical identifier is typically defined with a file system like structure (a hierarchical tree of names) independently on the physical location of data on the real storage. According to the main idea, StoRM has a name-space mechanism that relies on the underlying storage system structure to create the space of file names. It is based on a XML document that represents the different storage components managed by the service, the storage areas defined by the site administrator, the quality of service they provide and the VO that wants to use the storage area. An appropriate directory tree is realized in each storage component reflecting the XML schema. In this scenario, StoRM is able to identify the physical location of a requested data evaluating the logical identifier and the specified attributes following the XML schema, without querying any database service.

SRM services have to support the Grid security at different levels, from the user authentication and authorization to the permission enforcement on files and directories for the data access. StoRM provides a layered and flexible security framework, built to satisfy the requirements coming from heterogeneous scenarios, as the case of HEP and the more security demanding financial Grid use cases. The authorization is based on X.509 certificates and it supports the groups and roles attributes as defined by the VOMS service [22]. To verify if a user is authorized to perform a certain operation on the data, StoRM is able to interact with external authorization services (so-called authorization sources). Authorization sources contain information on users permissions and can be local or global. The former are based on local configuration (e.g. XML files containing regular expressions of path and user credentials), the latter are external services, such as file catalogs (e.g. the LCG File Catalog [9]) or dedicated policy decision services (e.g. the G-PBox service [23]). To provide a secure and local access to the data, StoRM relies on the security mechanisms provided by the underlying file system in order to enforce permissions. When an authorized user requests to access a certain file

StoRM enforces an appropriate ACL on the physical file and directories with the local identity corresponding to the (mapped) user credential. With this solution a direct local access can be performed complying with all Grid security layers.

Together with StoRM, a command line client tool for the SRM version 2.2 interface has been realized, and we have extensively used it for the tests described in the following sections of this paper. It provides the capability to interact with a generic SRM version 2.2 service - i.e. it is not specific to the StoRM service - using the standard UNIX command style for options, help and command output. It supports all the SRM version 2.2 functionalities.

3.2 StoRM clusterization

In order to satisfy the high availability and scalability requirements coming from the HEP community, StoRM can be deployed in a clustered configuration, with multiple instances of front-end and back-end services and with a dedicated DBMS installation. The communication between the different services takes place using the database to store SRM request data and a direct RPC connection. For the load balancing a simple dynamic DNS configuration can be used. Additional new front-end services can be easily added on-the-fly to improve the rate of requests managed by the system, while new back-ends can be added to work on the same database to improve the global processing time of the SRM requests. The DBMS (currently only MySQL is supported, but the porting to Oracle is foreseen) can be deployed in a dedicated single or clustered configuration, completely independent on the other StoRM services.

4 Test-bed layout

Our setup was composed by 4 disk-servers integrated into the CNAF SAN via FC links, while the communication of the disk-servers with the computing farm was via Gigabit LAN. As disk-storage hardware we used part of one EMC CX3-80 SAN system for a total of 40 TB of raw-disk space arranged in RAID-5 arrays. The disk arrays were aggregated on the 4 disk-servers by GPFS, version 3.1.0-10, i.e. the disk-servers actually acted as GPFS servers, serving a net 36 TB GPFS file system to the worker nodes.

GPFS is a high-performance shared-disk cluster file system developed by IBM. GPFS distinguishes itself from other cluster file systems by providing concurrent high-speed file access to applications executing on multiple nodes. With GPFS, a single file system containing several PBs of data can be built on dedicated hardware in a SAN configuration. In a large farm it is not feasible to connect each worker node in the GPFS cluster directly to the SAN via FC. For this reason GPFS, making use of a capability

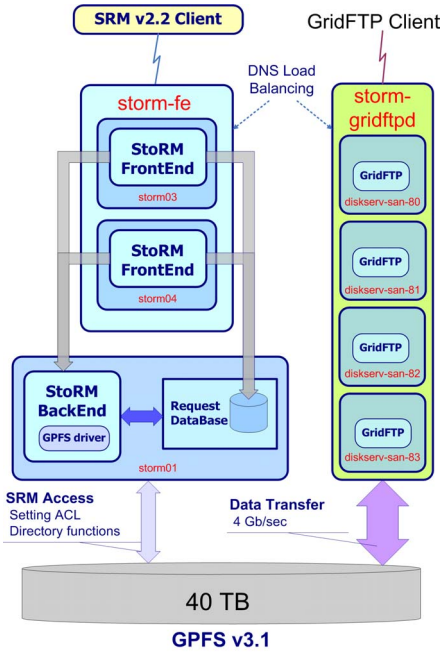

Figure 2. Sketch of the StoRM/GPFS test-bed layout.

called NSD (Network Shared Disk), provides a sort of software simulation of the SAN via Gigabit network.

The EMC system comprised two storage processors (with 4 GB of RAM each), connected to a Brocade 48000 Fibre Channel fabric director. The 4 disk-servers were equipped with dual Intel Xeon 1.6 GHz processors, 4 GB of RAM, dual-port Qlogic 246x Host Bus Adapter (HBA) and 1 Gigabit Ethernet link. Hence the total theoretical bandwidth available was 4 Gb/s. The disk-servers also acted as GridFTP [7] front-ends, i.e. besides the GPFS services they run also the GridFTP daemon. As Operating System (OS) we used a Scientific Linux CERN (SLC) [14] version 4.4, with a 2.6.9-42.0.8.EL.cernsmp kernel operating at 64 bits.

The StoRM service was provided by three machines: two machines running the StoRM front-end, balanced by means of a dynamic DNS, and one machine running the StoRM back-end and the MySQL database engine. The two front-end servers were equipped with dual AMD Opteron 2.2 GHz processors, 4 GB of RAM and Gigabit NIC. The back-end machine was a dual Intel Xeon 2.4 GHz, with 2 GB of RAM and Gigabit NIC.

A sketch of the test-bed layout is given in Fig. 2.

5 Tests and results

5.1 Realistic data transfers and analysis

Tests to understand how StoRM behaves under real operative conditions have been carried out. Aim of these tests

was proving whether StoRM copes with the use cases provided by a HEP experiment like LHCb [11]. According to the LHCb Computing Model [21] the disk-only based storage (T0D1) Service Class at each Tier-1 centre is used for data analysis and raw data reprocessing. The aggregated LHCb data flow from all the other Tier-1 centres and the CERN Tier-0 centre to the CNAF disk SE is estimated to be about 30 MB/s of steady traffic. However spikes on the consumed bandwidth should be also envisaged, as previous LHCb *Data Challenge* activities have proven so far.

The first test consisted of "real" data transfers from all the existing SRM version 1 LHCb production end-points to our StoRM end-point, by means of tools currently used in the daily operation within the LHCb Collaboration. The test has been running during 14 consecutive hours, with a steady throughput of about 150 MB/s without problems. The failure rate due to StoRM itself (TURL not returned, SRM requests not fully honored) was completely negligible, being the StoRM service not really overloaded for such a use case. The rate of problems in copying data via the 4 GridFTP daemons composing the StoRM instance was as well insignificant. A failure rate from 1% to 5% (depending on the source) has globally been observed. This is mainly due to timeouts in retrieving the TURL at the source.

A second test has been conceived to access data by means of a realistic LHCb analysis application, namely DaVinci [3], which internally makes use of ROOT [13] to access the data files. This test consisted in running simultaneously a certain number of analysis applications each one accessing several files stored in StoRM.

While this test might comfortably reflect the real situation of a Tier-1 as soon as the number of concurrent jobs gets close to the number of CPU slots currently available at CNAF, unfortunately we could not reserve for our tests the entire farm, since heavy production activities were going on at the same time, by LHCb and other VOs hosted at CNAF. However, it was possible to run about 200 DaVinci analysis jobs simultaneously, each one accessing 10 input files from the StoRM system. A few failures were observed due to misconfigurations of the used Worker Nodes, but no failures in the data access operations nor in issuing the SRM requests occurred for the rest of the correctly configured Worker Nodes. Despite the limited number of available CPU slots for this test, the successful simultaneous access to order of 2000 files through StoRM from 200 nodes is a promising result for the feasibility of a real data analysis activity. The same kind of test, but for a 24 hours period, is foreseen in the ongoing SRM testing activity of LHCb.

5.2 Throughput tests

A specific data transfer test has been performed to understand how StoRM reacts in heavy-duty operation conditions and to probe the limits of this particular instance. The core of the test consisted of client scripts which used low level tools (*globus-url-copy* [5]) for transferring data from LHCb dedicated disk pools at CERN (different source TURLs corresponding to real LHCb data produced by simulations) to always different destination files in StoRM. In this test we had not the possibility to control possible bottlenecks introduced by the source disk-servers at CERN, since irregular activities by LHCb were potentially running in parallel on the same sources. At the destination side, for each incoming file, the script issued a *srmPrepareToPut* SRM command, a polling requesting the status to the SRM until the TURL was returned, then it performed the data transfer and, once the transfer was complete, it issued a *srmPutStatusDone* SRM request to StoRM. This means that the full transfer chain was *de facto* emulated at the destination side and the usage of TURLs at the source could not alter the study of the behavior of the destination, i.e. StoRM.

The preliminary part of the test was dedicated to look for the best combination of the number of parallel streams for each data transfer and the number of concurrent data transfers, in order to optimize the throughput. The maximum throughput was reached with 15 parallel streams per transfer and 120 concurrent data transfers. The size of the files was O(100) MB. With this configuration we run a 14 hours test observing a sustained rate slightly exceeding 350 MB/s for about 8 hours, with a peak at 370 MB/s.

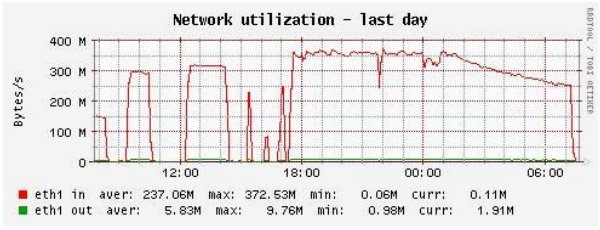

Figure 3. Aggregated network throughput measured during the data transfer test.

Fig. 3 shows the aggregated network throughput versus time during the test. The bandwidth started degrading after 6 hours because the workstation at CERN used to run the client script got overloaded and the available memory was exhausted. In 14 hours about 17 TB of data were moved from CERN to CNAF, interacting with the StoRM SRM interface about 400k times, with more than 100k files transferred. The GPFS disk-servers began suffering this large traffic: non negligible I/O Wait CPU load was observed on the disk-servers during the tests (but in parallel to our tests some additional activities from other VOs were using the same disk back-end storage).

Since the GridFTP servers were running on the 4 GPFS disk-servers, each one provided with a Gigabit network in-

terface, the total available network bandwidth was about 500 MB/s. The link from CERN to CNAF was not at all a limiting factor, since it actually had a bandwidth of 10 Gb/s. In conclusion we have been able to consume about 70% of the theoretical network bandwidth.

The observed percentage of failures has been very low, about 0.3% due to the file transfers themselves (i.e. non-zero error code got from *globus-url-copy*), and 0.1% due to StoRM. The failures during the data transfers can have several possible explanations, e.g. problems due to high load on the source disk-servers or temporary network glitches. The few failures due StoRM were basically due to timeouts and will be discussed in the following section.

5.3 StoRM stress tests

A series of stress tests without data transfers, i.e. only measuring the pure SRM performances, were also put in place to understand how the implementation of StoRM responds when critical conditions are reached. These tests provided an evaluation of the performance of this StoRM implementation and recipes for scaling up installations which would require larger performances.

It is worth noting that the results of these tests do not depend on the performance of the underlying GPFS file system, hence all the latencies we will mention have to be addressed to the StoRM instance itself. The test turned into an invaluable opportunity for tuning and knowing the behavior of StoRM in a tight collaboration with its developers, and led to a series of optimizations, in particular to the introduction of several indices into the StoRM database for improving the response of the relevant SQL queries.

A first step targeted to find the limits of the system, by gradually increasing the load on the SRM endpoint and studying the response as a function of the number of concurrent processes. Each process issues a sequence of SRM requests to list the content of a directory (*srmLs*), to remove the files in it (*srmRm*), and then to allocate space for a new file (*srmPrepareToPut*). The plot in Fig. 4 (upper part) shows how the average time for retrieving a TURL from StoRM varies as function of the number of concurrent processes. The average time was computed as the mean value of all processes mean times and the error bars represent a systematic uncertainty computed by repeating the test several times. Fig. 4 also shows (bottom part) the percentage of failures due to StoRM, which is negligible up to 500 concurrent requests, while it increases considerably from 600 parallel processes onward. This behavior suggests that the system can safely handle up to $\sim$ 500 parallel requests, corresponding to a rate of SRM client commands of about 80 Hz[5], whereas for $\sim$ 600 parallel processes it effectively enters into a critical regime. This was basically due to the pretty high load which was reached on the StoRM front-end service nodes, which could no longer process in due time the requests, resulting in hard timeouts with an error message from gSOAP: "CGSI-gSOAP: could not open connection! TCP connect failed in tcp_connect()". In fact, it indicates that the StoRM front-end, which is in charge of accepting and authenticating the incoming requests through the gSOAP package, was not able within a given timeout to serve the SRM request due to the large load. Of course in such cases the client code could perform one or more retries, thus reducing the effective rate of failures. The obvious solution to this kind of problems would be to scale up the system in order to cope with a rate of requests higher than the expected use case. However, such a predictable failure condition should be taken into account with an appropriate admission control queue in StoRM, in order to reject the requests that cannot be fulfilled under heavy load conditions. The StoRM developers are already working to implement a configurable timeout by-passing the gSOAP one, and to return an ordinary SRM message which states that the ap-

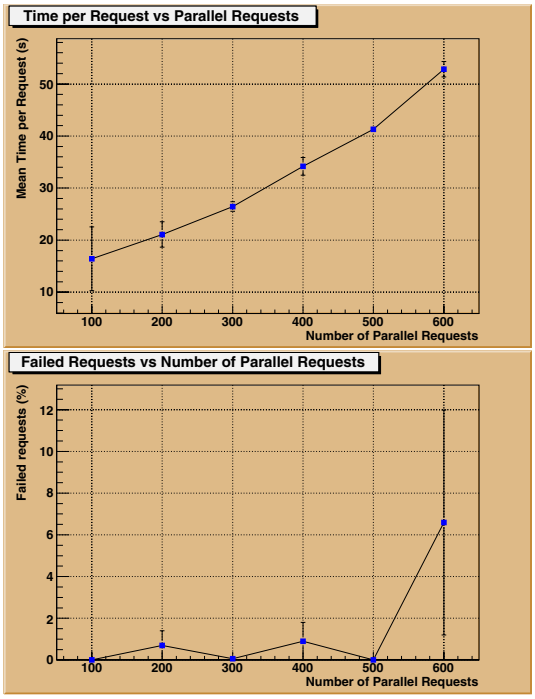

Figure 4. Upper plot: mean time required to fulfill a *srmPrepareToPut* request versus the number of parallel client processes. Lower plot: percentage of failed requests versus number of parallel client processes.

[5]Each *prepare to put* sequence implies on average 6.5 client commands (1 *srmPrepareToPut* + 5.5 *srmStatusPtP*), thus the rate can be roughly computed as $(6.5 \times 500\, requests)/40s \simeq 80Hz$. This is an upper limit of the frequency, since the 500 processes are not always active all at the same time.

plication rejected the request due to the high load.

The second part of the test was aimed to investigate the failures of each SRM functionality individually. It consisted in running different requests (*srmLs*, *srmMkdir*, *srmRmdir*, *srmRm*, *srmPrepareToPut*, *srmPrepareToGet*) while the system was kept in a critical status by a parallel stress test launched from different client machines. By splitting the test over multiple client nodes we avoided the results of the test to be affected by the high load of one single client machine. In Tab. 1 we summarize the percentage of failures found for this particular setup[6]. Synchronous operations like *srmLs*, *srmMkdir*, *srmRmdir*, and *srmRm* suffered the load of the service more than the asynchronous ones like *srmPrepareToPut*.

Functionality	Mean time per request (s)	Rate of failed requests (%)
srmMkdir	70 ± 20	6 ± 3
srmLs	30 ± 3	4 ± 2
srmRmdir	30 ± 3	4 ± 1
srmRm	30 ± 2	3 ± 1
srmPrepareToPut	85 ± 15	0.5 ± 0.5
srmPrepareToGet	57 ± 7	1 ± 1

Table 1. Results of dedicated stress tests on different SRM functionalities. In the second column the mean time to execute the command is reported, while in the third column the average rate of request failures. The errors are estimations of the systematic uncertainties obtained running the same test under the same conditions several times. These measurements were done while running in parallel other tests in order to keep the system under very high load. See the text for the proper interpretation of these numbers.

During the first phase of the tests we observed also another occasional error message due to failed XML-RPC executions. Thanks to our tests this problem has been discovered and fixed by the developers, by increasing the number of XML-RPC channels accordingly with the number of threads configured in the StoRM front-end.

In order to compare these results with a real use case, a rough estimation of the needs of LHCb has been done on the basis of the LHCb Computing Model [21]. The expected rate of calls to the SRM interface in a Tier-1 site during the first year of LHC data taking is estimated to be of order of 1 Hz. This computation includes the data transfer from CERN to the Tier-1s during the data taking, the data exchange amongst the Tier-1s and from the Tier-2s as well as data analysis activities going on at the Tier-1. The results reported above prove that the instance of StoRM used for these tests can comfortably cope with the needs of a HEP experiment such as LHCb.

[6]These numbers strongly depend on both the hardware setup of the instance and on the overall load of the system that we have appropriately put in place. Hence, they are not intended as a general estimation of the StoRM performance, but rather as a study of the response of the system to individual SRM operations while it is kept highly loaded.

5.4 File removal

Another interesting measurement consisted in deleting a large amount of data by issuing a series of *srmRmdir* requests. The effectiveness of this functionality is extremely important for the LHC VOs and production managers, since it would allow to handle a remote storage for freeing wasted space or removing old data without the need to interact directly with the system administrator of the remote site, hence speeding up considerably the whole process.

For these test we have spread 17 TB of data over 50 directories, and then simply issued 50 recursive *srmRmdir* requests for deleting the entire data-set. Fig. 5 shows the GPFS file system occupancy versus time. As it can be seen in the plot, about 1300 seconds were required to complete the removal of these 17 TB of data.

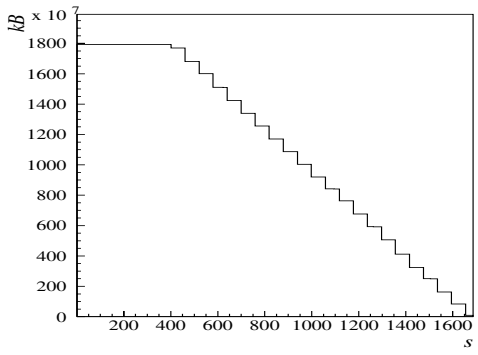

Figure 5. Occupancy of the GPFS file system as a function of time during the file removal test. The test started at about t=400 s, and was over at t=1700 s. The step behavior clearly visible in the plot is due to the time that GPFS needs to update its metadata information to the clients, and roughly corresponds to a few tens of seconds.

6 Conclusions

The main concepts of the StoRM implementation of SRM have been presented. An instance of StoRM has been deployed at the CNAF Tier-1 facility for test and optimization purposes. This StoRM instance has undergone a series of tests aiming to measure the throughput in data transfers, the capability in file access for analysis activity, to study the behavior of the system under high load, as well as to optimize the StoRM implementation itself.

The results show that the instance of StoRM has been able to sustain for several hours an incoming throughput over the WAN of about 360 MB/s during a period of 14 hours with a negligible rate of failed transfers. Then, the tests performed to access the data through the LHCb analysis software have given a promising result for the feasibility of data analysis activities in a realistic scenario, though the number of simultaneous jobs was limited by the available CPU slots at CNAF. Finally, the study of the system in heavy-duty run conditions showed that such an instance of StoRM can sustain a load of 500 parallel processes providing a total rate of about 80 Hz of SRM interactions.

Our results show that StoRM can comfortably fulfill the requirements of a HEP experiment like LHCb as a SRM pure disk implementation.

Acknowledgements

We kindly acknowledge the INFN-CNAF staff for the prompt support. We also wish to express our thanks to the LHCb computing group and to the LCG GSSD group for their interest and support to this work. The StoRM project was born as a collaboration between INFN, funded by the GRID.IT Project, and ICTP, funded by the EGRID Project. We warmly acknowledge our colleagues from ICTP. At the moment, the support, the maintenance and the development of StoRM are made in the framework of EGEE-SA1.

References

[1] BeStMan: Berkeley Storage Manager. http://datagrid.lbl.gov/bestman, July 2007.

[2] CASTOR: CERN Advanced STORage manager. http://castor.web.cern.ch/castor, July 2007.

[3] DaVinci: The LHCb Analysis Program . http://cern.ch/LHCb-release-area/DOC/davinci, July 2007.

[4] DPM: LCG Disk Pool Manager. https://twiki.cern.ch/twiki/bin/view/LCG/DpmAdminGuide, July 2007.

[5] The globus-url-copy tool. http://www.globus.org/grid_software/data/globus-url-copy.php, July 2007.

[6] GPFS: General Parallel File-System. http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.gpfs.doc/gpfsbooks.html, July 2007.

[7] GridFTP. http://www.globus.org/grid_software/data/gridftp.php, July 2007.

[8] HPSS: High Performance Storage System. http://www.hpss-collaboration.org, July 2007.

[9] LFC: LCG File Catalog. https://twiki.cern.ch/twiki/bin/view/LCG/LfcAdminGuide, July 2007.

[10] LHC: Large Hadron Collider. http://lhc.web.cern.ch/lhc, July 2007.

[11] LHCb Collaboration. http://lhcb.cern.ch, July 2007.

[12] Lustre. http://www.lustre.org, July 2007.

[13] The ROOT system. http://root.cern.ch, July 2007.

[14] Scientific Linux CERN. http://linux.web.cern.ch/linux, July 2007.

[15] SRM Working Group. http://sdm.lbl.gov/srm-wg, July 2007.

[16] The Storage Resource Manager Interface Specification, Version 2.2. http://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.html, April 2007.

[17] StoRM: Storage Resource Manager. http://storm.forge.cnaf.infn.it, July 2007.

[18] XFS file-system. http://oss.sgi.com/projects/xfs, July 2007.

[19] EGRID project. http://www.egrid.it/, Jan 2006.

[20] Lustre: A Scalable HighPerformance File System. Technical report, Cluster File Systems, Inc., Jan 2002.

[21] LHCb Computing TDR. Technical Report CERN-LHCC-2005-019, 2005.

[22] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, Á. Frohner, K. Lörentey, and F. Spataro. From gridmap-file to VOMS: managing authorization in a Grid environment. *Future Generation Comp. Syst.*, 21(4):549–558, 2005.

[23] A. Caltroni, V. Ciaschini, A. Ferraro, A. Ghiselli, G. Rubini, and R. Zappi. G-PBox: A Policy Framework for Grid Environments. In *Proceedings of the International CHEP 2004*, Interlaken, Switzerland, 2004.

[24] E. Corso, S. Cozzini, A. Forti, A. Ghiselli, L. Magnoni, A. Messina, A. Nobile, A. Terpin, V. Vagnoni, and R. Zappi. StoRM: A SRM Solution on Disk Based Storage System. In *Proceedings of the Cracow Grid Workshop 2006 (CGW2006)*, Cracow, Poland, October 15-18, 2006.

[25] M. De Riese, P. Fuhrmann, T. Mkrtchyan, M. Ernst, A. Kulyavtsev, V. Podstavkov, M. Radicke, M. Sharma, M. Litvintsev, and T. Perelmutov. dCache, the Book. http://www.dcache.org/manuals/Book/, Nov 2006.

[26] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System For Large Computing Clusters. *FAST'02*, pages 28–30, January 2002.

[27] A. Shoshani, A. Sim, and J. Gu. Storage Resource Managers: Essential Components for the Grid. In J. w. Edited by Jarek Nabrzyski, Jennifer M. Schopf, editor, *Grid Resource Management: State of the Art and Future Trends*, number 1-4020-7575-8, chapter 20, pages 321–340. Kluwer Academic Publishers, Norwell, MA, USA, 2004.