Replica Management in Data Grids

Leanne Guy Peter Kunszt Erwin Laure
Heinz Stockinger Kurt Stockinger

CERN, European Organization for Nuclear Research
CH-1211 Geneva 23, Switzerland
{Leanne.Guy,Peter.Kunszt,Erwin.Laure}@cern.ch
{Heinz.Stockinger,Kurt.Stockinger}@cern.ch

July 1, 2002

Abstract

Providing fast, reliable and transparent access to data to all users within a community is one of the the most crucial functions of data management in a Grid environment. User communities are typically large and highly geographically distributed. The volume of data that they wish to access is of the order of petabytes and may also be distributed. It is infeasible for all users to access a single instance of all data.

One solution is that of data replication. Identical replicas of data are generated and stored at various globally distributed sites. Replication can reduce data access latency and increase the performance and robustness of distributed applications. The existence of multiple instances of data however introduces additional issues. Replicas must be kept consistent, they must be locatable, and their lifetime must be managed. These and other issues necessitate a high level system for replica management in Data Grids.

This paper describes the architecture and design of a Replica Management System, called *Reptor*, within the context of the EU Data Grid project. A prototype implementation is currently under development.

1 Introduction

Grid computing is essentially distributed computing over wide-area networks that often involves large scale resource sharing among collaborations of individuals or institutes. Computational Grids address computationally intensive applications that deal with complex and time consuming computational problems on relatively small data sets, whereas Data Grids address data intensive applications that deal with the evaluation and mining of large amounts of data in the terabyte and petabyte range.

The inherently distributed nature of grid technologies is attractive to modern user communities, which are characterised by their large size and geographically distributed nature. As described in [16], such user communities form Virtual Organisations (VO), in which they agree on a common set of conventions and policies for use of the available resources. Users are typically organised within their application domain; in High Energy Physics (HEP), for instance, several experiment collaborations exist and each collaboration forms a separate VO.

One of the principal goals of Data Grids is to provide transparent access to globally distributed data, making data access and location as easy as if on a local computer. Due to the distributed nature of the Grid, there are several issues that need to be addressed carefully in order to achieve this goal, the most important of which are:

- optimised access to data over the wide area to avoid large penalties on access performance,
- the need for a solid, highly extensible and well-performing security and access policy framework.

Optimisation of data access can be achieved via data replication, whereby identical copies of data are generated and stored at various globally distributed sites. This can reduce significantly data access latencies. However, dealing with replicas of files adds a number of issues not present if only a single file instance exists. Replicas must be kept consistent and up-to-date, their location must be stored in a catalogue, their lifetime needs to be managed, etc. We claim that replica management is an essential part of a Data Grid, as discussed in further detail in this paper.

Many underlying services required for replica management are currently under development; file transfer services and protocols such as GridFTP [1], replica catalogues [1, 23], and security mechanisms (GSI [1]). From the users' point of view, the lack of a common interface to these independent services makes them difficult to use and error prone. For example, the task of generating a new replica requires the application to utilise: wide area copy (e.g. GridFTP), catalogue services, metadata, security, and many more. Similarly, a replica catalogue provides the user with information about the location of all replicas of a file, but does not help in selecting a replica that requires minimal access time. These examples clearly indicate the need for higher level services to provide extended ease-of-use and additional high level functionality.

In this paper we discuss the design of a high level Replica Management Service that is intended to provide the application programmer with an intuitive interface that is easy to use and that hides the underlying details. We describe in detail our reference implementation, "Reptor", that has been designed and partly implemented in the context of the EU Data Grid (EDG) project [9]. We present some detailed use-cases from the application areas of the EDG project to validate our design. In conclusion, we discuss related work in this field.

2 Terms and Concepts

In this section we define terms and concepts that are used throughout this paper.

Replica A replica is an exact copy of a file that is linked to the original file through some well-defined mechanisms. Replicas of a given file may be synchronised to different levels of consistency, they may be catalogued, and their lifetime may be managed. We refer to these mechanisms as replica management functionalities.

Master Copy Consistency management is facilitated if one of the replicas is marked as a *master copy*, or *master replica*. The semantics of master copies may differ from VO to VO. Useful semantics include:

- Master copy is the only replica that may be modified.
- Master copy is the only replica from which new replicas may be created.

Both semantics may require multiple master copies in order to avoid a single point of failure. However, distributed locking is required to keep multiple masters synchronised. See consistency.

Secondary Copy If master copies are defined and exist, all other existing replicas are termed *secondary copies, or secondary replicas*. The semantics of secondary copies can be deduced from those of the master copies: all replicas that are not master copies.

Consistency Replicas usually adhere to a certain consistency scheme. If there is no consistency mechanism between a replica and master copy, then the replication of files is equivalent to normal file copying, with the possible subsequent modification of the replicas

Enforced consistency schemes are:

- *Consistent read-only replicas*
  Once a file is under the control of a replica manager it becomes read-only. Modifications to a file result in the generation of a new file and, possibly, a new replica. Consequently, all replicas are consistent.
- *Consistent read/write replicas*
  Replicas may be modified. This scheme is equivalent to a readers/writers synchronisation on a set of files; hence it requires appropriate locking mechanisms.

  Modifications need to be propagated to all the replicas of a file before the write lock can be released. This is true for a synchronous replication model where all replicas have the same values and updates need to lock all existing replicas. More relaxed replication schemes that use asynchronous update mechanisms allow certain replicas to be out of synchronisation for a certain amount of time. In the asynchronous replication model, write locks only need to be applied to certain replicas but has the consequence that not all replicas are always up-to-date. For more details on synchronous versus asynchronous replication refer to [8].
- *Versioning*
  Versioning is a combination of read-only and read/write replicas. Once a file is written, a new version of the file is created. Version information is stored in a meta-data catalogue such that either a particular version or the latest version may be retrieved.

Filename A replica manager deals with two kinds of Filenames:

- *Logical Filename(LFN)* is the name that refers to the full set of replicas for a file. Based on the LFN, all replicas can be looked up in a replica catalogue.
- *Site Filename (SFN)* is the name of a file as used by the storage management system. Thus, it may be a physical filename of a file stored on disk or a logical filename that is meaningful to a mass storage system. In the latter case the SFN is mapped to a *Storage Filename (StFN)* by the storage management system.

Computing Element A Computing Element (CE) is an abstraction for any *computer fabric*. It provides Grid users with CPU cycles for job execution, as well as an interface for job submission and control on top of services offered by the computing fabric. Each computing element is located at a particular "site" on the Grid. Computing Elements have well-defined interfaces to schedule and monitor jobs.

Storage Element A Storage Element (SE) is an abstraction for any storage system (e.g., a mass storage system or a disk pool). It provides Grid users with storage capacity. The amount of storage capacity available for Grid jobs varies over time depending on local storage management policies that are enforced on top of the Grid policies. Storage Elements have well-defined interfaces to store and retrieve files.

Resource Broker A Resource Broker (RB) is a high level tool (e.g. an *agent*) that schedules jobs on the Grid. For instance, a Resource Broker is responsible for scheduling a job on one or more Computing Elements. The resource broker may interact with the replica management system to optimise the scheduling of jobs and data access.

Metadata Metadata is data that describes and gives information about the content and properties of other data. In the context of replica management we deal with several kinds of metadata. Some of these concepts overlap:

- *Application metadata* are metadata that are specific to the application area, e.g. information about logical collections of data,
- *Filename metadata* are the physical file properties as reported by the Storage Element,
- *Logical filename metadata* are file properties not specific to any given physical file instance, but more generic to the concept of a logical file. These may include the overall lifetime of a file, provenance and security data (see below) or application metadata,
- *Provenance data* are data that describe the origin of the data, where it was produced, by whom and when,
- *Security data* are metadata associated with aspects of security such as authentication and authorisation.
- *Bookkeeping data* are metadata used for logging, job monitoring and accounting purposes.

Fault Tolerance Fault tolerance deals with the aspect of failures in a Grid and how to recover from them. Another aspect is to mask failures and automatically find alternative ways to solve a given problem.

Replica Catalogue A Replica Catalogue (RC) stores in a database mappings between logical filenames and site filenames.

Replica Location Service The Replica Location Service (RLS), of which the Replica Catalogue is a component, maintains information about the physical locations of logical identifiers and provides access to this information.

Authentication Authentication is the process of identification of users and services by other users and services. There are many authentication schemes that are used in a Grid environment.

Authorisation Authorisation is the process of allowing or denying already authenticated users and services to use certain resources, or to perform certain operations within another service.

3 Replication Management Service Design

In this section we give an overview of the design of our Replica Management Service called Reptor. We discuss the main components of our system and identify the functionalities and interdependencies of the components. Figure 1 presents the components of the Replica Management Service from the user's point of view. The design is general enough to be applicable to any RMS and not only to our reference implementation.

3.1 Replica Management Service

The Replica Management Service (RMS) is a logical single entry point for the user to the replica management system. It encapsulates the underlying systems and services and provides a uniform interface to the user. Users of the RMS may be application programmers that require access to certain files, as well as high level Grid tools such as scheduling agents, which use the RMS to acquire information for their resource optimisation task and file transfer execution. Although the RMS provides transparent access to underlying

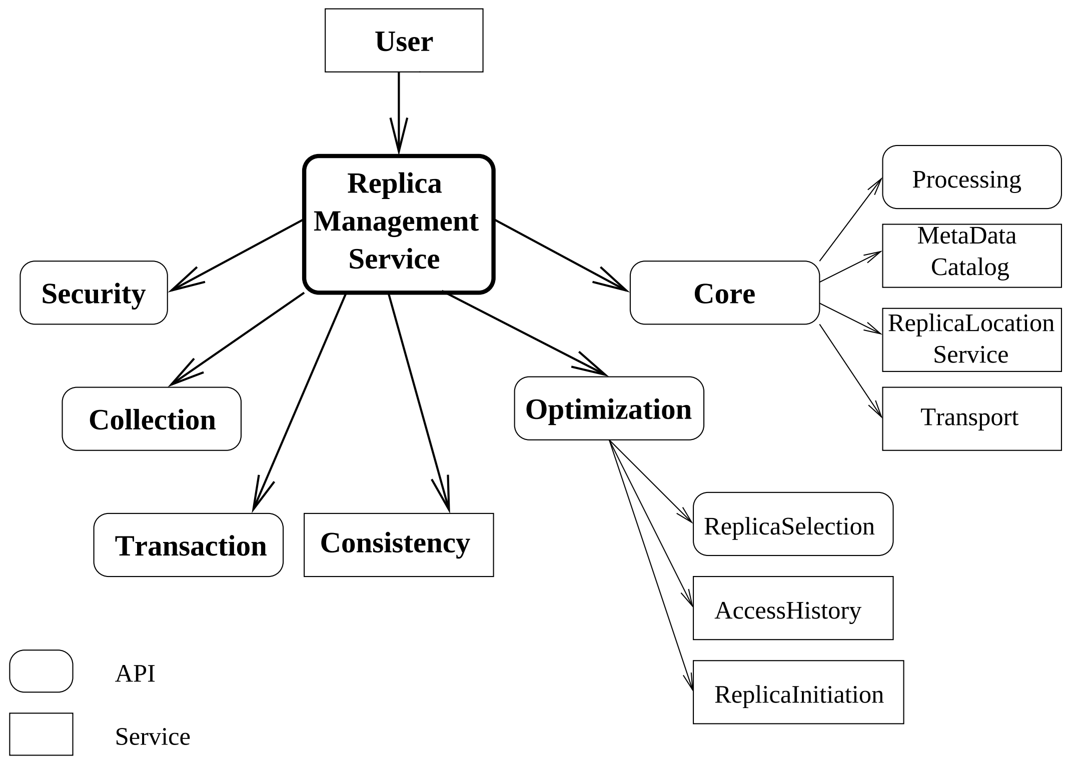

Figure 1: The main components of a Replica Management System from a user's perspective

systems and services, other tools may well require access to them, either directly or via a set of well defined interfaces. The RMS in this case acts as a proxy that delegates requests to the appropriate service.

The access mechanisms to the RMS must deal with the trade-off between ubiquitous access and high performance. To this end, we propose a multi-protocol strategy: an ubiquitously applicable SOAP [21] interface that would allow widespread publication of a generic interface to this service, while more specialised protocols such as RMI [25] would deliver the higher performance required. This is in line with the upcoming Open Grid Service Architecture [12, 26], that we will adopt in the first reference implementations of the RMS, Reptor.

The RMS also provides work-flow management functionality for VOs. It is able to schedule requests originating from within one VO according to some predefined priorities and should also be able to schedule independent activities within requests, such that the available resources are exploited optimally.

Logically, the RMS provides a single point of entry. For performance and fault tolerance reasons however, the RMS may be configured as a distributed service. In this case an RMS server being unavailable or overloaded could be replaced by an alternative RMS server. Of course, a distributed RMS service introduces consistency issues which are currently the subject of ongoing discussions.

The RMS has full control over all the files that have been created or registered through it. In other words, the RMS assumes a particular role on the Grid, allowing it to manage (copy, delete, modify) files that are under its control. Once a file is under RMS control, no one else is allowed to delete or modify the file. If the file is withdrawn from RMS control (e.g. by unregistering it) the original access control properties are restored. This also requires a close interaction with the underlying Storage Element that itself can create and delete files. We foresee certain file attributes (such as, permanent, volatile, durable) that influence the lifetime of the file within the Storage Element [2].

3.2 Core

The main functionalities of replica management, those of replica creation, deletion and cataloguing, are provided through the core API. The provision such services is implemented via the core API package that requires access to a set of services and APIs related to replica management:

- The *Transport* service provides the functionalities required for file transfer and may be implemented by means of different techniques, such as GridFTP.
- The *Processing* API allows the incorporation of pre- and post-processing steps before and after a file transfer respectively. It should be noted that processing in the context of file transfer is only allowed if the contents of the file are not altered. For instance, only loss-less compression techniques may be applied. An important design criterion is that the processing API be pluggable and be interchangeable for each Virtual Organisation or application domain.
- The core functionality also needs to keep related indices and catalogues up-to-date. In particular, access to the RLS, such as for instance *Giggle* [6] and the Metadata Service are required.

3.3 Consistency

The Consistency service performs two major tasks:

- it keeps the set of replicas of a file consistent, which is of particular importance if updates of replicas are allowed,
- it keeps the information stored in the replica location and metadata indices consistent with the physical files.

Inconsistent replicas, i.e. replicas that are not exact copies, may be due to

- unauthorised modification of a file,
- malicious or spurious modifications, e.g. attack or system failure.

A consistency service has to deal with both cases.

As mentioned in Section 2 and elaborated on in [8], a number of different consistency schemes are feasible. A Consistency Service should not impose any particular scheme on a VO, but be configurable in accordance with its specific consistency needs. Most forms of consistency management require some reference data with respect to which a consistency scheme is defined. In the context of replica management such reference data are *master copies*. One or more replicas of a replica set are assigned to be a master, all the others are *secondary copies*. Depending on the required consistency scheme, the consistency service may be responsible for any of the following tasks:

Update propagation In the case where a file is modified, these changes must be propagated to all existing replicas. Since a general update propagation scheme requires expensive distributed locking mechanisms in order to ensure that only one of the replicas is modified at any given time, and that no access occurs until the modification is propagated to all the replicas, some of the following restrictions might be imposed on replica modifications:

- Only a master copy is allowed to be modified. Before accessing a secondary copy an update with the master can occur thus removing the need for distributed locking. Of course, distributed locking is required among multiple masters.
- In versioning systems, a modification of a replica results in a new *version*. Version information is stored in the metadata catalogue. A read access might request a certain version or, as default value, the most recent version. In the latter case, the consistency service needs to determine the most recent version which is then provided. Depending on the VO policies, new versions might automatically be installed at all places where replicas of the old version exist. Alternatively, new versions might only be installed upon request. Efficient versioning can be achieved by specifying master copies as the only replicas from which versioning can be done.

Inconsistency detection The consistency service deals with inconsistencies due to attacks or system failures, and may or may not behave in a pro-active manner. More specifically, the consistency service checks whether:

1. the physical file still exists,
2. the file is still an exact replica.

These checks can efficiently be performed by means of checksum comparisons. In the case where inconsistencies are identified, the consistency service will remove the corresponding entries in both the Metadata and Resource Location Indices, and may also delete the inconsistent file.

Lifetime management Some replicas may not have a unlimited lifetime and hence may be deleted (garbage collected) after a *predefined lifetime* has expired, allowing the storage space to be reclaimed. This rather static scheme may be enhanced by a lifetime/subscription model: an application or user *subscribes* to a replica by registering an interest along with the lifetime of this interest. Subscriptions may be freely renewed and a replica may only be purged if no active subscription exists anymore. This scheme is equivalent to distributed garbage collection, e.g. applied in Java RMI.

3.4 Transaction

The Transaction API provides generic check-pointing and restart and rollback mechanisms that are needed by the work-flow management system to provide transactions.

One of the main aims of the transaction service is to add *fault tolerance* to the entire Replica Management Service. A transaction framework like the Java Transaction Service adds such a feature. No particular framework has been chosen yet but we will extend Reptor with software tools that follow existing standards in this area.

3.5 Collections

We define collections as a set of logical filenames and other collections, where a set is an unordered list. Collections are a very convenient way to handle datasets where the lowest granularity are files.

Semantically we define two different kinds of collections:

- **Confined collections.** Confined collections are collections where all elements of the collection are always kept together and are effectively treated as one - just like a tar or zip archive. Confined collections must be free of loops - i.e. it is not allowed to add the collection to itself or to have collections containing each other. The same element may not be added twice at the same layer, i.e. the list defining the collection must be free of duplicates. It is possible that the same LFN is effectively listed more than once in a collection by being member of the collection itself as well as sub-collections. Since all the information is available on confined collections when elements are added or removed, this semantic requirement is relatively easy to implement.

- **Free collections.** Free collections are composed of items not necessarily available on the same resource - they are effectively a bag of logical file and collection identifiers where we do not impose any semantic restrictions. Whereas confined collections assure the user that all the data are always accessible at a single data source, free collections provide a higher flexibility to freely add and remove items from the collection, but do not guarantee that all members are accessible or even valid at all times. Keeping free collections consistent with the constituent files requires additional services.

Confined collections are also catalogued in the replica location service because the necessary semantics exist to associate the collection with a location. Both collection types are catalogued in the Replica Metadata Catalogue service (RepMeC, cf. Section 3.7). The RMS provides a dedicated collection API that will store and retrieve information on collections from the RLS and RepMeC catalogues.

3.6 Security

The security API manages the required user authentication and authorisation, in particular, issues pertaining to whether a user is allowed to create, delete, read, and write a file.

The security layers used by the RMS are designed for maximal flexibility. Each VO should be able to enforce its own policies and each site should be able to impose their own security infrastructure.

3.6.1 Authentication

A user requesting a service from the RMS must be authenticated. Once authenticated the RMS is able to check the authorisation status for access to the requested service.

We do not expect the RMS to authenticate users on its own, rather, third party services will be leveraged. These services include grid authentication mechanisms such as GSI [1].

3.6.2 Authorisation

Once a user is authenticated, the RMS checks the authorisation status for access to the requested service based on identity and membership (VO, group, role) information. Users and applications will belong to a Virtual Organisation (VO), or a subgroup, thus inheriting the privileges of this VO or VO subgroup and assuming a certain role.

Authentication will also be handled by third party tools such as the Community Authorisation Service (CAS) [20]. Within the EU Data Grid project a Virtual Organisation Membership Service (VOMS) is currently under development, which will provide the above mentioned membership information embedded in the user's proxy certificate.

3.6.3 Delegated Rights

Following the authorisation process, the RMS acts on behalf of the user and performs all necessary tasks. This implies in particular that all administrative rights are delegated to the RMS. This delegation is necessary in order to enable automatic replication initiated by the optimisation module. Moreover, this scheme allows consistency enforcement since the only means to administer replicas is through the RMS; no user may incidentally delete or modify replicas and thereby cause inconsistencies.

It is worth noting that although the RMS assumes the administrative rights, a file may still be accessed by other means than via the RMS, for instance via a simple file open. The security system responsible for the file system will ensure proper authentication. Hence, read accesses are always possible, even if the RMS happens not to be available - unless the VO policies prevent such reads without authorisation from the RMS explicitly, as is the case in many Biomedical applications in the EU Data Grid.

The original administrative information is stored in the metadata service such that it may be restored once the file is removed from the Grid, and thus no longer under the RMS' control. The original administrative information may also be passed to other services, such as mass storage systems, that might require exact user information and cannot allow the RMS to act on behalf of the user.

3.7 Metadata

As mentioned in Section 2, there are many different kinds of Metadata that need to be stored such that they are accessible by the RMS. We foresee a dedicated replica metadata catalogue to provide this capability to the Data Grid and have a prototype implementation of such a catalogue, code-named RepMeC. We do not elaborate here on the details how the metadata is stored and distributed but rather try to categorise the different kinds of metadata needed by different components of the RMS, at different levels of service quality.

3.7.1 File Metadata

Basic information on the files that the RMS manages need to be stored persistently. This information may include, but is not restricted to:

Size Usually this is simply the size of the file in bytes. Note that the file size may be different on different platforms, depending on the block size of the underlying storage system. In addition, files may be compressed.

Checksum The file checksum and the function used to calculate it usually provide a better way to check for errors after file transfers.

Dates Creation date, last change date, replication date, all of these are sometimes necessary save to track the movement of files.

Type The file type can be simply ASCII or binary but we can have much more verbose descriptions like 'technical drawing' or 'FITS image'

Alias Files may have different names by which they are known. Aliases (which we define to have the same semantics as symbolic links in the Unix file system, see Section 3.10) are a well-defined way to handle different names for the same file.

3.7.2 Collection Metadata

Collections, as defined in Section 3.5, need to maintain information on their members. In addition to the information needed by files (that collections will need as well) we have

Elements This is a list of varying length that contains valid names of other LFNs and collections.

Type The file type as stated above is replaced by the collection type (free or confined).

3.7.3 Security Metadata

All information concerning security needs to be stored persistently in one way or another. Usually there are many different mechanisms in place to store and retrieve this kind of data but we usually also need to store local policies and additional authorisation data within the RMS framework. The non-exhaustive list for security metadata is

File ownership For files stored on Unix file-systems, there is a final mapping into a user and a group.

File access permissions In Unix file-systems, the file has a simple list of access permissions for owner, group and others.

File access control lists For more advanced storage systems, there are ACLs controlling fine-grained access to the files.

Group membership Depending on the security model, we need to store who is a member of which group. The Groups can then own groups of files. There are services external to the RMS that support this functionality, like CAS.

Local policies We also need to enforce local policies that need to be looked up in some local policy store or be stored directly within the RMS framework.

3.7.4 Application Metadata

Applications may define their own set of metadata for the data they access. This is highly application specific.

3.7.5 Management Metadata

Management metadata is every kind of data that needs to be accessed by the RMS in order that it perform its different functions, and that does not fall into the categories above.

File expiration time All files managed by the RMS have an expiration time that needs to be stored and managed.

Master copy The information on which copy (copies) is (are) the master.

Transactions Transactions have a lot of associated metadata. Locks, checkpoints, undo information all need to be kept depending on the implementation of the transaction mechanism.

3.8 Optimisation

In a Data Grid a user typically submits a *job* to the Grid and requires that the job be executed as fast as possible. For simplicity we consider a job as a request for a set of Logical Files Names (LFNs). In order to execute that job, three kinds of resources are required: computational resources, data resources, and network resources. Given the current status of the Grid resources (workload and features of Computing Elements, location of data, network load), the job should ideally be scheduled on a Computing Element such that the amount of required resources is minimised.

In this section we assume that the job has already been scheduled on a Computing Element that has a Storage Element attached to it. We refer to the attached Storage Element as the “close” Storage Element.

We will focus on the optimisation problems of replica selection and replica initiation rather than on the job scheduling problem.

The optimisation component consists of the following three services: replica selection, access history and replica initiation as described in the following subsections.

3.8.1 Replica Selection

The *Replica Selection* API requires the core service for translating LFNs to SFNs and the *Network Monitoring Service* for estimating the file transfer time. In particular, a single LFN may translate to multiple SFNs that are physically located on various Storage Elements all over the Grid. Based on the information provided by the network monitoring service, the replica selection API selects the LFN with the lowest estimated transfer time and calls the Transport Service to execute the file transfer to the close Storage Element.

3.8.2 Access History Service

The Access History Service keeps track of accesses to replicas. Statistics may include time series about requests for LFNs and users who initiated the file request. It is important to note that the Access History Service needs to distinguish between successful and unsuccessful file requests in order to avoid overestimation of file requests in case of failures.

The access history serves as the main input for dynamic replica creation (cf. Section 3.8.3). However, it can also be used for other purposes such a resource accounting [4, 5, 27].

3.8.3 Replica Initiation Service

The goal of the replica initiation service is to trigger replication dynamically. The decision about when to create new replicas can be based on the access history in order to optimise data locality for frequently requested files.

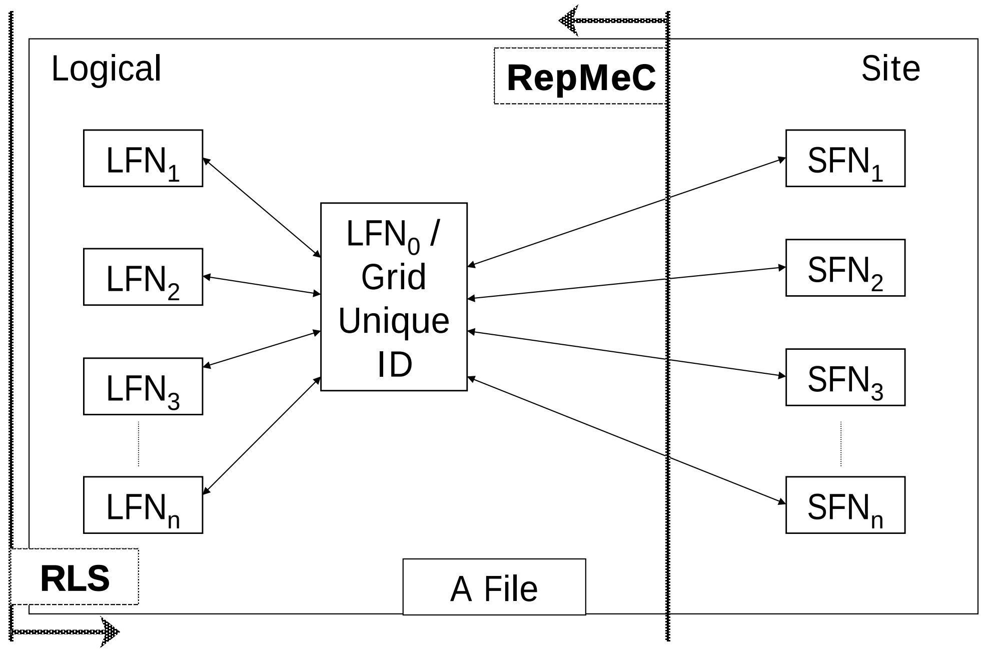

Figure 2: A single file can have many names - logical names and 'real' names on the storage site. This mapping information is kept in the RLS and RepMeC.

3.9 Communication

In the (quite likely) case that multiple RMS instances exist, these systems need to cooperate in an efficient and ubiquitous manner. As with the issue of user interaction, discussed in Section 3, we propose a multi-protocol strategy that also allows communication with other replica management services. A specific communication module may be provided that hides the multi-protocol issues and offers a common interface to all the RMS modules. These modules will specifically include the optimisation and transaction modules.

3.10 Globally Unique Identifier Generation

In order to locate and manage files and collections it is essential that these datasets are identifiable and trackable by the system. This necessitates a unique identifier that does not change over time. However, users of the RMS have different requirements: they want to freely assign names by which they can find their files. Different users might want to give different names to the same dataset. These are two clearly conflicting requirements but this conflict can be resolved because the Grid Services and the users may have different views of the same file.

3.10.1 The LFN Alias

As is shown in Figure 2, a single file may be identified by many different names. We require the file to have a globally unique Grid Unique Identifier (GUID). This identifier is assigned to each new dataset by the system. Users may provide a 'human readable' LFNs to the dataset which we treat as aliases to the GUID. Aliases have the same semantics as Unix symbolic links, in addition:

- Aliases may only be created for existing GUIDs.
- Upon deletion of an alias the dataset will not be removed.

- Aliases may be renamed.
- If the GUID is deleted the alias becomes invalid (dangling).

We have the additional requirement in the EU Data Grid project that the LFN aliases be unique. However this can not be assured globally at the time when the user defines the LFN because it would require a global lookup and subsequent lock while the file is being created.

At a given site it can be assured that the LFN is at least unique locally. But it may happen that two individuals at different sites assign the same LFN alias to a different GUID simultaneously. Nevertheless this is not a big problem since each LFN is tightly coupled to its GUID, so two conflicting LFNs can always be recognised as such and action may be taken upon detection of this conflict. The exact action that the replication services will take is customisable by the VOs running the service, there are several possibilities:

- Remove the LFN that has a later creation date.
- Remove the LFN with the earlier creation date.
- Remove both LFNs.
- Send an email to both LFN creators.

The VOs may provide some LFN naming scheme that will assure that within their VO the LFNs will indeed be unique. The ability to rename an LFN is a requirement from the application areas of the EDG project and will probably happen frequently, hence the need for a Grid-controlled GUID.

3.10.2 The GUID

The GUID is assigned at creation or registration time of a Grid dataset - this may be a file or a collection. The composition of a GUID is a combination of a VO namespace and the standard UUID scheme:

```
GUID://vo.virtual.host/UUID
```

where

- **vo.virtual.host** is a DNS-registered virtual host identifying the VO responsible for the dataset
- **UUID** is a long string based on the standard UUID generation scheme as described in [15].

The RLS will need to store the GUID of each LFN as an attribute. The GUIDs themselves of course are also stored in the RLS. Confined collections also receive a GUID at creation time, which is catalogued in the RLS. However it is RepMeC that stores the list of GUIDs that are referenced by collections (both free and confined collections). This has the advantage that if an LFN that is a member of a collection is renamed, the collection will still remain valid.

We foresee that the RMS provide a specific name generation API which allows the user to deal with these issues.

4 Case Studies

Typical usage scenarios of a Replica Management System (RMS) such as Reptor from the three application areas within the EU Data Grid (EDG) project, High Energy Physics (HEP), Biomedical, and Earth Observation Sciences, are presented here.

Grid users from all application areas will in most cases access data by specifying a unique logical identifier of the data, or of a dataset. For a file, the unique logical identifier is referred to as the “logical filename” (LFN). A dataset is defined as a read only collection of data and is typically implemented as one or more files, but may also be databases. A dataset’s unique logical identifier is referred to as the “logical dataset name”, (LDN). In the case where the logical identifier refers to an object, an object to file mapping stage is required, since our current replica management architecture is file based.

The RMS must determine how best to access a physical instance of the data. Possible options include:

- access via a remote protocol to an existing physical copy of the data,
- replication to another SE and subsequent access of this replica,
- copying to a temporary local area on the CE where the analysis job runs.

In all cases, the user or automated application that submits the job must have the required credentials for access to the data, update of any associated metadata catalogues, and must be authorised by the VO to submit jobs.

4.1 High Energy Physics Applications

High Energy Physics (HEP) data management requires very large amounts of both processing power and data storage. The four experiments of the Large Hadron Collider (LHC) will accumulate of the order of 5-8 petabytes of raw data per year. In addition, during the preparation phase prior to the start of LHC data taking, a similar order of magnitude of simulated data will be required to design and optimise the detectors. Each LHC experiment will form a single Virtual Organisation (VO), comprising of the order of 2000 scientists from over 50 countries. Constructing a single central site is obviously infeasible and the HEP community seeks to take advantage of the distributed nature of computing grids to provide physicists with the best possible access to both simulated and real LHC data, from their home institutes. Data replication and management is hence considered to be one of the most important aspects of HEP computing grids. The task of replicating LHC data to the various collaborating institutes within a VO will be handled by an RMS, such as Reptor.

A very comprehensive study of common use cases encompassing all four LHC experiments, for LHC applications to use Grid services has been produced [11]. One use case, pertaining to replica management, is the replication of a predefined collection or dataset to another Grid node.

Use case: Replicate a set of data to another Grid location.

- $t = t_0$: The user specifies a unique logical identifier for the dataset (LDN) to be replicated and the destination SE,
- $t = t_1$: The RMS queries the metadata service to obtain a list of all LFNs of which the LDN is comprised,
- $t = t_1$: The RMS gets a list of all extant replicas for each LFN in the dataset from the RLS,
- $t = t_2$: The RMS calculates the total space needed for the dataset and reserves sufficient space on the destination SE,
- $t = t_3$: The RMS determines the best replica to use based on the access cost. This is calculated using the replica optimisation component of the RMS which provides an estimate of the time to access a given SFN. Note that not all files comprising a dataset will necessarily be located on the same SE, or at the same site,
- $t = t_4$: The RMS copies all files in the dataset to the specified destination SE,
- $t = t_5$: The Replica Location Service (RLS) is updated with information about the new replicas,
- $t = t_5$: The user is notified of termination of job.

Figure 3 shows a sequence diagram for this use case. Errors may occur in the case where the destination SE does not have enough space for the dataset to be replicated. In such a case, the SE, in conjunction with the RMS, must decide how to proceed. Possible options include:

- the RMS returns an error stating that the job cannot be completed due to lack of resources on the specified destination SE,

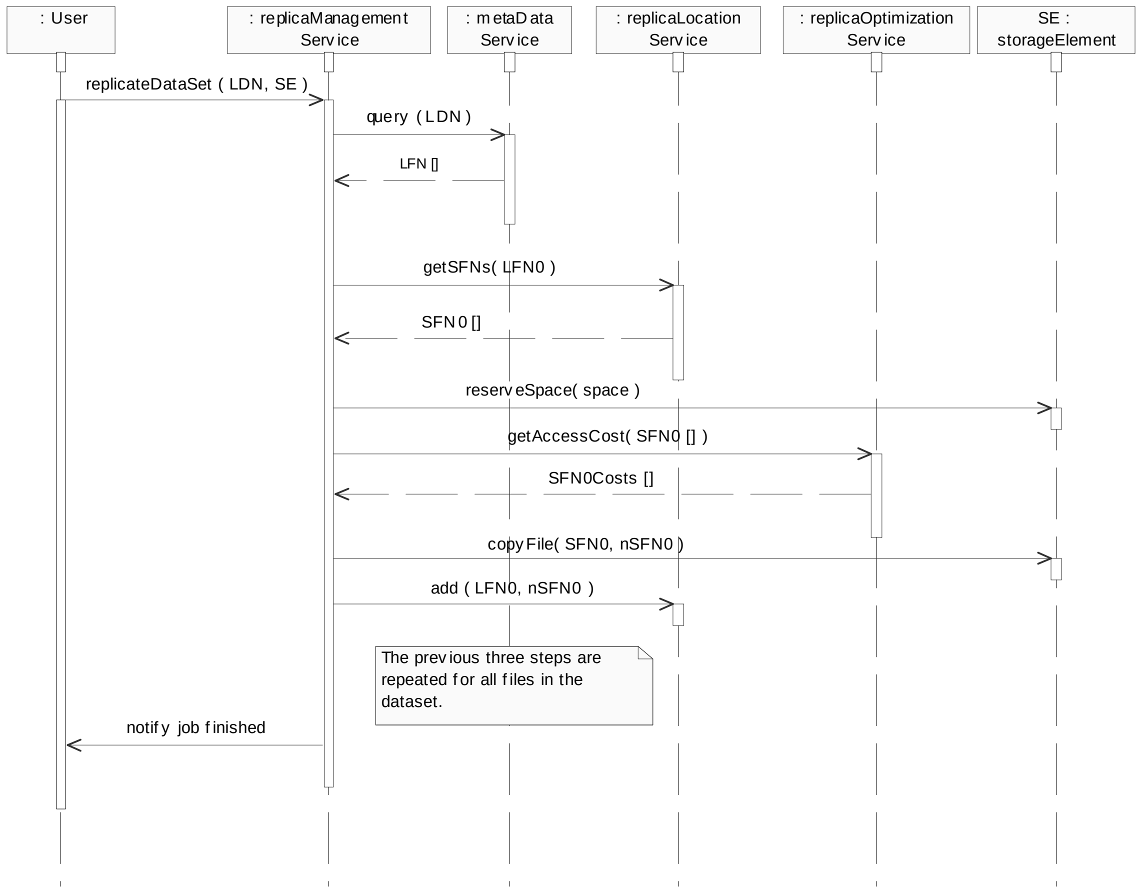

Figure 3: Sequence diagram showing the use case for dataset replication in High Energy Physics.

- files tagged as "volatile" on the SE are identified and deleted to release sufficient space for the replication request to proceed,
- an alternative SE is identified that satisfies the job requirements.

The replication of the dataset should be considered as an atomic transaction. If one or more files in the dataset are not accessible, then the job cannot be completed. Similarly, if the copy of one or more files in a dataset to the destination SE fails, then the job is considered to have failed. In such a case, a retry should take into account files in the dataset that were successfully replicated to avoid unnecessary waste of bandwidth. The replication of a single file is considered as a special case of the replication of a dataset, in which the dataset contains only one file.

4.2 Biomedical Applications

The Biomedical application area intends to use the Grid to facilitate the archiving of biological objects and medical images in distributed databases, collaborations between hospitals and medical institutes, and to provide distributed access to data.

Medical, biological and genomics data differ in nature from HEP data in that they are highly sensitive. Security is of the utmost importance to prevent unauthorised access to private data and hence has strong implications for the automated replication of data in a Grid environment. Not all data can be replicated to any Grid node in a VO. The Biomedical application area considers data to be basically of two different types; private or public. Medical images are stored in a format that contains a header part and an image part. The header part contains sensitive metadata, including patient details. To ensure a minimal level of confidentiality, the metadata must be stripped from the image prior to replication in a Grid environment. Encrypted image data can then be stored on standard grid nodes, whereas the associated metadata and mapping to the image file will be stored only at trusted sites. This will require the RMS to handle image and patient related metadata, to store additional security metadata for image access, as well as the encryption key for the image. A specialised SE has been defined with additional functionality that will allow it to: manage image metadata, encrypt images, log access to private data and interface with grid middleware, in particular, the RMS.

The growth in size of Biomedical databases is exponential and frequent updates are necessary to determine homologies with the addition of every new genome sequence. Databases can grow to several TB/year, with image sizes ranging from 20MB to 2GB. Public data banks, distributed by Bio-Informatics providers, will be replicated in a Grid environment using an RMS, such as Reptor.

Use case: Produce and register a new medical image with the Grid

- $t = t_0$: A new medical image is produced and stored on a private medical server,
- $t = t_1$: The server informs the medical SE that interfaces to the RMS that a new image is to be replicated on a Grid SE,
- $t = t_2$: Header information on the image is blanked by the medical SE,
- $t = t_3$: The medical image is encrypted by the medical SE,
- $t = t_4$: The SE registers the image and its metadata with the RMS,
- $t = t_5$: The RMS reserves sufficient space on the destination SE for the new image and copies it from the non grid SE to the grid SE.
- $t = t_6$: The RMS updates both the image and patient metadata information. The image and patient metadata will be stored on a trusted server and accessed via the RMS metadata interface. The same interface can be used to access the file related metadata, which may be stored on a grid node.

An interaction diagram for this use case is shown in Figure 4. Full details of other Biomedical application use cases for data management are described in [18]

Requirements concerning replication policy across sites within a Biomedical VO are more stringent than for other application areas, as illustrated in the following examples:

- An authorised and authenticated user submits a request to replicate a medical image on a Grid SE within the VO. However the site that hosts the specified SE may not be authorised to access this data. To RMS must first ascertain that the destination site is authorised to accept a replica of the data. This is not the case in other application areas where it is assumed that any site within a VO is authorised to receive replicas of data from any other site in the same VO.
- An authorised and authenticated user submits an explicit request to remove data from the grid. The replica manager will delete all entries for the logical identifier of this data from the RLS. It will then flag all physical instances of this data as “volatile”. To satisfy the Biomedical requirements, the RMS must then ensure that all physical instances of this data are explicitly removed. The removal of entries from the RLS and the removal of all physical instances of the data should be considered as an atomic transaction.

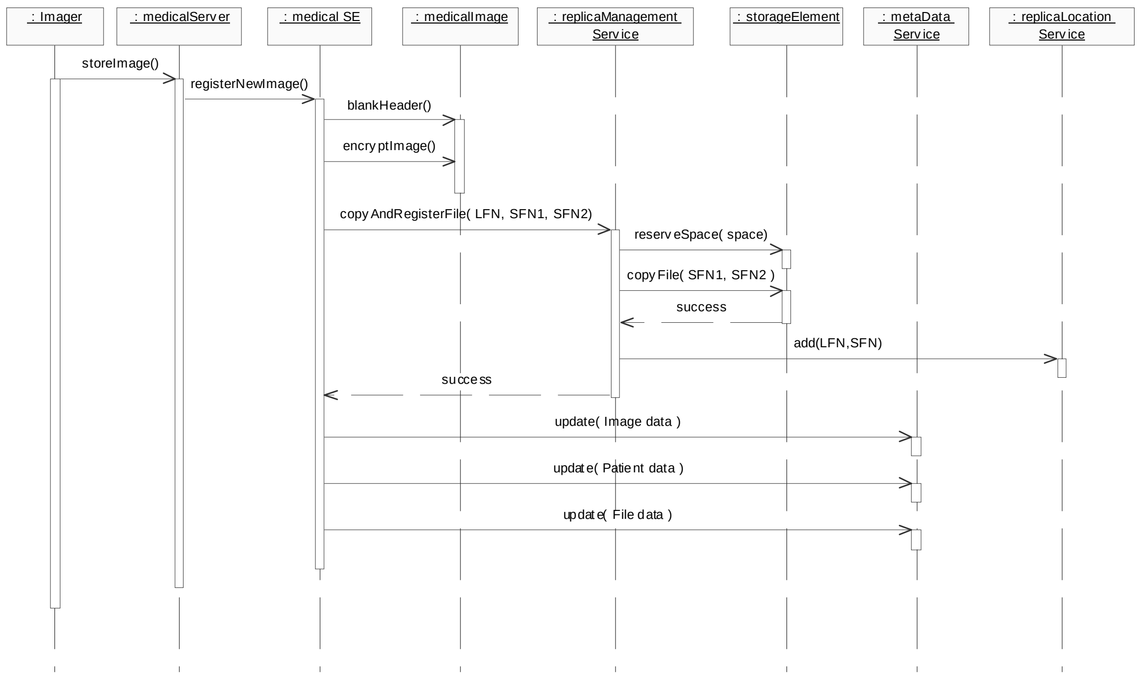

Figure 4: Sequence diagram showing the use case for Biomedical applications

4.3 Earth Observation Science Applications

The Earth Observation application area studies the nature of the planet's surface and atmosphere. One application of Grid technology is for the processing, archiving and validation of ozone profiles. In the following use case, it is the Resource Broker (RB) that interacts with the RMS and not the user directly.

Use case: User runs a job to analyse ozone profiles

- $t = t_0$: User submits a job to analyse ozone profiles,
- $t = t_1$: The RB determines the "best" CE from the list of candidate CEs with respect to physical locations of the data,
- $t = t_2$: The RB reserves space on an SE for the job's output data,
- $t = t_3$: The RB determines the best CE based upon the network costs returned by the RMS for each candidate CE and schedules the job on the best CE,
- $t = t_4$: The job queries the application metadata managed by the RMS to extract a list of LFNs that match the requirements,
- $t = t_5$: The job calls the RMS interface to determine the corresponding SFNs to access,
- $t = t_6$: The job reads and processes the data,
- $t = t_7$: The job writes the output dataset to the output SE,
- $t = t_8$: The job registers the output dataset with the RMS,
- $t = t_9$: The job updates the metadata service for the output dataset
- $t = t_{10}$: Job finishes, the RB informs the user.

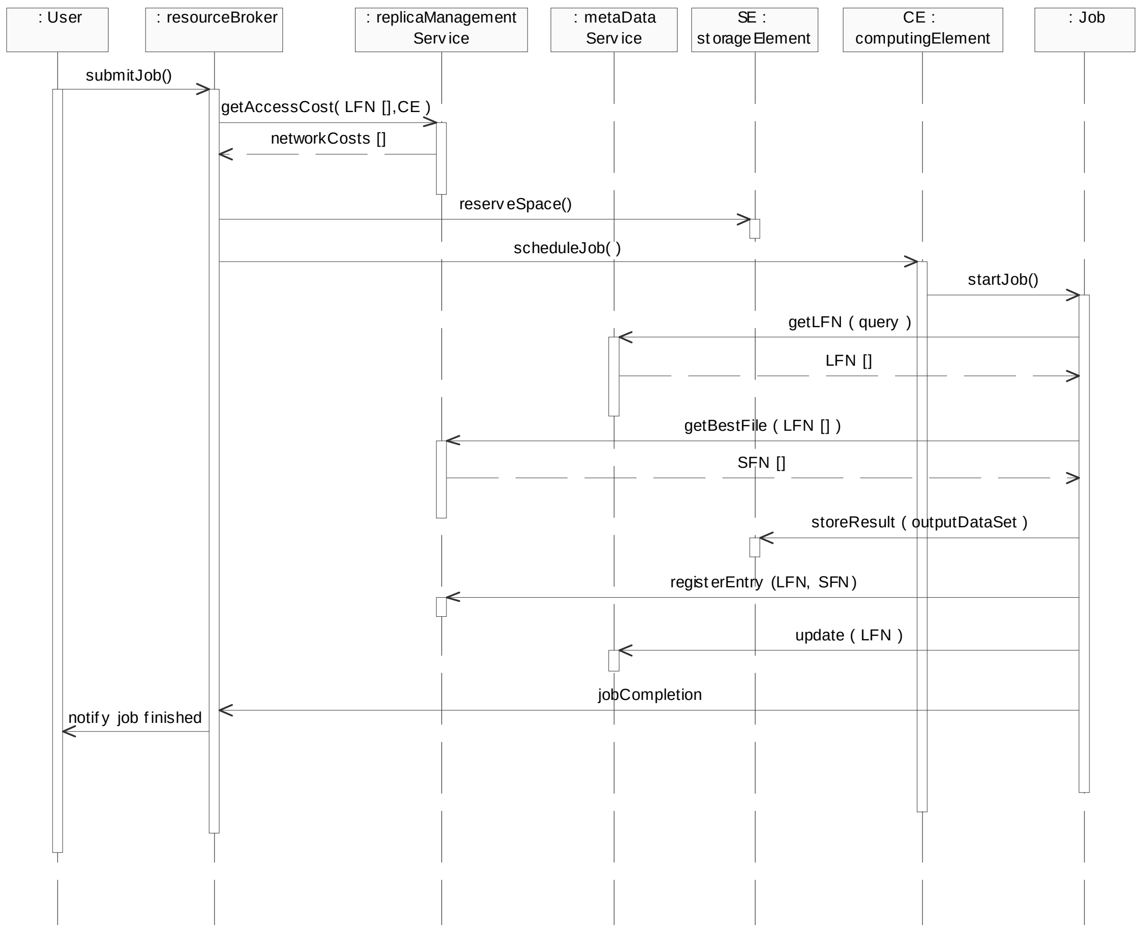

Figure 5: Sequence diagram showing the use case for Earth Sciences Applications

An interaction diagram for this use case is shown in Figure 5. This use case presents the basic generic use case for "Run job on Grid", and can be applied to any application area.

At this point in time, the Earth Observation Science application area places no additional requirements on data replication than those of HEP and Biomedical applications. Use cases for Earth Observation Science applications are described in more detail in [28].

5 Related Work

Within the Data Grid as well as in distributed computing communities, several projects deal with replica management related issues. Here, we first state related projects that deal explicitly with file replication and then discuss them in a broader scope.

Within the context of the EU Data Grid project as well as the Particle Physics Data Grid (PPDG) project, we developed the Grid Data Mirroring Package (GDMP) [24]. GDMP is a general file replication tool based on several Globus Data Grid tools such as GridFTP, a replica catalogue, GSI security etc. Mirroring of individual files as well as entire directory structures, a sub-scription based model, a preliminary mass storage system interface are among the main features of this client-server software system. The GDMP package has been used in the EU Data Grid project as well as in some High Energy Physics experiments in Europe and the U.S.A. For file mirroring use cases, the software fulfils its requirements, but Reptor will provide a more generic replica management system.

The most recent development in the EU Data Grid has been the edg-replica-manager [10] that makes partial use of the Globus replica managemLent [14] libraries for file replication. The edg-replica-manager implements parts of the core API that are detailed in Section 3.2 and can also be regarded as a prototype for Reptor. The edg-replica-manager is a client software tool only that relies on GridFTP servers on the server side and thus does not maintain any state about replication nor provide any of the high level functionality described in this document. The same is true for the Globus replica management libraries.

Within the Grid community, a strongly related projects is the Storage Resource Broker (SRB) [22], a client-server based middleware tool implemented to provide distributed clients with uniform access to different types of storage devices, diverse storage resources, and replicated data sets in a heterogeneous computing environment. The SRB supports automatic creation of data replicas by grouping two or more physical resources into a resource group or logical resource.

Related work can also be found in the file system domain, Of particular interest to us is InterMezzo [17], a distributed file system project that has its origin in Coda. InterMezzo focuses on high availability and is suitable for the replication of servers, mobile computing, and for managing system software on large clusters. InterMezzo is a client-server file system that maintains replicas of file system data across multiple server machines, it offers disconnected operation and automatic recovery from network outages.

Chord [7] is a flexible lookup primitive for peer-to-peer environments. The Chord application provides all desired authentication, caching, replication, and naming of data. For instance, data could be authenticated by storing it under a key derived from a cryptographic hash of the data. A file could be replicated by storing it under two distinct keys derived from the name of the file. However, Chord cannot be adapted as-is for several reasons. It is designed to use a set of resources with similar capabilities (memory, bandwidth, CPU speed) which is not the case on a Data Grid. There is no control over who is responsible for providing replica location information for a given file and in a highly dynamic environment, in which new files are inserted frequently, there may be a significant delay associated with inserting new keys into the system. This means that newly inserted files may not be immediately retrievable by the lookup service.

The OceanStore project [19] provides a global-scale persistent data store. OceanStore is designed to scale up to thousands of millions of users, and to provide a consistent, highly-available, and durable storage utility atop an infrastructure comprised of untrusted servers. Any computer can join, contribute or consume storage, and provide the user access in exchange for economic compensation. Many components of OceanStore are already functioning, but an integrated prototype is still currently being developed. However, OceanStore is based on the assumption that the resources are basically free and infinite. To apply some of these ideas to a controlled heterogeneous system is our challenge.

Freenet [13] is a distributed, anonymous, and adaptive peer-to-peer network information storage and retrieval system. Freenet is designed to respond adaptively to usage patterns, transparently moving, replicating, and deleting files as necessary. The files are dynamically and automatically replicated and deleted from locations depending on the usage. Hence Freenet forms an adaptive caching system based on usage and incorporating lazy replication. Freenet includes both replica management and replica location mechanisms: popular files are replicated closer to users, while the least popular files eventually disappear. Freenet's file location mechanism is built based on usage patterns, using dynamic routing tables. However, Freenet's assumption that non-popular data is unimportant data is not valid for many scientific applications.

6 Conclusion and Future Work

We have presented the design of a replica management service and in particular of our reference implementation, Reptor. Irrespective of our Reptor reference implementation, the replica management service design is a general one, applicable to any Data Grid where replication is a major aspect. This is supported by our flexible and modular design that can interface to existing Grid Services such as a replica catalogue service or a file transfer service.

The Reptor core API as well as the Optimisation Service (Optor) have already been partially implemented. Further APIs and services will be implemented and interfaced to within the next year within the scope of the EDG project as well as in the context of and in co-operation with other related Data Grid projects in Europe and the U.S.A.

One of the major tasks for the future is to interface Reptor to a Mass Storage System like the Storage

Resource Manager (SRM) [3]. Here, we will leverage existing work. A reference implementation in the EU Data Grid is expected soon.

Acknowledgements

The authors would like to thank our colleges within the EU Datagrid Project for fruitful discussions and input for our use cases. In addition we thank Ann Chervenak and Carl Kesselman from the Globus team for stimulating discussions on replica management and design considerations.

References

[1] B. Allcock, J. Bester, J. Bresnahan, et al. Efficient Data Transport and Replica Management for High-Performance Data-Intensice Computing. In *18th IEEE Symposium on Mass Storage Systems and 9th NASA Goddard Conference on Mass Storage Systems and Technologies*, San Diego, April 17-20 2001.

[2] I. Bird, B. Hess, A. Kowalski, et al. SRM Joing Functional Design—Summary of Recommendations. http://edms.cern.ch/document/333386.

[3] Ian Bird, Bryan Hess, Andy Kowalski, Don Petravick, Rich Wellner, Junmin Gu, Ekow Otoo, Alex Romosan, Alex Sim, Arie Shoshani, Wolfgang Hoschek, Peter Kunszt, Heinz Stockinger, Kurt Stockinger, Brian Tierney, and Jean-Philippe Baud. Srm joint functional design. Global Grid Forum Document, GGF4, Toronto, February, 2002.

[4] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic Models for Resource Management and Scheduling in Grid Computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 2002.

[5] M. Carman, F. Zini, L. Serafini, and K. Stockinger. Economy-Based Optimisation of File Access and Replication on a Data Grid. In *International Workshop on Agend based Cluster and Grid Computing at International Symposium on Cluster Computing and the Grid (CCGrid'2002)*, Berlin, Germany, May 2002. IEEE Computer Society Press.

[6] A. Chervenak, E. Deelman, I. Foster, L. Guy, A. Iamnitchi, C. Kesselmanand, W. Hoschek, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Giggle: A Framework for Constructing Scalable Replica Location Services. In *SC'2002*, Baltimore, USA, November 2002.

[7] The chord project. http://www.pdos.lcs.mit.edu/chord/.

[8] Dirk Düllmann, Wolfgang Hoschek, Javier Jean-Martinez, Asad Samar, Ben Segal, Heinz Stockinger, and Kurt Stockinger. Models for Replica Synchronisation and Consistency in a Data Grid. In *10thIEEE Symposium on High Performance and Distributed Computing (HPDC-10)*, San Francisco, California, August 7-9 2001.

[9] European datagrid project. www.eu-datagrid.org.

[10] edg-replica-manager 1.0. http://www.cern.ch/grid-data-management/edg-replica-manager/.

[11] F. Carminati, P. Cerello, C. Grandi, E. van Herwijnen, O. Smirnova, J. Templon. Common use cases for a HEP common application layer. Technical Report HEPCAL RTAG Report, LHC Computing Grid Project, 2002.

[12] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www.globus.org/research/papers.html#OGSA, January 2002.

[13] The freenet project. http://freenetproject.org/.

[14] Globus replica management. http://www.globus.org/datagrid/software.html.

[15] Andrew Hanushevsky Heinz Stockinger. HTTP Redirection for Replica Catalogue Lookups in Data Grids. . In *SAC2002*, Madrid, Spain, March 2002.

[16] I. Foster and C. Kesselman and S. Tuecke. The Anatomy of the Grid. *The International Journal of High Performance Computing Applications*, 15(3):200–222, Fall 2001.

[17] The intermezzo project. http://www.inter-mezzo.org/.

[18] J. Montagnat, H. Duque. Medical data storage and retrieval on the DataGrid. Technical Report DataGrid-10-TED-345148-0-1, The European DataGrid Project, 2002.

[19] The ocean store project. http://oceanstore.cs.berkeley.edu/.

[20] L. Pearlman, Von Welch, I. Foster, C. Kesselmand, and S. Tuecke. A Community Authorization Service for Group Collaboration. In *Policy 2002: IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, Monterey, California, USA, June 2002.

[21] Simple Object Access Protocol (SOAP). W3C Note, http://www.w3.org/TR/SOAP/.

[22] The storage research broker. http://www.npaci.edu/DICE/SRB/.

[23] H. Stockinger and A. Hanushevsky. HTTP Redirection for Replica Catalogue Lookups in Data Grids. In *ACM Symposium on Applied Computing (SAC2002)*, Madrid, Spain, March 10-14 2002.

[24] Heinz Stockinger, Asad Samar, Shazhad Muzaffar, and Flavia Donno. Grid data mirroring package (gdmp). *Journal of Scientific Programming*, to be published in 2002.

[25] SUN Microsystems. *Java Remote Method Invocation Specification*, 1998.

[26] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman. Grid Service Specification. http://www.globus.org/research/papers.html#GSSpec, February 2002.

[27] Work Package 1. An Accounting System for the DataGridProject. Technical Report DataGrid-01-TED-0115-3_0, The European DataGrid Project, 2002.

[28] Work Package 9. WP9.4 Use Case. Technical Report DataGrid-09-TED-0101-1_1, The European DataGrid Project, 2002.