RRS: Replica Registration Service for Data Grids

Arie Shoshani, Alex Sim, Kurt Stockinger

Computational Research Division
Lawrence Berkeley National Laboratory
University of California
1 Cyclotron Road, Berkeley, California 94720, USA
{AShoshani, ASim, KStockinger}@lbl.gov

Abstract. Over the last few years various scientific experiments and Grid projects have developed different catalogs for keeping track of their data files. Some projects use specialized file catalogs, others use distributed replica catalogs to reference files at different locations. Due to this diversity of catalogs, it is very hard to manage files across Grid projects, or to replace one catalog with another.
In this paper we introduce a new Grid service called the Replica Registration Service (RRS). It can be thought of as an abstraction of the concepts for registering files and their replicas. In addition to traditional single file registration operations, the RRS supports collective file registration requests and keeps persistent registration queues. This approach is of particular importance for large-scale usage where thousands of files are copied and registered. Moreover, the RRS supports a set of error directives that are triggered in case of registration failures. Our goal is to provide a single uniform interface for various file catalogs to support the registration of files across multiple Grid projects, and to make Grid clients oblivious to the specific catalog used.

1 Introduction

Managing a large number of files at distributed locations is one of the challenges that many large-scale scientific experiments face. For efficiency reasons, many of the files are replicated in multiple storage systems. In order to keep track of the files and their replicas, various file and replica catalogs are used. Typical questions are: What is the name-space for files registered in the catalogs? Shall the catalog be organized as a centralized or decentralized service? How can information about files be retrieved efficiently? How can different sites interact with each other's catalogs? In order to solve some of these issues, various Grid projects have developed different catalogs for keeping track of their data files. Some experiments use specialized file catalogs, others use distributed replica catalogs to reference files at different locations. This diversity of catalogs makes it very hard to manage files across Grid projects or even within a single project. The solution to this problem is not to attempt and standardize a particular file catalog system. Rather, the approach taken here is to provide a uniform specification of a functional interface that permits the multiplicity of catalogs to co-exist. This approach is similar to many commercial products (such as relational database systems) that have a common functional interface specification that permits multiple systems to co-exist. Furthermore, our approach isolates the Grid client programs from the specific catalog system used.

In this paper we introduce a Grid service called the Replica Registration Service (RRS), that provides a uniform functional interface to various file catalogs, replica catalogs, and meta data catalogs. It can be thought of as an abstraction of the concepts used in catalog systems to register files and their replicas. Some experiments may prefer to support their own file catalogs (which may have their own specialized structures, semantics, and implementations) rather than use a standard replica catalog. Providing a RRS that can interact with such a catalog can permit that catalog to be invoked as a service in the same way that other more general-purpose replica catalogs do. If at a later time the experiment wishes to change to another file catalog, it is only a matter of developing a RRS for that new catalog and replacing the existing catalog. Similarly, an existing replica location service (RLS, [3]) that supports the RRS interface can be plugged in instead of the current catalog. In addition, some systems use meta-data catalogs or other catalogs to manage the file name-spaces, and those could be accessed through the same RRS interface as well.

The main contributions of this paper are as follows:

- We introduce a novel Grid service - the Replica Registration Service. We discuss the design considerations and the main functional interface components.
- We report on our experience of using an early implementation of a RRS in a production environment. The results show that the RRS could greatly simplify the management of replicas and reduce registration errors.

2 Related Work

An early version of a replica management framework is presented in [4] where the terms *Replica Management* and *Replica Selection* are defined within the context of a Data Grid. In the European Data Grid Project a similar replica management framework was implemented [7]. An integrated approach for data and meta-data management is provided in the Storage Resource Broker (SRB) [1]. In general, data replication can be done at the file or object level, where multiple objects can either be stored in a single file, or a single object can be stored across multiple files. The differences between object and file replication are discussed in [13]. However, in practice only file replicas are cataloged so far, since the number of objects can be much larger than the number of files. Moreover, the implementation of object replication systems is more complex. For this reason, we focus on file replica registration only in this paper.

In the early days of Grid computing, replicas were stored in LDAP catalogs. Due to performance issues, subsequent replica catalogs or replica location services stored the replicas in relational databases [3]. In recent projects, various file and replica catalogs were implemented for different experiments that are often not compatible [2, 6].

Currently, different organizations that use the Grid, develop their own specialized version of replica catalogs. They vary in their functionality greatly. For example, some support GUIDs and LFNs, some support only a single LFN; some support logical and physical directories and some only a single level; some include extensive meta-data information, such as file usage and some have no meta-data at all. Our purpose in this paper is to identify a reasonable set of replica registration functionality independent of any specific implementation. We hope that this approach will provide a uniform interface that will allow for multiple implementations to co-exist.

3 Terminology and Name-space

3.1 GUIDs, LFNs, and PFNs

There are several ways to refer to a file. If the location of the file is known, one can specify its Physical File Name (PFN). However, since a file may have multiple replicas, it is convenient to refer to the file by using Logical File Names (LFN). Some communities of users prefer to support a unique immutable LFN for each file, and provide a mapping between the LFN and one or more Physical File Names (PFNs). In many cases, LFNs are designed to be structured names. This is a desired property, since the file name conveys a meaning, such as the date, purpose, or conditions that were used at the time the file was generated. However, having such structured names makes it difficult to guarantee global uniqueness of the name. Furthermore, there may be a need to change file names over time, or even have multiple aliases for a file name. For this reason, some communities use a *globally unique identifier*, referred to as GUID, to identify a file, in addition to a LFN. Given that a GUID is used for a file, that file can now have multiple LFNs that are treated as name-aliases for the GUID.

Since we wish to have this specification applicable to all communities, we adopt the more general case of having a GUID for a file. In addition, we permit multiple LFNs per GUID. For communities that only use a single LFN and no GUID, we consider that LFN to be equivalent to a GUID.

The one-to-many relationships between a GUID-to-PFNs and a GUID-to-LFNs are shown schematically in Figure 1. Note that we use SURL (Site URLs) as a generalization of PFNs. The reasons for that are explained next in Section 3.2.

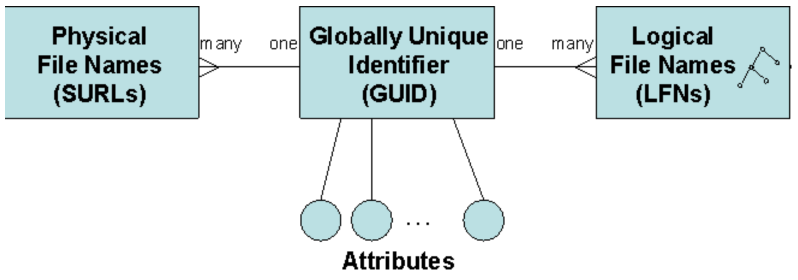
Physical File Names (SURLs)
many
one
Globally Unique Identifier (GUID)
one
many
Logical File Names (LFNs)
...
Attributes

Fig. 1. The relationships between SURLs, GUIDs, and LFNs.

3.2 SURLs as a Generalization of PFNs

Specifying a Physical File Name (PFN) is straightforward. It is specified as a URL made of the format: protocol://machine:port/directory-path/file-name. For example: gridftp://cs.berkeley.edu/home/data. Note that the protocol specified in this case is the transfer protocol.

Some storage systems support multiple physical devices and multiple directories, and may want to have the freedom of changing the physical location of a file without changing the reference to it by Grid clients. An example of a software layer that permits this functionality is a Storage Resource Manager (SRM) [9]. The SRM is a single endpoint for accessing a file regardless of its physical location on a particular site. The site is a virtual entity referring to the collection of resources under the administrative control of the site manager. The concept of a site permits a single filename to be assigned to a physical file regardless of it physical storage location.

The reference for a file on a site is called a *Site URL* (SURL). For example: srm://data.berkeley.edu:4004/dir/data is the name of a file managed by a SRM residing on the machine data.berkeley.edu, on port 4004. When requesting the file from the SRM using the SURL, the SRM returns the *transfer URL* (TURL) which is the actual PFN. For example, for the file above the SRM may return the PFN on another machine, cs.berkeley.edu, by using the URL gridftp://cs.berkeley.edu/home/data. Note that a PFN is a special case of an SURL, where the transfer service specified by the protocol is the site endpoint. Therefore, we only use SURL in the remainder of this document.

3.3 The LFN Name-space Structure

LFNs are commonly organized into directory structures, similar to any file system (such as the Unix file system). Some file systems consider directory names as LFNs as well and assign GUIDs to them (this can be automatically assigned by the catalog). The value of treating directories as LFNs is that one can refer to a directory path in a similar way as a reference to a file. In this specification we allow the creation and removal of directories and references to them, so that systems that support this feature will be accessible through the RRS. This is shown schematically in the box referring to LFNs in Figure 1 by having the directory icon in it.

A common use case for using multiple LFNs is that a file is first registered with a particular LFN, and then additional LFNs are allowed to refer to the same file. The original registration is sometimes referred to as the *primary LFN*, and subsequent references to it are referred to as *secondary LFNs* or as *LFN-aliases*. We do not find this distinction generalizable, useful, or necessary. Therefore we refer to all LFNs in the same way regardless of when they were defined. Thus, the RRS interface does not permit references to *primary LFNs*, only to LFNs.

3.4 File Attributes

File attributes are associated with GUIDs as shown in Figure 1. These attributes represent only global properties that do not depend on where the file resides (the

SURL site) and how it is named (its LFNs). Some attributes are considered essential to verify the correctness of file transfers. These attributes are: fileSize, checkSumType, and checkSumValue. We refer to these attributes as *core attributes*. In addition, there may be other attributes that the underlying catalogs may store. We permit the entry and retrieval of all such attributes through the RRS interface. When requesting these attributes, one can refer to *core attributes* only, or to "all" attributes. The RRS will return an array of triples: fileAttributeName, fileAttributeType, and fileAttributeValue. Note that all values are passed through the interface as strings. The fileAttributeType refers to the type of the attribute, as it is stored in the underlying catalog. The RRS was designed to have functions to retrieve the attribute values.

4 Replica Management Architecture

Before discussing the Replica Registration Service, we provide an outline of the replica management architecture which is based on the following three components: (a) the Replica Selection Service (RSS), (b) the Replica Copying Service (RCS), and (c) the Replica Registration Service (RRS). This is shown in Figure 2.

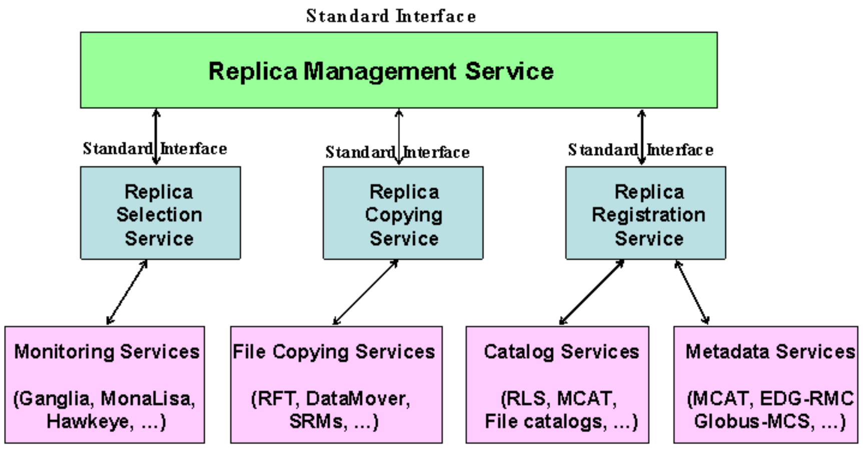
Standard Interface
Replica Management Service
Standard Interface
Standard Interface
Standard Interface
Replica Selection Service
Replica Copying Service
Replica Registration Service
Monitoring Services (Ganglia, MonaLisa, Hawkeye, ...)
File Copying Services (RFT, DataMover, SRMs, ...)
Catalog Services (RLS, MCAT, File catalogs, ...)
Metadata Services (MCAT, EDG-RMC Globus-MCS, ...)

Fig. 2. Replica Management Architecture.

Given a GUID or a LFN of a file that has to be replicated to a target site, the Replica Management Service (RMS) first invokes the RRS in order to find all possible replicas. It can then choose to ignore some of these replicas based on its own policies, such as getting replicas only from a certain region of the world. The RMS can then invoke the RSS. The RSS's function is to order the replicas according to cost estimates of copying the physical files to the target site. In order to determine the copying costs, the RSS may consult various monitoring services. Once the physical file is selected for copying, the RMS invokes the RCS in order to copy the file to its target destination. Note that the RMS may not choose the source replica with the smallest cost. It may choose a site based on policy information about what sites to avoid (for example, to prevent bottlenecks at some sites). The RCS may use various data copying services, such as the Reliable File Transfer (RFT) service [8], Storage Resource Managers (SRMs) [9], or the DataMover [11] service that relies on SRMs. After each file is copied, the RMS can communicate with the RRS in order to register the files. Depending on the mode requested, the RRS registers the files immediately or in a delayed mode. The RRS interacts with a file catalog, a replica catalogs, and/or a meta data catalog depending on its configuration.

The orchestration between the services is mainly the coordination between copying and registration of files. As discussed above, the client should be provided with a choice of modes, such as register only if the entire multi-file copying is successful, or register only the files that were successfully copied and report failures. However, in addition to coordination between the underlying services, the RMS can be expected to provide the functionality of recovering from failures. For example, suppose that a specific physical file was selected to be copied to the target location, and that the copying failed because the file was not found (it could have been removed in the interim). The RMS can then select an alternative physical file from another site based on the information provided by the RSS. This type of functionality is expected from a well-functioning, robust, service. For this purpose one or more RRSs can be known to the RMS as part of this configuration.

5 RRS Design Considerations

In this section we outline the functionality of the Replica Registration Service. A detailed discussion of the interface can be found in [10].

The functionality of the interface specification is split into two parts, namely, the *Basic API* and *Advanced API*. The Basic API covers file registration, unregistration, and command status operations. The Advanced API introduces file attributes and name space management. A summary of the most important commands is given in Tables 1 and 2. In this paper we only discuss the design consideration of the Basic API. The Advanced API is similar to familiar file management functions and is therefore not discussed here.

5.1 Registration Functions

All the registration functions are for files only. Files can be referred to by their GUID, LFNs, or SURLs. We use the term *file references* to refer to any of these names. All registration requests are made of pairs of file references, such as (LFN, SURL). The first item of this pair is referred to as *given* and the second item as *toBeRegistered*. For example, register (LFN, SURL) is interpreted as *for the given LFN, register the SURL*. Similarly, register (SURL, SURL) is interpreted as *for the given (first) SURL, register the (second) SURL*. In such cases, the RRS may need to get first the GUID for the existing file reference (if it is not a GUID), and then register the second file reference using the GUID. In some cases, we allow a file reference to be null, such as the first-time registration of a LFN without providing a GUID, denoted as (–, LFN). In this case, it is expected that the underlying catalog

Command	**Explanation**
Register Functions	
openCollectiveRegistration	Prepare for multiple registration calls with specific registration mode and error directive. The system returns a request token.
register	Register one or more file references using the request token.
closeCollectiveRegistration	Close a specific collective registration using the request token.
getRegistrationStatus	Retrieve information on file status (done, in progress, pending) and error codes (file not found, already exists,...).
abortRegistration	Stop registration request and unregister files.
unregister	Unregister files.
getUnregisterStatus	Retrieve status of unregister request.
Discovery Functions	
getFileReferences	Retrieve GUID, LFN and SURLs of a given file reference.
getFileReferenceStatus	Retrieve status of file reference.

Table 1. Basic API

Command	**Explanation**
Attribute Functions	
getFileAttributes	Get file attributes such as *checkSum* or *fileSize.*
getFileAttributesStatus	
Name Space Management	
makeDirectory	Create a directory and register it in the catalog.
removeDirectory	Remove directory.
listDirectory	List the content of a directory.
getListDirectoryStatus	

Table 2. Advanced API

will generate the GUID. This is explained in more details in the section on first-time and subsequent registration.

Note that from the discussion above, it is obvious that all registration actions are for file references. However, in the remainder of this article we often use the term *register a file* as a short form for *register a file reference.*

Collective Registration The RRS is designed to allow the coordination between copying and registration of files. Because copying a large number of files can be a slow process, it is necessary to allow the registration process to be a long-lasting activity. Therefore, it is necessary to have a way of specifying the beginning and the end of multiple registrations. This is achieved by starting with an openCollectiveRegistration function, followed by one or more register functions, and ending with a closeCollectiveRegistration function.

Registration Modes As one or more files are registered, the RRS can use different modes of registration as requested by the user. A client may prefer that files be registered one-at-a-time as soon as each file is copied successfully, or may prefer to register all the files only after the entire request of copying multiple files (or entire directories) is successful. We refer to the desired behavior as the *registration mode*. Accordingly, the two registration modes supported are: *continuous* and *atEnd*. *continuous* means: register as soon as possible, and *atEnd* means: register all the files after the closeCollectiveRegistration function is called. Another advantage of this choice is that it allows the burden of accumulating the deferred registration of multiple files (until all the copying is finished) to be placed on the RRS; that is, the RRS has to accept and manage delayed registration requests. Thus, the client or the component calling the RRS does not have to save the deferred registration requests. Instead, it can pass them on to the RRS.

The implementation of the registration modes behavior depends on the target catalog. Some catalogs permit bulk registration of files, a feature that the RRS can choose to take advantage of. Some may prefer a limited number of files in the bulk registration (such as 100 at a time), and some may allow only a single file registration at-a-time. The RRS has to perform the registration as close as possible to the requested mode.

Error Directives Registration to the underlying catalog(s) may result in unrecoverable errors. Typical errors are that a GUID, LFN or SURL is not found. For example, registering an SURL for an given LFN may result in an error that the LFN was not found, or that the SURL already exists.

Under failure conditions, clients may prefer different behaviors. We refer to this as *error directives*. These can be specified at the time the collective registration request is initiated with the openCollectiveRegistration function. Three error directives are supported:

- *stop*: register files until a non-transient error occurs and stop.
- *stopAndUndo*: register files until a non-transient error occurs, stop, and unregister all the files submitted for registration so far (undo).
- *continue*: record the error and continue registering files.

If a registration request involves thousands of files, it may be unwise to stop or undo the entire request because of a single error. It may be better to permit a few errors before the error directive gets triggered. We allow for such a parameter, called the *error directive trigger*, to be set as an integer. Regardless of whether the trigger occurs, the RRS records all such errors.

First-Time and Subsequent Registration The registration of files into a catalog requires the distinction between a *first-time registration*, and *subsequent registrations*. During a first-time registration, a unique GUID needs to be provided by the requester, or is automatically generated by the underlying catalog GUID generation service. Some simple file catalogs use the source physical file name, that was first registered, as the GUID. In other catalogs the GUID is generated by its own *GUID generator* that guarantees a globally unique identifier. Our goal is to have a single Replica Registration Service (RRS) that can accommodate special purpose file catalogs (such as a file catalog of a scientific experiment), catalog services (such as the RLS), or other more general catalogs.

As mentioned above, all file registrations have pairs of file references, such as (LFN, SURL). For subsequent registrations the first item of this pair is referred to as *given* and the second item as *toBeRegistered*. Thus, the *given* item has to be found in the underlying catalog, while the second item should not exist. In contrast, for first-time registration, such as (GUID, LFN), both items need to be registered, and therefore both should not exist in the underlying catalog.

When a subsequent registration is requested, the RRS needs to verify that the *given* file reference is already registered. For example, a registration of a (GUID, SURL) implies that a new SURL is registered for that GUID. The RRS needs to check that the GUID already exists, and also check that the SURL does not exist. To allow full flexibility, we allow the registration of a LFN or a SURL given an existing GUID, LFN, or a SURL. This can simplify the client interaction with the RRS. For example, register (LFN, SURL) may require the client to get the GUID for the LFN first from a meta-data catalog, and then register the (GUID, SURL) to a replica catalog. The RRS is designed to save the client from having to do this extra step.

To summarize, all registration requests are made of pairs of file references. For first-time registration, the registration of (GUID, LFN) or a (GUID, SURL) implies that the first file reference, the GUID, needs to be registered as well, and therefore should not exist. In subsequent registration, the first file reference must exist and the second file reference should not exist in the underlying catalogs. The RRS relies on the underlying catalogs to verify correctness. Therefore, catalogs should provide verification of the existence of file references.

5.2 Unregister Function

The unregister function can be used to refer to a registration previously made by a collective registration request by using the request-token. The most general case is when only a request-token is provided without any specific file references. This is interpreted as *global-unregister* that means *unregister all the file reference registrations in that request*. However, since this is a global operation, and can cause serious difficulties if a mistake is made, we added a flag called *unregisterCollectiveRequest* that also has to be set. Note that we do not consider unregisterCollectiveRequest meaningful before the closeCollectiveRegistration function is called, and therefore will return an error in that case, saying *cannot unregister entire request before closeCollectiveRegistration is called*.

All the other cases to consider are requests to unregister specific file references. The specification of what to unregister can be done using pairs of file references. Similar to the subsequent *register* case, for the *unregister* function the first file reference in the pair is interpreted as *given* and the second as *toBeUnregistered*. For example, unregister (LFN, SURL) is interpreted as *for the given LFN, unregister the SURL.*

We note that a request to unregister some files can be issued while a collective registration request is in-progress (i.e. not closed yet) and therefore some of the files that need to be unregistered may still be in the RRS queue. In such a case the RRS needs to remove these requests from the queue, and unregister files that were already registered. Specifically, the RRS should suspend the collective registration process, perform the unregistration as requested, and then continue with subsequent registration requests. The same is the case after the request is closed, but the actual registration is still in-progress. The RRS needs to check the status of the request, and act accordingly.

6 Replica Registration Service in Production Use

We have implemented a Replica Registration Service (RRS) for the STAR experiment which has been used in production for over a year now. The STAR experiment [12] is a high-energy nuclear physics experiment producing real data at Brookhaven National Laboratory (BNL). The data files are replicated daily from BNL to another laboratory, Lawrence Berkeley National Laboratory (LBNL) for post-processing and analysis. The registration of the files at the LBNL site was done manually or by using scripts, a process that was prone to errors. In production, there are several thousands of files registered per month, for a total volume that averages more than 5 Terabytes.

The RRS now used in the STAR experiment automated the registration process, and practically eliminated the error rates (from about 1% to 0.02%), according to the person in charge of the replication operation [5]. The RRS was implemented as a daemon module that listens for information provided by the component responsible to replicate the files, called a DataMover [11]. The DataMover interacts with two Storage Resource Managers (SRMs), one at the source site (BNL) and one at the target site (LBNL). When the target SRM receives a file and archives it, it notifies the RRS. It is the responsibility of the RRS to register the files. The registration is made to a STAR file catalog (which uses a relational database, MySQL), by invoking a script. The production setup is shown in Figure 3.

The RRS was implemented having three modes: *continuous*, *every-n-files*, and *at-end*. In the *continuous mode*, the RRS tries to register each file immediately. In the *every-n-files mode*, the RRS queues the registration requests until it has n files, and then registers then n files in a single *bulk* registration. In the *at-end mode*, the RRS waits till the replication of the *entire set* of files is successful, and only then registers them. Another function of the RRS is to make sure that successful registration occurs when the File Catalog is temporarily unavailable. If the File Catalog is busy or down, the RRS keeps trying periodically until the registration is successful. This alleviates the burden from the client of having to keep track of successful registrations. Initially we experimented with the *every-n-files mode* of registration. However, we found that in this application of large-scale continuous replication, the *continuous mode* was effective as the *every-n-files mode*.

It is worth mentioning that we considered the option of having the RRS being a client that *pulls* the information from the SRM, but this design would require the RRS to poll the SRM continuously. We chose, instead, to implement the RRS as a daemon, so that the SRM can *push* the information to the RRS. This reduced the communication overhead significantly. However, a more general replication service

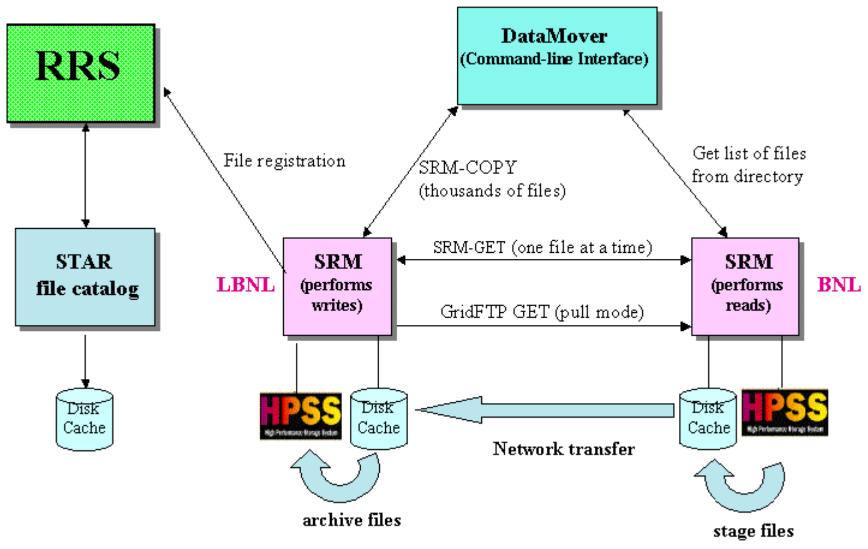

Fig. 3. Replica Registration Service used in production for a physics experiment.

may choose to *poll* the SRM or the target storage system, and then notify the RRS of files that were already transferred.

The RRS was designed to interact with various catalogs, where most of the code is reusable and only the request to register files changes depending on the target catalog. By having a uniform interface it should be possible to replace one catalog by another if necessary without effecting the client programs.

In general, a Virtual Organization (VO) can use multiple RRS services. The coordination between these RRS services must be done by the RMS layer according to their configuration for keeping track of file assignments to storage systems. However, this is a VO issue that is beyond the scope of this paper.

7 Conclusions

In this paper we introduced the Replica Registration Service (RRS) that provides a uniform interface to various file catalogs, replica catalogs, and meta data catalogs. The RRS supports collective file registration requests and keeps persistent registration queues for large-scale usage where thousands of files are registered. The RRS also supports a set of error directives that are triggered in case of registration failures. This work is a first step toward standardizing the access of file and replica catalogs to allow interchangeability of such systems.

8 Acknowledgment

We want to thank Jean-Philippe Baud, James Casey, Ann Chervenak, Peter Kunszt, Reagan Moore, and Robert Schuler for their comments on the specification and possible extensions of the RRS interface. We also thank Viji Natarajan for providing the current implementation. This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

References

1. C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Research Broker. In *CASCON'98*, Toronto, Canada, 30 November - 3 December 1998.
2. J.-P. Baud and J. Casey. Evolution of LCG-2 Data Management. In *Computing in High Energy Physics*, Interlaken, Switzerland, September 2004.
3. A. Chervenak, E. Deelman, I. Foster, L. Guy, A. Iamnitchi, C. Kesselman, W. Hoschek, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Giggle: A Framework for Constructing Scalable Replica Location Services. In *Super Computing 2002*, Baltimore, USA, November 2002.
4. A. Chervenak, I. Foster, C. Kesselman, and C. Salisburyand S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 23, 2001.
5. PPDG Collaboration. Physics results from the STAR experiment at RHIC benefit from production Grid data services. http://www.ppdg.net/docs/oct04/ppdg-star-oct04.pdf.
6. P. Kunszt et al. EGEE gLite User's Guide - Overview of gLite Data Management. Technical Report EGEE-TECH-570643-v1.0, CERN, Geneva, Switzerland, March 2005.
7. P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. File-based Replica Management . *Future Generation Computer Systems*, 22(1), 2005.
8. R. Madduri, C. Hood, and W. Allcock. Reliable File Transfers in Grid Environments. In *27th IEEE Conference on Local Computer Networks*, Tampa, Florida, November 6 - 8 2002.
9. A. Shoshani, A. Sim, and J. Gu. Storage Resource Managers: Essential Components for the Grid. In *Grid Resource Management: State of the Art and Future Trends*, 2003. Edited by J. Nabrzyski, J. M. Schopf, J. Weglarz, Kluwer Academic Publishers.
10. A. Shoshani, A. Sim, and K. Stockinger. Replica Registration Service - Functional Interface Specification 1.0. Berkeley Lab, Berkeley, California, April 2005.
11. A. Sim, J. Gu, A. Shoshani, and V. Natarajan. DataMover: Robust Terabyte-Scale Multi-file Replication over Wide-Area Networks. In *Scientific and Statistical Database Management*, Santorini Island, Greece, June 2004.
12. The STAR Experiment. http://www.star.bnl.gov/.
13. H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney. File and Object Replication in Data Grids. *Journal of Cluster Computing*, 5(3), 2002.