# Storage Resource Managers:
# Recent International Experience on Requirements and Multiple Co-Operating Implementations

Lana Abadie[1] , Paolo Badino[1] , Jean-Philippe Baud[1] ,Ezio Corso[2], Matt Crawford[3], Shaun De Witt[4], Flavia Donno[1] , Alberto Forti[5], Ákos Frohner[1], Patrick Fuhrmann[6], Gilbert Grosdidier[7], Junmin Gu[8], Jens Jensen[4], Birger Koblitz[1] , Sophie Lemaitre[1], Maarten Litmaath[1], Dmitry Litvinsev[3], Giuseppe Lo Presti[1], Luca Magnoni[5], Tigran Mkrtchan[6], Alexander Moibenko[3], Rémi Mollon[1], Vijaya Natarajan[8], Gene Oleynik[3], Timur Perelmutov[3], Don Petravick[3], Arie Shoshani[8], Alex Sim[8], David Smith[1], Massimo Sponza[2], Paolo Tedesco[1], Riccardo Zappi[5].
Editor and coordinator: Arie Shoshani[8]

*[1] CERN, European Organization for Nuclear Research, Switzerland;*
*[2] ICTP/EGRID, Italy;*
*[3] Fermi National Accelerator Laboratory, Batavia, Illinois, USA;*
*[4] Rutherford Appleton Laboratory, Oxfordshire, England;*
*[5] INFN/CNAF, Italy;*
*[6] Deutsches Elektronen-Synchrotron, DESY, Hamburg, Germany;*
*[7] LAL / IN2P3 / CNRS, Faculté des Sciences, Orsay Cedex, France;*
*[8] Lawrence Berkeley National Laboratory, Berkeley, California, USA.*

## Abstract

*Storage management is one of the most important enabling technologies for large-scale scientific investigations. Having to deal with multiple heterogeneous storage and file systems is one of the major bottlenecks in managing, replicating, and accessing files in distributed environments. Storage Resource Managers (SRMs), named after their web services control protocol, provide the technology needed to manage the rapidly growing distributed data volumes, as a result of faster and larger computational facilities. SRMs are Grid storage services providing interfaces to storage resources, as well as advanced functionality such as dynamic space allocation and file management on shared storage systems. They call on transport services to bring files into their space transparently and provide effective sharing of files. SRMs are based on a common specification that emerged over time and evolved into an international collaboration. This approach of an open specification that can be used by various institutions to adapt to their own storage systems has proven to be a remarkable success – the challenge has been to provide a consistent homogeneous interface to the Grid, while allowing sites to have diverse infrastructures. In particular, supporting optional features while preserving interoperability is one of the main challenges we describe in this paper. We also describe using SRM in a large international High Energy Physics collaboration, called WLCG, to prepare to handle the large volume of data expected when the Large Hadron Collider (LHC) goes online at CERN. This intense collaboration led to refinements and additional functionality in the SRM specification, and the development of multiple interoperating implementations of SRM for various complex multi-component storage systems.*

## 1. Introduction and Overview

Increases in computational power have created the opportunity for new, more precise and complex scientific simulations leading to new scientific insights. Similarly, large experiments generate ever increasing volumes of data. At the data generation phase, large volumes of storage have to be allocated for data collection and archiving. At the data analysis phase, storage needs to be allocated to bring a subset of the data for exploration, and to store the subsequently generated data products. Furthermore, storage systems shared by a community of scientists need a common data access mechanism which allocates storage space dynamically, manages stored content, and automatically remove unused data to avoid clogging data stores.

When dealing with storage, the main problems facing the scientist today are the need to interact with a variety of storage systems and to pre-allocate storage to ensure data generation and analysis tasks can take place. Typically, each storage system provides different interfaces and security mechanisms. There is an urgent need to standardize and streamline the access interface, the dynamic storage allocation and the management of the content of these systems. The goal is to present to the scientists the same interface regardless of the type of system being used. Ideally, the management of storage allocation should become transparent to the scientist.

24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007)
0-7695-3025-7/07 $25.00 © 2007 IEEE

To accommodate this need, the concept of Storage Resource Managers (SRMs) was devised [15, 16] in the context of a project that involved High Energy Physics (HEP) and Nuclear Physics (NP). SRM is a specific set of web services protocols used to control storage systems from the Grid, and should not be confused with the more general concept of Storage Resource Management as used in industry. By extension, a Grid component providing an SRM interface is usually called "an SRM."

After recognizing the value of this concept as a way to interact with multiple storage systems in a uniform way, several Department of Energy Laboratories (LBNL, Fermilab, and TJNAF), as well as CERN and Rutherford Appleton Lab in Europe, joined forces and formed a collaboration that evolved into a stable version, called SRM v1.1, that they all adopted. This led to the development of SRMs for several disk-based systems and mass storage systems, including HPSS (at LBNL), Castor (at CERN), Enstore (at Fermilab), and JasMINE (at TJNAF). The interoperation of these implementations was demonstrated and proved an attractive concept. However, the functionality of SRM v1.1 was limited, since space was allocated by default policies, and there was no support for directory structures. The collaboration is open to any institution willing and able to contribute. For example, when INFN, the Italian institute for nuclear physics, started working on their own SRM implementation (StoRM, described below), they joined the collaboration. The collaboration also has an official standards body, the Open Grid Forum, OGF, where it is registered as GSM-WG (GSM is Grid Storage Management; SRM was already taken for a different purpose).

Subsequent collaboration efforts led to advanced features such as explicit space reservations, directory management, and support for Access Control Lists (ACL) to be supported by the SRM protocol, now at version 2.1. As with many advanced features, it was optional for the implementations to support them, partly to be inclusive: we did not want to exclude implementations without specific features from supporting version 2.1. This inclusiveness principle is a foundation for the SRM collaboration, but is a source of problems in writing applications and in testing interoperability, as we shall see below.

Later, when a large international HEP collaboration, WLCG (the World-wide LHC Computing Grid) decided to adopt the SRM standard, it became clear that many concepts needed clarification, and new functionality was added, resulting in SRM v2.2. While the WLCG contribution has been substantial, the SRM can also be used by other Grids, such as those using the EGEE gLite software. There are many such Grids, often collaborations between the EU and developing countries. Having open source and license-free implementations (as most of the implementations described in this paper are) helps these projects.

In this paper, we elaborate on the process of the definition of the SRM v2.2 protocol and its interface to a variety of storage systems. Furthermore, we establish a methodology for the validation of the protocol and its implementations through families of test suites. Such test suites are used on a daily basis to ensure inter-operation of these implementations. This joint international effort proved to be a remarkable and unique achievement, in that now there are multiple SRMs developed in various institutions around the world that interoperate. Many of these SRMs have a large number of installations around the world. This demonstrates the value of inter-operating middleware over a variety of storage systems.

In section 2, we describe related work. In Section 3 and 4 we concentrate on the basic functionality exposed by SRM and the concepts that evolved from this international collaboration. Section 5 focuses on five inter-operating SRM v2.2 implementations over widely different storage systems, including multi-component and mass storage systems. Section 6 describes the validation process, and presents the results of interoperation tests and lessons learned from such tests.

## 2. Related Work

The Storage Resource Broker (SRB) [11] is a client-server middleware that provides uniform access for connecting to heterogeneous data resources over a wide-area network and accessing replicated data sets. It uses a centralized Meta Data Catalog (MCat) and supports archiving, caching, synchs and backups, third-party copy and move, version control, locking, pinning, aggregated data movement and a Global Name space (filesystem like browsing). SRB provides as well for collection and data abstraction presenting a Web Service interface. While SRB offers a complete storage service, in comparison, SRM is only the interface to storage; it is an open (in particular, non-proprietary) web service protocol, allowing storage systems to fit in as components into a larger data and computational Grid. Consequently, SRMs can have independent implementations on top of various storage systems, including multi-disk caches, parallel files systems, and mass storage systems.

Condor [4] from University of Wisconsin at Madison is a comprehensive middleware suite, supporting storage natively via the Chirp protocol. Chirp is a remote I/O protocol that provides the equivalent of UNIX operations such as open(), read(), write(), close(). Chirp provides a variety of authentication methods, allowing remote users to identify themselves with strong Globus or Kerberos credentials. However, it does not offer space management capabilities, such as those available in SRM. The Chirp protocol is also used by the NeST component that aims to deliver guaranteed allocations, one of the optional features of SRM. However, NeST currently relies primarily on an underlying file system to provide access to storage. The Condor storage middleware suite presents some overlap with SRM in terms of features and intent. However, generally speaking the SRM protocol is designed mainly for managing storage spaces and their content and Chirp is focused on data access.

There is some interest in interoperability between SRB and SRM, or between SRM and Condor. However, such efforts did not come to fruition since the effort required to do that properly outweighs the need, particularly since the implementations fit into Grids at different levels of the software stack.

Other computational Grids use distributed file systems. A protocol that is gaining in popularity is NFSv4. It is the IETF standard for distributed file systems that is designed for security, extensibility, and high performance. The NFSv4 offers a global name space and provides a pseudo file system that enables support for replication, migration and referral of data. One of the attractive features of NFS4 is the decoupling of the data paths from the storage access protocol. In particular, the possibility of negotiating a storage access and management protocol between data servers would allow for SRM to play a role in the integration of mass storage systems in an NFSv4 infrastructure.

## 3. The Basic Concepts

The ideal vision of a distributed system is to have middleware facilities that give clients the illusion that all the compute and storage resources needed for their jobs are running on their local system. This implies that a client only logs in and gets authenticated once, and that some middleware software figures out where are the most efficient locations to move data to, to run the job, and to store the results in. The middleware software plans the execution, reserves compute and storage resources, executes the request, and monitors the progress. The traditional emphasis is on sharing large compute resource facilities, sending jobs to be executed at remote computational sites. However, very large jobs are often "data intensive", and in such cases it may be necessary to move the job to where the data sites are in order to achieve better efficiency. Alternatively, partial replication of the data can be performed ahead of time to sites where the computation will take place. Thus, it is necessary to also support applications that produce and consume large volumes of data. In reality, most large jobs in the scientific domain involve the generation of large datasets, the consumption of large datasets, or both. Therefore, it is essential that software systems exist that can provide space reservation and schedule the execution of large file transfer requests into the reserved spaces. Storage Resource Managers (SRMs) are designed to fill this gap.

In addition to storage resources, SRMs also need to be concerned with the *data resource* (or files that hold the data). A data resource is a chunk of data that can be shared by more than one client. In many applications, the granularity of a data resource is a file. It is typical in such applications that tens to hundreds of clients are interested in the same subset of files when they perform data analysis. Thus, the management of shared files on a shared storage resource is also an important aspect of SRMs. The decision of which files to keep in the storage resource is dependent on the cost of bringing files from remote systems, the size of the file, and the usage level of that file. The role of the SRM is to manage the space under its control in a way that is most cost beneficial to the community of clients it serves.

In general, an SRM can be defined as a middleware component that manages the dynamic use and content of a storage resource in a distributed system. This means that space can be allocated dynamically to a client, and that the decision of which files to keep in the storage space is controlled dynamically by the SRM. The main concepts of SRMs are described in [15] and subsequently in more detail in a book chapter [16]. The concept of a storage resource is flexible: an SRM could be managing a disk cache, or a hierarchical tape archiving system, or a combination of these. In what follows, they are referred to as "storage components". When an SRM at a site manages multiple storage resources, it may have the flexibility to store each file at any of the physical storage systems it manages or even to replicate the files in several storage components at that site. The SRMs do not perform file transfer, but rather cooperate with file transfer services, such as GridFTP, to get files in/out of their storage systems. Some SRMs also provide access to their files through Posix or similar interfaces. Figure 1 shows a schematic diagram of the SRM concepts as well as the storage systems and institutions that developed them for v2.2, described in this paper.

SRMs are designed to provide the following main capabilities:

1) *Non-interference with local policies*. Each storage resource can be managed independently of other storage resources. Thus, each site can have its own policy on which files to keep in its storage resource and for how long. The SRM will not interfere with the enforcement of local policies. Resource monitoring of both space usage and file sharing is

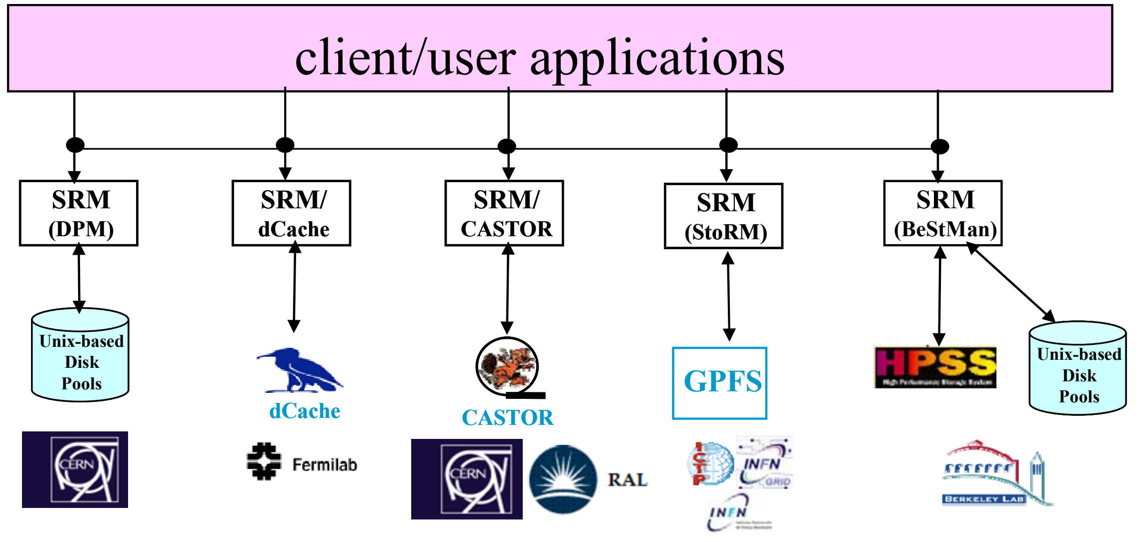

Figure 1: Multiple inter-operating SRM implementations. Clients can access different mass storage and file systems through a uniform SRM interface

needed in order to profile the effectiveness of the local policies.

2) *Pinning files.* Files residing in one storage system can be temporarily locked in place before being removed for resource usage optimization or transferred to another system that needs them, while used by an application. We refer to this capability as *pinning* a file, since a pin is a lock with a *lifetime* associated with it. A pinned file can be actively *released* by a client, in which case the space occupied by the file is made available to the client. SRMs can choose to keep or remove a released file depending on their storage management needs.

3) *Advance space reservations.* SRMs are components that manage the storage content dynamically. Therefore, they can be used to plan the storage system usage by permitting advance space reservations by clients.

4) *Dynamic space management.* Managing shared disk space usage dynamically is essential in order to avoid clogging of storage resources. SRMs use file replacement policies whose goal is to optimize service and space usage based on access patterns.

5) *Support abstract concept of a file name.* SRMs provide an abstraction of the file namespace using "Site URLs" (SURLs), while the files can reside in any one or more of the underlying storage components. An example of an SURL is: srm://ibm.cnaf.infn.it:8444/dteam/test.10193, where the first part "ibm.cnaf.infn.it:8444" is the address and port of the machine where the SRM resides, and the second part "dteam/test.10193" is the abstract file path, referred to as the Site File Name (SFN). Multiple replicas of a file in different sites will have different SURLs, which can be published in replica catalogs. When clients wish to get a file based on its logical file name, they need to consult a replica catalog to determine where to get a replica from (e.g. nearest site). Such global decisions are purposefully not provided by SRMs, since they only provide local services.

6) *Temporary assignment of transfer file names.* When requesting a file from an SRM, an SURL (see above) is provided. The SRM can have the file in several locations, or can bring it from tape to disk for access. Once this is done a "Transfer URL" (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 file into the SRM. The request provides the desired SURL for the file, and the SRM returns a TURL for the transfer of the file into the SRM. A TURL must have a valid transfer protocol such as: gsiftp://ibm139.cnaf.infn.it:2811//gpfs/dteam/test.10193. Note that the port 2811 is a GridFTP port.

7) *Directory Management and ACLs.* The advantage of organizing files into directories is well known, of course. However, SRMs provide directory management support to the SURL abstractions and

keep the mapping to the actual files stored in the underlying file systems. Accordingly, Access Control Lists (ACLs) are associated with the SURLs.

8) *Transfer protocol negotiation.* When making a request to an SRM, the client needs to end up with a protocol for the transfer of the files that the storage system supports. In general, systems may be able to support multiple protocols and clients should be able to use different protocols depending on the system they are running on. SRM supports protocol negotiation, by matching the highest protocol they can support given an ordered list of preferred protocols by the client.

9) *Peer to peer request support.* In addition to responding to clients requests, SRMs are designed to communicate with each other. Thus, one SRM can be asked to copy files from/to another SRM.

10) *Support for multi-file requests*. The ability to make a single request to get, put, or copy multiple files is essential for practical reasons. This requirement is supported by SRMs by specifying a set of files. Consequently, such requests are asynchronous, and status functions need to be provided to find out the progress of the requests.

11) *Support abort, suspend, and resume operations*. These are necessary because requests may be running for a long time, in case that a large number of files are involved.

The main challenges for a common interface specification are to design the functionality of SRMs and their interfaces to achieve the goals stated above, and to achieve the interoperation of SRM implementations that adhere to the common interface specification. More details of the basic functionality can be found in [16]. The specification of SRM interfaces and their corresponding WSDL can be found at the collaboration web site [13].

The functions supported by SRMs in order to get or put files into the SRMs are referred to as “srmPrepareToGet” and “srmPrepareToPut”. A set of files (or a directory) is provided in the form of SURLs, and TURLs are returned. The TURLs are used by the requesting clients to get or put files from/into the SRM using the TURL’s transfer protocol. The function srnCopy provides the capability to replicate files from one SRM to another.

When using the space reservation function srmReserveSpace, the client can specify the desired space and duration of the reservation. The SRM returns the space and duration it is willing to allocate according to its policies, and a space token. If the client does not wish to accept that, it can issue srmReleaseSpace. Otherwise, it can put files into the reserved space by referring to the space token.

Directory functions are very similar to the familiar Unix functions and include srmLs, srmMkdir, srmRmdir, srmMv, and srmRm. Since files may have a limited lifetime in the SRM, these functions need to reflect lifetime status as well.

## 4. Additional concepts introduced with v2.2

Soon after the WLCG collaboration decided to try and adopt version 2.1 of the SRM specification as a standard for all their storage systems, it became clear that some concepts needed to be clarified, and perhaps new functionality added. The main issues were: 1) the specification of the storage properties; 2) the clarification of space and the meaning of a space token when it is returned after a space reservation is made; and 3) the ability to request that files will be brought from archival storage into an online disk system for subsequent access. This led to a new SRM specification, referred to as SRM v2.2. We discuss each of these concepts further next.

***Storage component properties***

The issue of how to expose expected behavior of a storage component by the SRM was debated at great length. In the end, it was concluded that it is sufficient to expose two orthogonal properties: Retention Policy and Access Latency. These are defined below:

1) **Retention Policy**: REPLICA, OUTPUT, CUSTODIAL

The Quality of Retention is a kind of Quality of Service. It refers to the probability that the storage system loses a file. The type is used to describe the retention policy assigned to the files in the storage system, at the moment when the files are written into the desired destination in the storage system. It is used as a property of space allocated through the space reservation function. Once the retention policy is assigned to a space, the files put in the reserved space will automatically be assigned the retention policy of the space. The description of Retention Policy Types is:

- REPLICA quality has the highest probability of loss, but is appropriate for data that can be replaced because other copies can be accessed in a timely fashion.
- OUTPUT quality is an intermediate level and refers to the data which can be replaced by lengthy or effort-full processes.
- CUSTODIAL quality provides low probability of loss.

2) **Access Latency**: ONLINE, NEARLINE

Files may be Online or Nearline. These terms are used to describe how the latency to access a file is improvable. Latency is improved by storage systems replicating a file such that its access latency is online. We do not include here “offline” access latency, since a human has to be involved in getting offline storage mounted. For SRMs, one can only specify ONLINE and NEARLINE. The type will be used to describe an access latency property that can be requested at the time of space reservation. The files that are contained in the space may have the same or lower access latency as the space. The ONLINE cache of a storage system is the part of the storage system which provides file access with online latencies. The description of Access Latency types is:

- ONLINE has the lowest latency possible. No further latency improvements are applied to online files.
- NEARLINE files can have their latency improved to online latency automatically by staging the files to online cache.

***Storage Areas and Storage Classes***

Because of fairly complex storage systems used by the WLCG collaboration, it was obvious that referring to “storage system” is imprecise. Instead, the concept of a “storage area” is used. A storage system usually is referred to as a Storage Element, viz. a grid element providing storage services.

A Storage Element can have one or more storage areas. Each storage area includes parts of one or more hardware components (single disk, RAID, tape, DVD, …). Any combination of components is permissible. A storage area is specified by its properties which include the Access Latency and Retention Policy described above. Explicitly supported combinations are known as Storage Classes: online-replica (e.g. a common disk space allocated for online access), nearline-custodial (e.g. a high-quality robotic tape system), or online-custodial (e.g. a highly protected online disk that may keep multiple replicas, or an online disk with backup on a high-quality robotic tape system). Storage areas that consist of heterogeneous components are referred to as “composite storage areas” and the storage space in them as “composite space”. “Composite storage elements” are storage elements serving composite storage areas. Storage areas can share one or more storage components. This allows storage components to be partitioned for use by different user-groups or Virtual Organizations (VOs).

The SRM interface exposes only the storage element as a whole and its storage areas, not their components. However, a space reservation to a composite storage element can be made requesting Access Latency-Retention Policy combinations that may determine which storage components are assigned. Specifically, a space reservation to a composite storage element can request the following combinations to target the online or nearline storage components: a) online-replica to target the online storage components; b) nearline-custodial to target the nearline storage components (assuming they support custodial retention policy); c) online-custodial to target both the online and nearline storage components.

***The function srmBringOnline***

When a file is requested from a mass storage system (MSS), it is brought onto disk from tape in case that the file is not already on disk. The system determines which files to keep on disk, depending on usage patterns and system loads. However, this behavior is not always acceptable to large projects, since they need to be in control of what is online in order to ensure efficient use of computing resources. A user performing a large analysis may need to have all the files online before starting the analysis. Similarly, a person in charge of a group of analysts may wish to bring all the files for that group online for all of them to share. Therefore the concept of bringing files online was introduced.

srmBringOnline can be applied only to a composite space that has nearline as well as online components. When performing this function the SRM is in full control as to where files end up and this information is not visible to the client. For example, the SRM may have multiple online spaces, and it can choose which will be used for each file of the request. Similarly, the SRM can choose to keep multiple online replicas of the same file for transfer efficiency purposes. Once srmBringOnline is performed, subsequent srmPrepareToGet requests can be issued by clients, and TURLs returned, where each TURL indicates where the corresponding file can be accessed, and the protocol to be used.

## 5. The Implementation of five SRMs

In this section we describe briefly implementations of five SRM that adhere to the same SRM v2.2 specification, in order to illustrate the ability of SRMs to have the same interface to a variety of storage systems. The underlying storage systems can vary from a simple disk, multiple disk pools, mass storage systems, parallel file systems, to complex multi-component multi-tiered storage systems. While the implementations use different approaches, we illustrate the power of the SRM standard approach in that such systems exhibit a uniform interface and can successfully interoperate. Short descriptions of the SRMs implementation are presented (in alphabetical order) next.

### *BeStMan – Berkeley Storage Manager*

BeStMan is a java-based SRM implementation from LBNL. Its modular design allows different types of storage systems to be integrated in BeStMan while providing the same interface for the clients. Based on immediate needs, two particular storage systems are currently used. One supports multiple disks accessible from the BeStMan server, and the other is the HPSS storage system. Another storage system that was adapted with BeStMan is a legacy MSS at NCAR in support of the Earth System Grid project (www.earthsystemgrid.org).

Figure 2 shows the design of BeStMan. The Request Queue Management accepts the incoming requests. The Local Policy Module contains the scheduling policy, garbage collection policy, etc. The Network Access Management module is responsible for accessing files using multiple transfer protocols. An in-memory database is provided for storing the activities of the server. The Request Processing module contacts the policy module to get the next request to work on. For each file request, the necessary components of the Network Access Management module and the Storage Modules (the Disk Management and the MSS Access Management modules) are invoked to process the data.

BeStMan supports space management functions and data movement functions. Users can reserve space in the preferred storage system, and move files in and out of their space. When necessary BeStMan interacts with remote storage sites on their behalf, e.g. another gsiftp server, or another SRM. BeStMan is expected to replace all currently deployed v1.1 SRMs from LBNL.

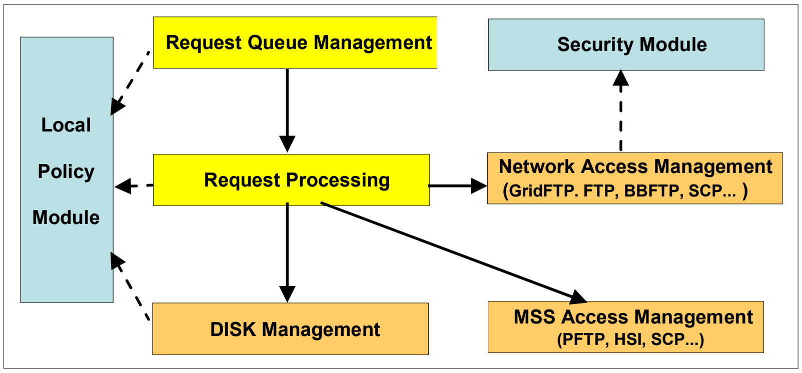

Figure 2: The architecture diagram of BeStMan

### *Castor-SRM*

The SRM implementation for the CERN Advanced Storage system (CASTOR) is the result of collaboration between Rutherford Appleton Laboratory and CERN. Like that of other implementations, the implementation faced unique challenges. These challenges were based around the fundamental design concepts under which CASTOR operates, which are different from those of other mass storage systems. CASTOR trades some flexibility for performance, and this required the SRM implementation to have some loss of flexibility, but with gains in performance.

CASTOR is designed to work with a tape back-end and is required to optimise data transfer to tape, and also to ensure that data input to front-end disk cache is as efficient as possible. It is designed to be used in cases where it is essential to accept data at the fastest possible rate and have that data securely archived. These requirements are what cause differences between the CASTOR SRM implementation and others.

The need to efficiently stream to tape and clear disk cache for new incoming data leads to two effects:

- the SURL lifetime is effectively infinite and
- the TURL, or pinning, lifetime is advisory.

In fact the latter is merely a modified garbage collection algorithm which tries to ensure those files with a low weighting are garbage collected first.

Also, space management in the CASTOR SRM is significantly different to those of other implementations. Since the design of the MSS is to optimise moving data from disk to tape, there is no provision for allowing dynamic space allocation at a user level. The CASTOR SRM does support space reservation, but as an asynchronous process involving physical reallocation of the underlying disk servers. Other implementation designed to work with only disk based Mass Storage Systems, or a combination of disk and tape, often allow for dynamic space reservation.

The architecture of the CASTOR SRM, shown in Figure 3, includes two stateless processes, which interact through a RDBMS. A client-facing process (the 'server') directly deals with synchronous requests and stores asynchronous requests in the database for later processing. The database is therefore used to store all storage-oriented requests as well as the status of the entire system. A separate process (the 'daemon') faces the CASTOR backend system, and updates the status of the ongoing requests, allowing for a more fault resilient behaviour in the event the backend system shows some instability, as the clients will always be decoupled from the CASTOR backend.

This architecture leverages the existing framework that has been designed and developed for the CASTOR mass storage system itself [1]. The entire Entity-Relationship (E-R) schema has been designed using the UML methodology, and a customized code generation facility, maintained in the CASTOR framework, has been used to generate the C++ access layer to the database.

0-7695-3025-7/07 $25.00 © 2007 IEEE

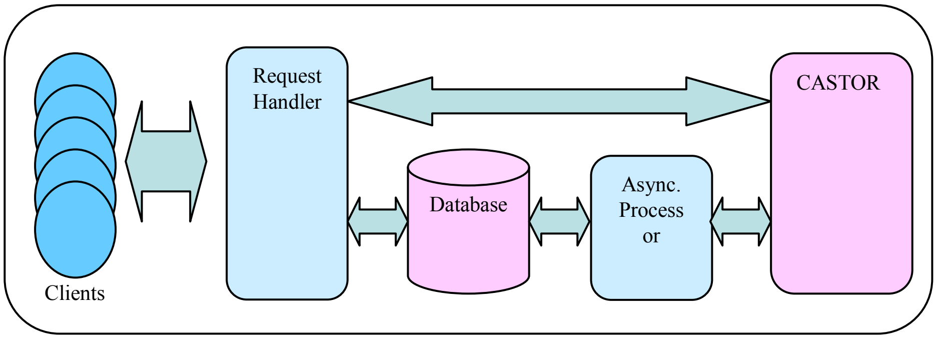

Figure 3: The architecture of the CASTOR SRM

***dCache-SRM***

dCache is a  Mass Storage System developed jointly by Fermilab and DESY which federates a large number of disk systems on heterogeneous server nodes to provide a storage service with a unified namespace. dCache provides multiple means of file access protocols, including FTP, Kerberos GSSFTP, GSIFTP, HTTP, and dCap and xRootD,  POSIX APIs to dCache. dCache can act as a standalone Disk Storage System or as a front-end disk cache in a hierarchical storage system backed by a tape interface such as OSM, Enstore [7], Tsm, HPSS [8], DMF or Castor [2]. dCache storage system, shown in Figure 4, has a highly scalable distributed architecture that allows easy addition of new services and data access protocols.

dCache provides load balancing and replication across nodes for “hot” files, i.e. files that are accessed often. It also provides a resilient mode, which guarantees that a specific number of copies of each file are maintained on different hardware. This mode can take advantage of otherwise unused and unreliable disk space on compute-nodes. This is a cost-effective means of storing files robustly and maintaining access to them in the face of multiple hardware failures.

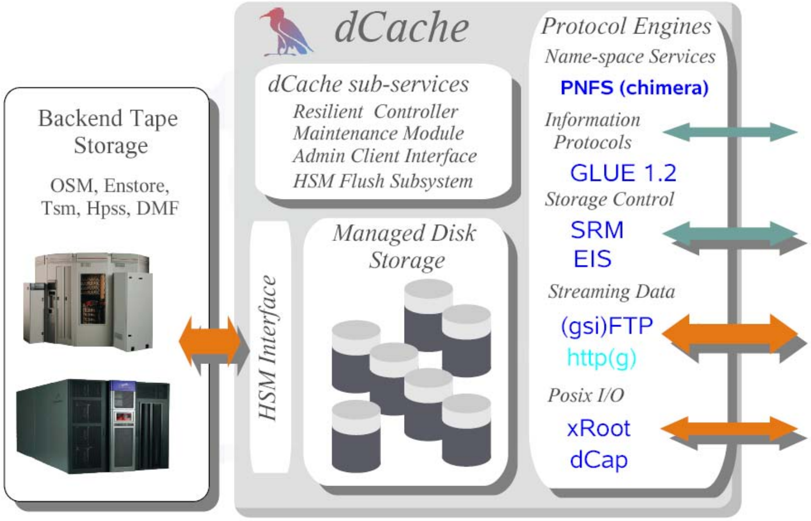

Figure 4: The Architecture of dCache

The dCache Collaboration continuously improves the features and the Grid interfaces of dCache. It has delivered the gPlazma element that implements flexible Virtual-Organization (VO)-based authorization. DCache’s GridFTP and GsiDCap services are implementations of the grid aware data access protocols. But the most important step to connect dCache to the Grid was the development of the SRM interface.

dCache has included an implementation of  SRM Version 1.1 since 2003 and now has all protocol elements of SRM v2.2 required by the WLCG. The new SRM functions include space reservation, more advanced data transfer, and new namespace and access control functions. Implementation of these features required an update of the dCache architecture and evolution of the services and core components of the dCache Storage System. Implementation of SRM Space Reservation led to new functionality in the Pool Manager and the development of the new Space Manager component of dCache, which is responsible for accounting, reservation and distribution of the storage space in dCache. SRM's new "Bring Online" function, which copies tape-backed files to dCache disk, required redevelopment of the Pin Manager service, responsible for staging files from tape and keeping them on disk for the duration of the Online state. The new SRM concepts of AccessLatency and RetentionPolicy led to the definition of new dCache file attributes and new dCache code to implement these abstractions. SRM permission management functions led to the development of the Access Control List support in the new dCache namespace service, Chimera

24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007)
0-7695-3025-7/07 $25.00 © 2007 IEEE

### *DPM – Disk Pool Manager*

The DPM (Disk Pool Manager) aims at providing a reliable and managed disk storage system for the Tier-2 sites. It is part of the EGEE project. It currently supports only disk-based installations. The architecture is based on a database and multi-threaded daemons (see Figure 5):

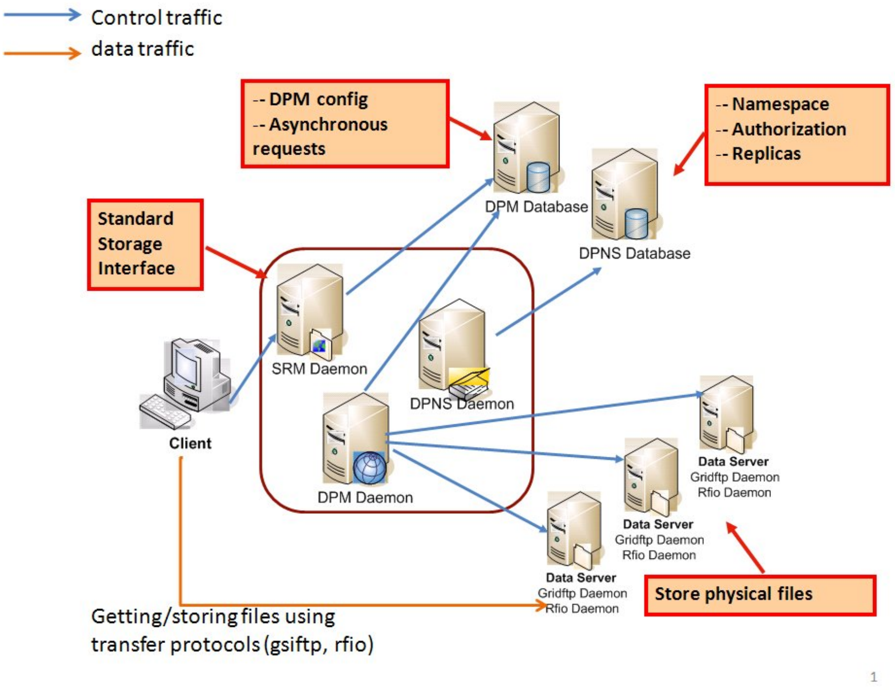

Figure 5: Overview of the DPM architecture

- The dpns daemon controls the hierarchical namespace, the file permissions and the mapping between SFN (Site File Name) and physical names; An SFN is the file path portion of an SURL.
- The dpm daemon manages the configuration of disk pools and file systems. It automatically handles the space management and the expiration time of files. It also processes the requests.
- The SRM (v1.1 and v2.2) daemons distribute the SRM requests workload (delete, put, get, etc);
- The Globus gsiftp daemon provides secure file transfers between the DPM disk servers and the client;
- The rfio daemon provides secure POSIX file access and manipulation.

In most cases, all the core daemons are installed on the same machine. However for large deployment, they can run on separate nodes.

Although not represented in Figure 5, https and xrootd [19] protocols can be used to access data.

A database backend (both MySQL and Oracle are supported) is used as a central information repository. It contains two types of information:

- Data related to the current DPM configuration (pool and file system) and the different asynchronous requests (get and put) with their statuses. This information is accessed only by the DPM daemon. The SRM daemons only put the asynchronous requests and poll for their statuses.
- Data related to the namespace, file permissions (ACLs included) and virtual IDs which allow a full support of the ACLs. Each user DN (Distinguished Name) or VOMS (Virtual Organization Membership Service) attribute is internally mapped to an automatically allocated virtual ID. For instance, the user Chloe Delaporte who belongs to the LHCb group could be mapped to the virtual UID 1427 and virtual GID 54. This pair is then used for a fast check of the ACLs and ownership. This part is only accessed by the DPNS daemon.

The GSI (Grid Security Infrastructure) ensures the authentication which is done by the first service contacted. For instance, if it is an SRM request, then the SRM daemon does the authentication. The authorization is based on VOMS.

The load balancing between the different file systems and pools is based on the round robin mechanism. Different tools have been implemented to enable users to manipulate files in a consistent way. The system is rather easy to install and to manage. Very little support is needed from the developers' team. The DPM is currently installed at roughly 80 sites. For a given instance, the volume of data managed ranges from a few TB up to 150 TB of data. So far no limit on the volume of data has been reported.

### *StoRM - Storage Resource Manager*

StoRM [3] (acronym for Storage Resource Manager) is an SRM service designed to manage file access and space allocation on high performing parallel and cluster file systems as well as on standard POSIX file systems. It provides the advanced SRM management functionalities defined by the SRM interface version 2.2 [14]. The StoRM project is the result of the collaboration between INFN – the Italian National Institute for Nuclear Physics - and the Abdus Salam ICTP for the EGRID Project for Economics and Finance research.

StoRM is designed to respond to a set of requests coming from various Grid applications allowing for standard POSIX access to files in local environment, and leveraging on the capabilities provided by modern parallel and cluster file systems such as the General Parallel File System (GPFS) from IBM. The StoRM service supports guaranteed space reservation and direct access (by native POSIX I/O calls) to the storage resource, as well as supporting other standard Grid file access libraries like RFIO and GFAL.

More generally, StoRM is able to work on top of any standard POSIX file system providing ACL (Access Control List) support, like XFS and ext3. Indeed, StoRM uses the ACLs provided by the underlying file system to implement the security model, allowing both Grid and local access. StoRM supports VOMS [17] certificates and has a flexible authorization framework based on the interaction with one or more external authorization services to verify if the user can perform the specified operation on the requested resources.

Figure 6 shows the multilayer architecture of StoRM. The are two main components: the frontend, that exposes the SRM web service interface and manages user authentication, and the backend, that executes all SRM functions, manages file and space metadata, enforces authorization permissions on files, and interacts with file transfer services. StoRM can work with several underlying file systems through a plug-in mechanism that decouples the core logic from the specific file system functionalities. The specific file system driver is loaded at run time.

To satisfy the availability and scalability requirements coming from different Grid applications scenarios, one or more instances of StoRM components can be deployed on different machines using a centralized database service. Moreover, the namespace mechanism adopted by StoRM makes it unnecessary to store the physical location of every file managed in a database. The namespace is defined in an XML document that describes the different storage components managed by the service, the storage areas defined by the site administrator and the matching rules used at runtime to map the logical to physical paths. The physical location of a file can be derived from the requested SURL, the user credentials and the configuration information described in the XML document.

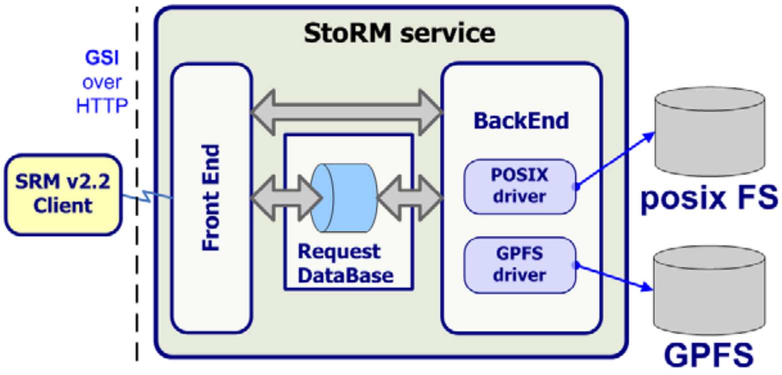

Figure 6: StoRM Architecture

## 5. The testing procedure

An important aspect in the definition of the SRM v2.2 protocol is the verification against existing implementations. The verification process has helped understanding if foreseen transactions and requirements make sense in the real world, and identifying possible ambiguities. It uncovered problematic behaviors and functional interferences early enough in the definition cycle to allow for the protocol specification to be adjusted to better match existing practices. The verification process has shown if the protocol adapted naturally and efficiently to existing storage solutions. In fact, it is crucial that a protocol is flexible and does not constrain the basic functionality available in existing services. As an example we can mention the time at which a SURL starts its existence in the namespace of an SRM. Implementations like dCache mark a file as existent in the namespace as soon as a client starts a transfer for the creation of the file. This is to avoid the need for cleanup of the name space when the client never gets to write the file. Other implementations, instead, prefer to reserve the name space entry as soon as possible, to present a consistent view to all concurrent clients, or to simplify the interfacing with the MSS backend.

The verification process has helped proposing and refining a conceptual model behind the protocol, with an explicit, clear and concise definition of its underlying structural and behavioral concepts. This model has made it easier to define the service semantics, helped implementation developers, and provided for a more rigorous validation of implementations. The model is a synthetic description of a user's view of the service, with the basic entities (such as space, file,…), their relationships, and the changes they may go through. The model is described in some details in [6].

The analysis of the complexity of the SRM interface through its formal model shows that a high number of tests need to be executed in order to fully check the compliance of the implementations to the specifications. Therefore, an appropriate testing strategy has to be adopted in order to reduce the number of tests to be performed to a manageable level, while at the same time covering those aspects that are deemed to matter in practice.

Testing activities aim at finding differences between the actual and the intended behavior of a system. In particular, [10] gives the following definition: “Testing is the process of executing a program with the intent of finding errors.” A test set is defined to be exhaustive if and only if it fully describes the expected semantics of the specifications, including valid and invalid behaviors.

In order to verify the compliance to a protocol of a specific implementation a test-case-design methodology known as Black Box testing is often used. The Black Box testing technique focuses on identifying the subset of all possible test cases with the highest probability of detecting the most errors. In particular, the most popular black box testing approaches are Equivalence partitioning, Boundary value analysis, Cause-effect graphing and Error guessing [10]. Each of these approaches covers certain cases and conditions but they do not ensure the identification of an exhaustive testing suite.

The black box testing technique has been used to design 5 families of tests to verify the available implementations of SRM v2.2. Furthermore, many hypotheses have been made in order to make the model simpler and to reduce the total number of tests, while keeping the test sets valid and unbiased. The 5 families of tests are the following:

- Availability: the srmPing function and a full put cycle for a file is exercised (srmPrepareToPut, srmStatusOfPutRequest, file transfer, srmPutDone). This family is used to verify availability and very basic functionality of an SRM endpoint.
- Basic: the equivalence partitioning and boundary condition analysis is applied to verify that an implementation satisfies the specification when it has a single SRM call active at any given time.
- Use cases: cause-effect graphing, exceptions, functional interference, and use cases extracted from the middleware and user applications are exercised.
- Interoperability: remote operations (servers acting as clients for some basic SRM functions) and cross copy operations among several implementations are executed.
- Stress: the error guessing technique and typical stress situations are applied to verify resilience to load.

A specific language, the S2 [9] has been adopted for a fast implementation of test cases, and the open source implementation is now maintained by WLCG. The S2 language has several attractive characteristics:

- It allows for the quick development of test programs that exercise a single test case each.
- It helps minimize human errors that are typically made in writing test cases.
- It offers an easy way to plug-in external libraries such as an SRM client implementation.
- It offers a powerful engine for parsing the output of a test, expressing the pattern to match in a compact and fully descriptive way.
- It offers a testing framework that supports the parallel execution of tests where the interactions among concurrent method invocations can be tested easily.
- It offers a “self-describing” logging facility that makes it possible to automatically publish the results of a test.

The S2 families of tests run automatically 5 times a day. The results of the tests are published on a web page. In particular, the data of the last run together with the history of the results and their details are stored and made available to the developers through the web. Plots are produced every month on the entire period of testing to track the improvements and detect possible problems.

The testbed that we set up includes five different implementations: CASTOR, dCache, DPM, BeStMan, and StoRM. It currently has 13 available endpoints located in Europe and the US. In particular, 5 endpoints are where the main development happens. These endpoints have been tested for a period of 7 months. The other endpoints have been added recently. They are used to verify that the implementation can accommodate different specific needs at different sites and help smooth the installation and configuration process.

In Figure 7 the availability of the main endpoints over the mentioned period of time is shown. Figures 8,9,10 show the number of failures over the total number of tests executed over time. While for the basic and use case families of tests the errors have improved greatly in a relatively short time, we still have to do some work in terms of interoperability and cross copy operations. Stress testing has just started and some of the available endpoints are being equipped with more resources for that. The instabilities shown in the results usually are caused by service upgrades (to deploy fixes in the code) or circumstances where the server is too busy serving other requests (when the endpoint is a production system not dedicated to tests). Also, underpowered hardware can limit the transaction rates.

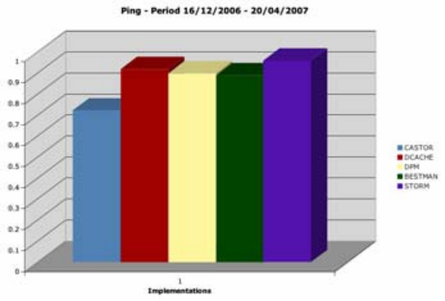

Figure 7: Availability (in percentage) of SRM 2.2 endpoints

The 'srmv2Suite' is built as a perl wrapper gluing all of the 36 individual test modules - corresponding almost one to one to the 38 srmv2.2 methods. Each test module is a small C application, and is built on top of gSOAP 2.6. It was written mainly to allow DPM srmv2.2 implementation, but has also been used to crosscheck some features of BeStMan and dCache SRM v2.2 front-ends. It is most of the time used as a regression test to ease the development lifecycle, and new use cases and specific tests are added as soon as new features become available on the DPM srmv2.2 server. It now includes about 400 different steps, and runs in about 500 sec. Transfers are achieved through Secure Rfio or GridFTP when targeting a DPM server, but are switched back to GridFTP only when testing some other server.

Another SRM test program was developed at LBNL, is being run several times daily, and the results published [12]. S2 and SRM-tester compliment each other in that S2 uses C++ clients while SRM-tester used java clients.

## 6. Publishing SRMs status information

Together with the SRM v2.2 protocol and the data transfer protocols, an information protocol is needed for service discovery and accounting purposes. In service discovery, clients need to check both static and dynamic status information. The GLUE schema [18] is used by several national and international Grids to provide information services for compute and storage resources.

After analyzing the capabilities offered by the SRM, such as the possibility of specifying classes of storage and the negotiation of the file access protocol between client and server, an extensive discussion took place on how much of the configuration information specific to a storage service needed to be exposed to applications, monitoring and accounting facilities. One of the constraints on the schema was that it could not assume that all storage will be provided through SRM implementations. For example, the schema should allow for a simple GridFTP server to be published as a storage service with limited capabilities. Coming up with a flexible model that could satisfy all needs turned out to be quite complicated. As an example, users are interested in the free space for a given storage instance. Defining what "free space" means was not straightforward. One problem was to avoid double counting of storage capacity when a given storage component (tape or disk) is shared among multiple spaces, e.g. for different virtual organizations, while

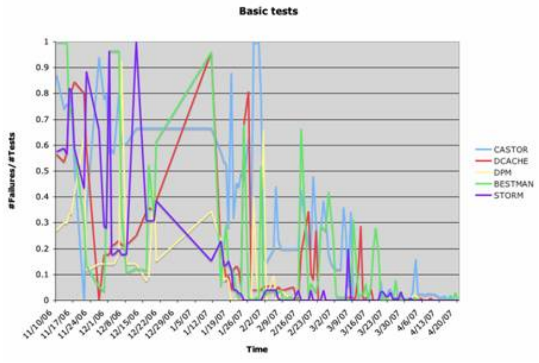

Figure 8: Basic test family: Number of failures/Number of tests over time

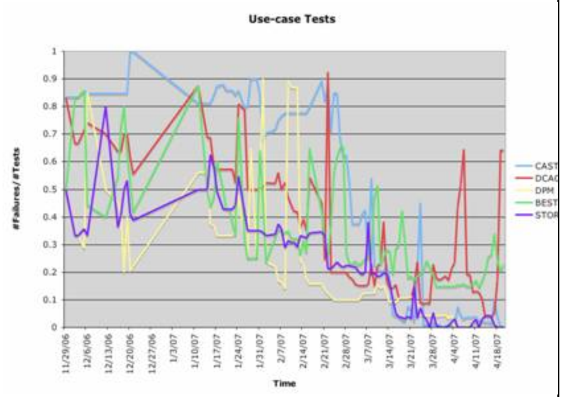

Figure 9: Use-case test family: Number of failures/Number of tests over time

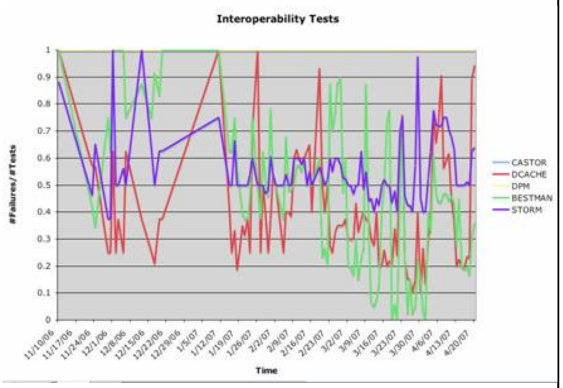

Figure 10: Interoperability test family: Number of failures/Number of tests over time

each of the spaces is published separately. Another interesting quantity is the "used space", for which an unambiguous and useful definition is not obvious either. This space could be in use by files, or allocated by space reservation methods, part of it being potentially available to store new files, or space used by files being migrated to tape but available as soon as the migration is over. For certain implementations some of these numbers may be expensive to keep exact track of. Finally, the proposed information schema for a storage service had to be backward compatible with the one used before SRM v2.2 was introduced. This forced us to make some information unavailable, delaying a more adequate description of the resources to the next major revision of the schema.

## Acknowledgments

Other people who have made contributions to this work include (in no particular order) Owen Synge from DESY, Ákos Frohner, and Laurence Field from CERN, Sergio Andreozzi from INFN, Stephen Burke from RAL, Jiří Mencák (author of the S2 language) from (then) RAL, Ted Hesselroth from FNAL, Andy Hanushevsky from SLAC.

## Conclusions

In this paper, we have described the global collaboration behind the Storage Resource Manager protocol and the definition and validation processes for the SRM protocol that derived from it. We have described the key reasons for the success of SRM, namely, (a) an open protocol, unencumbered by patents or licensing, (b) an open collaboration where any institution willing to contribute can join, (c) a well establish validation process (d) the existence of five interoperating implementations, many of which are open source. We have described how the SRM interfaces diverse storage systems to the Grid, from single disk over distributed file systems, to multi-petabyte tape stores.

The fact that the protocol supports advanced capabilities such as dynamic space reservation enables advanced Grid clients to make use of these capabilities, but since storage systems are diverse, implementation support for capabilities must be optional. On the Grid, SRM is complemented by the very widely used GLUE information schema, which allows clients to discover services supporting the right capabilities.

Finally, we have described how our test collaboration has been crucial to the definition of the protocol, its validation and the interoperability of the implementations, with a range of tests from individual functions in the API to whole use cases and control flow. Not only are interoperability problems discovered before the users do, thus leading to improved perception of the SRM services in the users' view, but the testing also allows advanced but optional features to be tested incrementally as they become supported by each implementation.

## References

[1] O. Bärring, R. Garcia Rioja, G. Lo Presti, S. Ponce, G. Taurelli, D. Waldron, *CASTOR2: design and development of a scalable architecture for a hierarchical storage system at CERN*, CHEP, 2007.

[2] http://castor.web.cern.ch/castor/

[3] Corso, E. and Cozzini, S. and Donno, F. and Ghiselli, A. and Magnoni,, L. and Mazzucato, M. and Murri, R. and Ricci, P.P. and Stockinger, H. and Terpin, A. and Vagnoni, V. and Zappi, R., "StoRM, an SRM Implementation for LHC Analysis Farms Computing in High Energy Physics", CHEP'06, Feb. 13-17, 2006, Mumbai, India, http://indico.cern.ch/contributionDisplay.py?contribId=373&sessionId=13&confId=048.

[4] http://www.cs.wisc.edu/condor/

[5] http://www.dcache.org/

[6] A. Domenici, F. Donno, A Model for the Storage Resource Manager, Int. Symposium on Grid Computing 2007, 26-29 March 2007, Academia Sinica, Taiwan

[7] http://www-ccf.fnal.gov/enstore/

[8] http://www.hpss-collaboration.org/hpss/index.jsp

[9] J. Mencak, F. Donno, The S2 testing suite, http://s-2.sourceforge.net

[10] G. J. Myers, C. Sandler (Revised by), T. Badgett (Revised by), T. M. Thomas (Revised by) The ART of SOFTWARE TESTING 2 edition, December 2004.

[11] http://www.sdsc.edu/srb/index.php/Main_Page

[12] SRM Storage Tests and Monitoring, http://datagrid.lbl.gov/

[13] http://sdm.lbl.gov/srm-wg

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

[15] Arie Shoshani, Alex Sim, Junmin Gu, Storage Resource Managers: Middleware Components for Grid Storage, Nineteenth IEEE Symposium on Mass Storage Systems, 2002

[16] Arie Shoshani, Alexander Sim, and Junmin Gu, Storage Resource Managers: Essential Components for the Grid, in Grid Resource Management: State of the Art and Future Trends, Edited by Jarek Nabrzyski, Jennifer M. Schopf, Jan weglarz, Kluwer Academic Publishers, 2003

[17] The Virtual Organization Membership Service, http://www.globus.org/grid_software/security/voms.php

[18] http://glueschema.forge.cnaf.infn.it/Spec/V13

[19] http://xrootd.slac.stanford.edu/