

# DataGRID

## REPLICA MANAGEMENT WITH REPTOR

EDG REPLICA MANAGER

---

Document identifier:	<b>DataGrid-02-TED-030408</b>
EDMS id:	
Date:	April 8, 2003
Work package:	<b>WP2</b>
Partner(s):	<b>CERN</b>
Lead Partner:	<b>EDG</b>
Document status:	<b>Version 1.0.0</b>
Author(s):	Peter Kunszt, Erwin Laure, Heinz Stockinger, Kurt Stockinger
File:	<b>reptor</b>

---

**Abstract:** Data replication is one of the best known strategies to achieve high levels of availability and fault tolerance, as well as minimal access times for large, distributed user communities using a world-wide Data Grid. In certain scientific application domains the data volume can reach the order of several petabytes; in these domains data replication and access optimization play an important role in the manageability and usability of the Grid.

In this paper we present the design and implementation of our replica management Grid middleware providing a high-level user and application interface to replication services. It is designed to be extensible and evolvable so that user communities can adjust its detailed behavior according to their QoS requirements.

Our first implementation – Reptor – provides access to fast and secure transfer mechanisms, includes the management of data and associated meta-data and takes into account information provided by Grid monitoring tools. The design and feature-set of Reptor are based on our experience in data management in the EU DataGrid project and on the refined requirements of our user communities.

Our prototype implementation is based on the web service paradigm, in accord with the emerging Open Grid Services Architecture, and provides promising results.



---

## CONTENTS

<b>1. INTRODUCTION</b>	<b>3</b>
<b>2. DESIGN OF A REPLICA MANAGEMENT FRAMEWORK</b>	<b>3</b>
<b>3. PROTOTYPE IMPLEMENTATION</b>	<b>5</b>
3.1. WEB SERVICES FOR REPLICATION . . . . .	5
3.2. INTERACTION WITH EXTERNAL SERVICES . . . . .	6
3.3. THE REPTOR CLIENT . . . . .	6
3.4. THE OPTIMIZATION SERVICE . . . . .	7
<b>4. EXPERIMENTAL RESULTS</b>	<b>7</b>
4.1. TESTBED SETUP . . . . .	8
4.2. FILE REPLICATION . . . . .	8
<b>5. RELATED WORK</b>	<b>10</b>
<b>6. CONCLUSION AND FUTURE WORK</b>	<b>11</b>

## 1. INTRODUCTION

Grid computing addresses the issue of distributed computing over the wide-area network that involve large-scale resource sharing among collaborations of individuals or institutions. We distinguish between computational Grids and data Grids: Computational Grids address computationally intensive applications that deal with complex and time intensive computational problems usually on relatively small data sets whereas data Grids address the needs of data intensive applications that deal with the evaluation and mining of large amounts of data in the terabyte and petabyte range. Therefore, Grid technologies are especially attractive to large-scale, geographically distributed user communities. In this paper we address the most basic techniques used in data Grids to raise the availability, fail-safety and robustness of the Grid: data replication and access optimization.

One of the principal goals of data Grids is to provide easy-to-use, transparent access to globally distributed data and to abstract the complexities from the user and the applications. Essentially, it aims at making data access and location as easy as on a single computer. In order to achieve this goal, the most important issues that need to be addressed are:

- How to optimize access to data over the wide area to avoid large penalties on data access times.
- How to provide a solid, highly extensible and performing security and data access policy framework.

Optimization of data access can be achieved via data replication, whereby identical copies of data are generated and stored at various sites. This can significantly reduce data access latencies. However, dealing with replicas of files adds a number of problems not present when only a single file instance exists. Replicas must be kept consistent and up-to-date, their location must be stored in a catalog, their lifetime needs to be managed, etc.

Many underlying services required for replica management have been developed: file transfer services and protocols (GridFTP) [2], replica catalogs [2, 16], and security mechanisms (GSI [13]). Additional services are under development: data access and integration services, replica location services [7], etc.

There is a clear need for higher level services that abstract all the intricacies of the basic Grid services from the users. As an example, consider the task of generating a new replica. This requires the application to perform a wide area transfer (e.g. using GridFTP) and to update various Grid catalogs while at the same time checking access rights and dealing with errors and failures.

Another example is replica selection: without a higher level Grid service to call on, the users will need to implement their own replica selection algorithms, increasing the burden on the application programmers.

In this paper we present the design and implementation of a replica management service that is intended to provide the application programmer with an easy-to-use, intuitive interface, hiding the details of the underlying services. We have designed and implemented a prototype, “Reptor”, in the context of the EU DataGrid project (EDG) [9]. Experimental work shows promising results.

The remainder of this paper is organized as follows: we present the design of our replica management framework in Section 2. and its prototype implementation in Section 3.. Initial results obtained with this prototype implementation are reported in Section 4.. After a discussion of related work (Section 5.) we end this paper with a brief summary and an outlook on future work.

## 2. DESIGN OF A REPLICA MANAGEMENT FRAMEWORK

In [14] we presented the design of a general replica management framework. In that model, a replica management service contains all the logic to coordinate the underlying services, providing the user with a

unified interface to all replica management functionalities (Figure 1). In our current implementation, the logic of the coordinating replica management service is implemented on the client-side, i.e. there is no actual service associated with it; this is to defer the issues we face with the details of authorization with respect to delegation of rights to services. Because computing fabric nodes usually do not have outbound (WAN) connections, we will have to solve this issue and provide a service that acts as a proxy to those clients. Having a service executing the method logic is also necessary to provide advanced optimization metrics (logging usage patterns for many users) as well as to increase fault tolerance (asynchronous execution and automatic re-tries) and performance (caching).

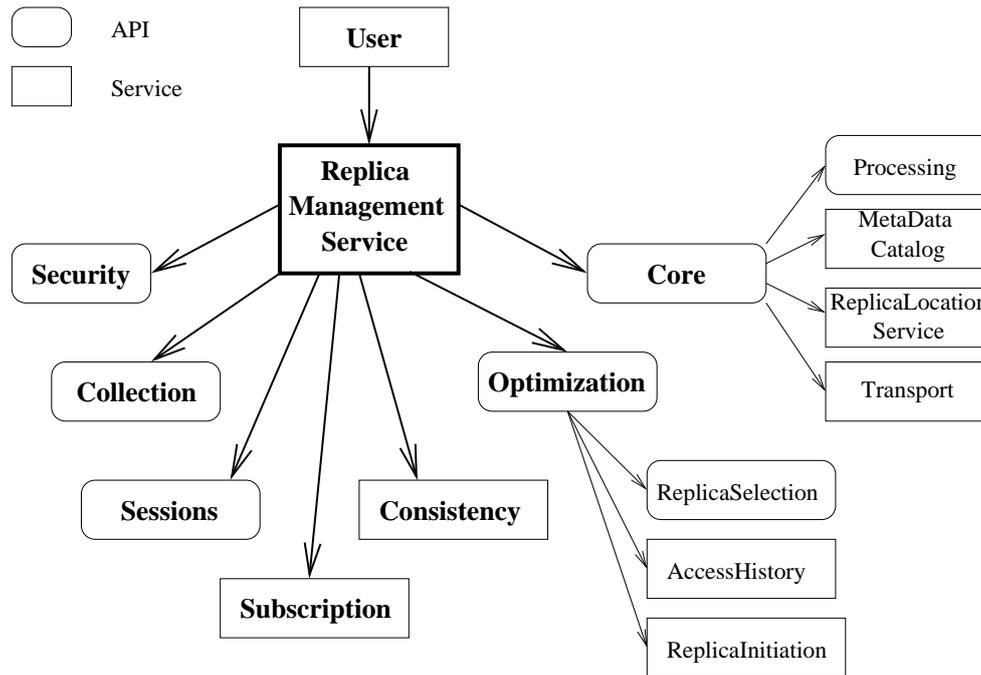
Figure 1 presents the user's perspective of the logical layout of the components of our replica management system [14]. The entry point to all services is the **Replica Management Service** which interacts with the subcomponents of the system.

- The **Core** module coordinates the main functionality of replica management, which is replica creation, deletion, and cataloging by interacting with third party modules. These external modules include transport services, replica location services, meta-data services for storing replication meta-data such as file meta-data (size, checksum, etc), management meta-data, and security meta-data (such as access control lists), and processing services that allow pre- and post-processing of files being replicated.
- The goal of the **Optimization** component is to minimize file access times by pointing access requests to appropriate replicas and pro-actively replicating frequently used files based on access statistics gathered.
- The **Consistency** module takes care of keeping the set of replicas of a file consistent as well as the meta information stored in various catalogs.
- The **Subscription** module takes care of subscription-based replication where data appearing at a data source is automatically replicated to subscribed sites.
- The **Session Management** component provides generic check-pointing, restart, and rollback mechanisms to add fault tolerance to the system.
- **Collections** are defined as sets of logical filenames and other collections.
- The **Security** module manages the required user authentication and authorization, in particular, issues pertaining to whether a user is allowed to create, delete, read, and write a file.

As mentioned before, we decided to implement the Replica Management Service and the core module functionality on the client side in the Replica Manager Client. The other subservices and APIs are modules and services in their own right, allowing for a multitude of deployment scenarios in a distributed environment.

One advantage of such a design is that if a subservice is unavailable, the Replica Manager Client can still provide all the functionality that does not make use of that particular service. Also, critical service components may have more than one instance to provide a higher level of availability and to avoid service bottlenecks. We foresee to provide consistent replicas for stateful services; services like the Replica Location Service are already designed to be distributed in their own right [7].

The model of providing high-level functionality by coordinating a set of underlying services is one of the paradigms of Grid computing. By following this paradigm we allow for deployments providing customized levels of Quality of Service as we foresee that Virtual Organizations will use the Replica Management Services in very different ways.



**Figure 1: Main Components**

### 3. PROTOTYPE IMPLEMENTATION

We implemented a first prototype of the replica management system, named “Reptor”, within the framework of the EU DataGrid (EDG) project. The services are all implemented in the Java language. The servers are deployed as web services. We depend only on open source technologies like the Jakarta Tomcat [20] servlet container, the MySQL database and Apache Axis.

The prototype implementation provides the core module interfacing to the transport and catalog services as well as on the optimization component, leaving the collection, session management, and consistency modules for future versions.

Security is provided within the Grid Security Infrastructure (GSI) framework [13]. Our web services are able to authenticate Grid certificates via our trust manager, an authentication module which can be plugged into web service containers. We also have a fine grained authorization mechanism in place that can interpret certificate extensions provided by the Virtual Organization Membership Service VOMS [1]. The advantage of such an approach is that our security infrastructure is backward compatible with the existing GSI implementations that simply ignore these extensions.

Reptor currently provides a Java API and Java-based command line interface. C and C++ client APIs are also available for parts of the interfaces and it is straightforward to generate bindings for other languages as well due to the web service technologies used.

In this section we first motivate the use of web service technologies as our technology paradigm, discuss the mechanisms used to access third party components from within Reptor, and provide details about the implementation of the client.

#### 3.1. WEB SERVICES FOR REPLICATION

In wide area distributed computing, web service technologies [22] are becoming increasingly popular since they provide easy and standardized access to distributed services in a multi-lingual, multi-domain environment. The same is true in Grid computing where the upcoming OGSA standard [15] aims at

leveraging web services in the Grid context. Due to the general recognition of the web service standard and to be prepared to move to the OGSA standard, we have chosen to adopt the web services paradigm.

The control and data channels between the services and their clients is managed through interfaces published in WSDL. For the control channel we use SOAP over HTTPS. For data channels that require higher levels of performance, we rely on the existing mechanisms like GridFTP [2] for file transport.

We have found that the available technologies to build standard web service components already address many of the issues of Grid computing and we could profit from the extensive code base made available by the web service community.

### 3.2. INTERACTION WITH EXTERNAL SERVICES

As described above, Reptor needs to interact with many Grid services such as the replica location service and the information services. We have implemented Reptor as set of modules that is easy to extend and interface to other Grid components.

We have defined interfaces for each Grid service that Reptor is accessing. In the spirit of general software engineering principles as well as OGSA, we have made all these modules pluggable: if one of the services needs to be replaced by another service providing the same functionality, the appropriate implementation of the interface needs to be provided to Reptor, which can be configured to load the given implementation instead of the default ones.

Currently, Reptor has been tested with the following Grid services

- *Replica Location Service (RLS)* [7] as the replica catalog service: used for locating replicas by their unique identifiers (GUIDs).
- *Replica Meta-data Catalog (RMC)* stores the user-definable logical file name aliases to GUIDs and trivial meta data on replicas, e.g. the owner, file size, and time stamps.
- *Relational Grid Monitoring Architecture (R-GMA)* [12] as the information service: used for obtaining information about the topology of the Grid.
- The *EDG Network Monitoring Services* [5] providing statistics about network characteristics.
- The *EDG Storage Element Services* [5] providing information about the storage latency.

Some of these services are currently client/server architectures using proprietary protocols for communication, but most of them will in the near future turn into proper web services, advertising their interface through WSDL.

### 3.3. THE REPTOR CLIENT

We provide a command line interface and an API for the clients. The client is a thin layer that interacts with the internal and external services, acting as the coordinator of the replica management sub-services.

We have a pure Java version of the client service using the Commodity Grid for Java *CoG* [21] that provides GridFTP client functionality. However, in order to be able to support parallel streams for better performance, we can also configure the Reptor client to use the native GridFTP C libraries where available.

Currently, the client provides the following functionalities:

- *copy* a file between two sites using third-party transfer [2] if required.

- *register* an existing file in the replica location service to allow its retrieval by other Grid services. A GUID is assigned to the file and registered in the meta-data catalog. It is used as the immutable unique identifier of the file and its replicas. Since GUIDs are typically not human-readable, the user may provide a logical identifier on its own which acts as an alias of the GUID. It must be unique as well, but unlike the GUID, it can change in time.
- *create replicas* of a file that has previously been registered in the replica location service. The optimization service discussed below can be exploited for selecting an existing replica to be copied to the destination location.
- *delete* a replica which results in the physical deletion of the file and its removal from the associated catalogs.
- *unregister* a file from the catalog without physically deleting the file.
- *list* all the replicas that have been registered by a given identifier (GUID), or look up the logical names or GUID based on a replica.
- *select* the “best” replica to be used on a given site with the help of the optimization service (see below).
- retrieve the *access costs* for all available replicas of a given file with respect to specified sites.

### 3.4. THE OPTIMIZATION SERVICE

The goal of the optimization service is to select the best replica with respect to network and storage access latencies. In other words, if for a given file several replicas exist, the optimization service determines the replica that should be accessed from a given location. Similarly, the optimization service might also be used to determine the best location for new replicas. We currently do not take into account an eventual possibility to access the data directly over the wide area — this is the subject of ongoing work — but assume that in order to access a file it has to be available in the local area network.

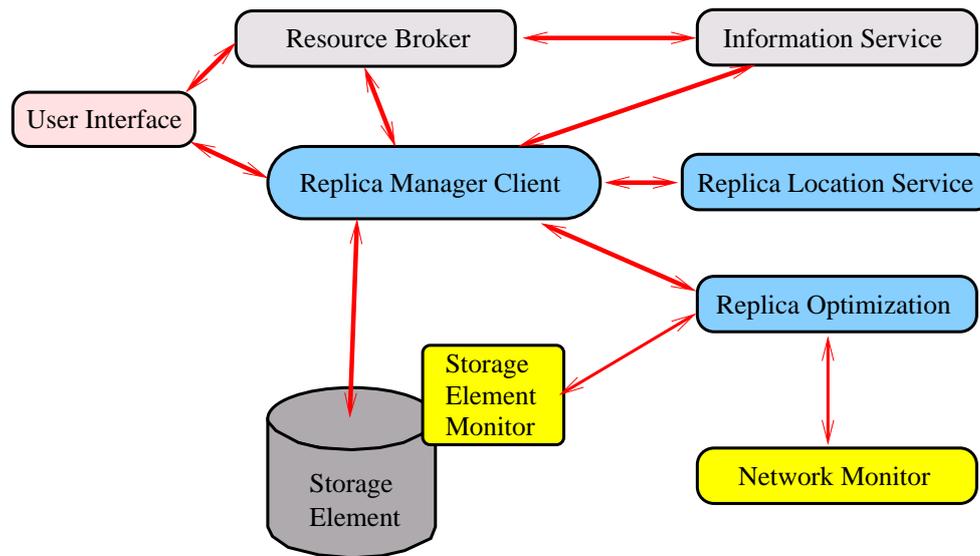
The *Replica Optimization Service (ROS)* is implemented as a light-weight web service (called *Optor*). It gathers information from the EDG network monitoring service and the EDG storage element monitoring service about their network and storage access latencies. Based on this information *Optor* takes a decision which network link should be used in order to minimize the transfer time between two end points as described in [5].

Apart from selecting replicas and storage locations *Optor* also provides methods to retrieve estimated file access costs. These can be exploited by other Grid services, such as meta schedulers like the EDG Resource Broker [10]. Based on the information obtained by *Optor* the broker can schedule jobs to sites that allow efficient file access while maximizing the overall throughput. Thus, the Replica Manager, in particular its optimization component, assists the Resource Broker in the job scheduling process.

The interaction of the Grid components in our current experimental setup is shown in Figure 2.

## 4. EXPERIMENTAL RESULTS

In this section we report on the results of tests carried out with *Reptor* to perform optimized replication of files. In particular we discuss the performance of the core functionalities and the optimization component.



**Figure 2:** Interaction of Replica Manager with other Grid Components.

#### 4.1. TESTBED SETUP

All our experiments were run on five major sites of the EU DataGrid testbed in France, Italy, the Netherlands, Switzerland and the United Kingdom (see Figure 3). Each site consists of *Computing Elements* (CE) and *Storage Elements* (SE). A Computing Element runs a (Globus) gate-keeper and a local scheduling system like PBS or Maui. A Storage Element is a generic Grid interface to storage; i.e. it can be a single disk, a disk farm or a mass storage system. A typical Grid job is submitted from a *User Interface* to the Resource Broker and is scheduled to be run at a Computing Element based on information provided by the Information Services and the Replica Manager (see Figure 2).

The components needed by the of Replica Manager Client were deployed at dedicated hardware at CERN, each machine running a single service only. We had one machine each to run the User Interface (i.e. the Replica Manager Client), the Replica Location Service, the Replica Optimization Service and the Information Service (R-GMA in our case [12]).

The network monitoring components have been deployed at each testbed site by the EDG networking work package. We make use of their network monitoring information provider which is accessible at IN2P3 in Lyon. The network monitors keep track of the performance of the network over time (see Figure 4 for an example). The monitors show that there are very large deviations in network performance over time. The spikes and dips in these network metrics usually span tens of minutes and hours — this motivates our strategy of optimization based on recent network metrics.

Table 1 shows the commands of Reptor that were used in our tests.

#### 4.2. FILE REPLICATION

In this section we provide performance numbers for replicating files of various sizes over a wide-area network including updating the replica catalog. We compare these numbers with raw data transfers achieved via GridFTP. Note that Reptor adds some overhead to the basic file transfer due to various consistency checks: Reptor checks whether the file already exists on the destination before starting the copy, obtains file meta-data, and checks the file existence after having performed the copy. This overhead is constant, independent of the actual file size.



**Figure 3:** Major testbed sites of the EU DataGrid Project.

Reptor provides two different replication options, *conventional replication* and *optimized replication*. The conventional option takes source and destination SEs as input parameters. The optimized option takes a logical file name (LFN) and the destination SE as input parameters and chooses the best replica as described above: For a given LFN, all physical replicas are identified through the RLS and for each replica the transfer costs to the specified destination SE are calculated using the Network Monitor. The replica having the minimal cost is used as the source file for replication.

By having both options, users may either control the source and destination of the copy themselves or let the system decide on the best resources to use on their behalf.

In Figure 5 we show our measurements for replicating files between three sites within the EU DataGrid testbed. The file sizes range from 100 MB to 1 GB. We use GridFTP as the file transfer mechanism with 8 parallel streams.

For replicating 1 GB files we can observe an overhead of around 10% due to the additional consistency checks.

The next experiment shows the possible performance gain one can obtain by optimized replication (see Figure 5 d)). Due to the different network bandwidths within the testbed and their rapid change in time (Figure 4) we gain factors of two and more in transfer efficiency by automatically choosing the best network link for file replication. If the users have to specify the source for the copy by themselves, they can achieve at best the same result. Without an automatic replica selection mechanism they most likely will choose a suboptimal network link. In both cases, the consistency overhead can be neglected since it is identical for both replication scenarios.

To sum up, we demonstrated network optimized replication with Reptor and achieved a substantial gain in efficiency especially in a distributed Grid environment where the network latencies are subject to rapid change. We have found that the changes are slow enough so that recent network monitoring information is usually still applicable at the time of the file transfer.

Command	Description
registerFile	register file with the Replica Location Service (RLS)
listReplicas	list all replicas registered with RLS
copyAndRegisterFile	copy file to SE and register with RLS
replicateFile	replicate file to SE and register with RLS
listBestFile	return replica with minimal transfer time no replication is involved
unregisterFile	unregister file from RLS
deleteFile	unregister file from RLS and delete from SE
getBestFile	return replica with minimal transfer time this command might involve replication
getAccessCost	get access cost of replicas for specific destination according to the network cost function

**Table 1:** Main Replica Manager Commands used during Experiment.

## 5. RELATED WORK

The design and implementation of the replica management framework presented in this paper is based on years of experience with preceding data management and replication middleware that we briefly describe in this section next to other relevant work on replication in the Grid community.

Early prototype work has been performed with a data replication tool named *Grid Data Mirroring Package (GDMP)* [18]. It is a basic file replication tool with support for mass storage system staging and automatic replication through a publish-subscribe notification system. Similar to Reptor, it is based on the Globus toolkit including GridFTP and GSI security. It has been in use in production for High Energy Physics experiments in Europe and the U.S. as well in the European DataGrid testbed. Reptor is meant to replace GDMP and was designed based on the user feedback on GDMP and on our better understanding of replication in a distributed Grid environment.

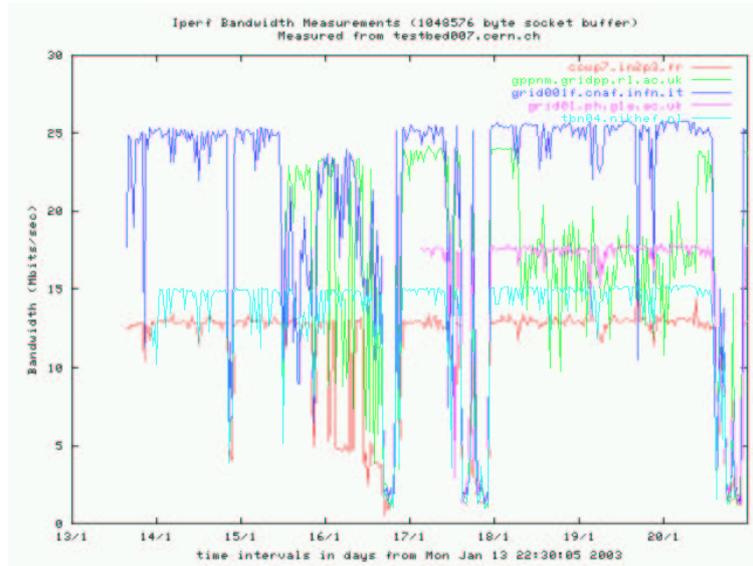
The Globus project provides in its toolkit a replica catalog based on the LDAP protocol and a straightforward replica manager that can manage the copy and registration as a single step. We have been using and extending these modules within the EU DataGrid project by a user-friendly API and mass storage bindings. The *edg-replica-manager* [11], can be regarded as the direct predecessor of Reptor. However, neither the Globus replica manager, nor *edg-replica-manager* nor GDMP provide replica optimization or any other of the capabilities that we foresee for the immediate future (see Figure 1).

An integrated approach for data and meta-data management is provided in the Storage Resource Broker (SRB) [4]. Data cataloging and access to relational and object-oriented databases is provided. Replica optimization is not provided yet but SRB might profit from approaches like ours.

The most related work with respect to replica access optimization can be found in the Earth Since Grid (EDG) [3] project where preliminary replica selection has been demonstrated using the Network Weather Service (NWS) [23]. Rather than providing an integrated replication framework with consistency checks etc., GridFTP transfer sources and destinations are selected via NWS.

Within the Grid and High-Energy Physics community, one of the most closely related projects is SAM [19] (Sequential data Access via Metadata) that was initially designed to handle data management issues of the D0 experiment at Fermilab. It also addresses replication job submission issues.

Finally, since file replication often deals with large storage systems, local storage resource management becomes vital and we participate in the standardization effort of the Storage Resource Management (SRM) [6] specification that builds the natural link between Globus, Grid-wide replica management and



**Figure 4:** Network bandwidth from CERN to other testbed sites. Data taken from iperf measurements in January 2003.

local file management.

In terms of technology we feel that our approach is the most extensible one in terms of functionality and is best geared towards the emerging Open Grid Services Architecture.

## 6. CONCLUSION AND FUTURE WORK

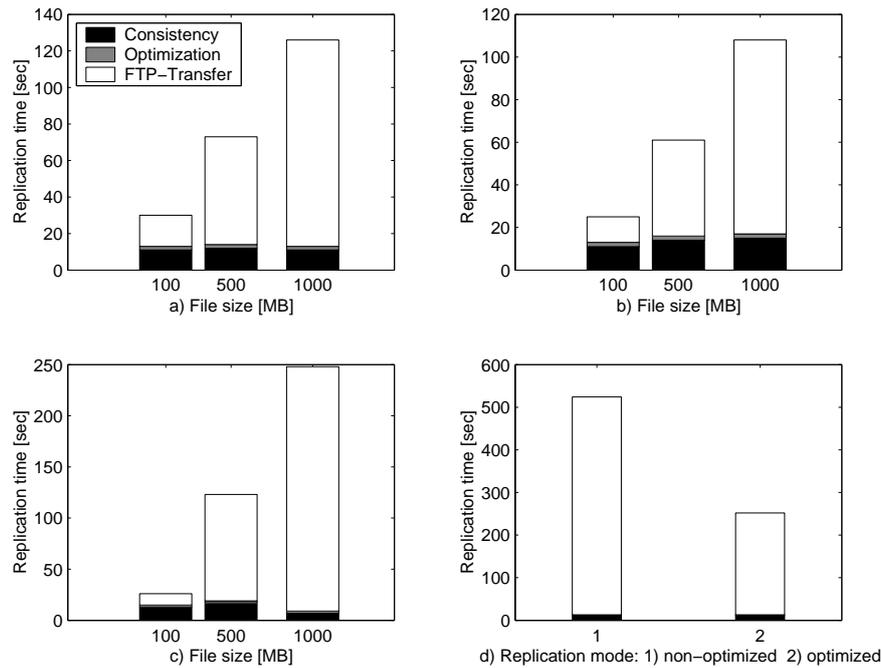
We have presented the design and a prototype implementation of a high level replica management tool providing the functionality required for efficient data access in a Grid environment. We have discussed in detail our optimized replica selection mechanism that takes into account network monitoring data in order to identify the optimal network link with respect to data transfer. The experimental results show that our approach significantly reduces wide area transfer times.

The framework is easily extensible and allows for many different deployment scenarios to serve the needs of many different Virtual Organizations.

Currently we have implemented the core and optimization functionality of the replica management design. In the immediate future we will extend the functionality by adding the first versions of the collection and consistency modules to Reptor. The consistency module is based on some of our previous ideas [8] and will contain a file versioning and update mechanism based on file replacement or binary difference between two files [17].

Future work also includes the refining and exact definition of the semantics for the session management module.

The security framework in the current Reptor prototype provides authentication based on GSI. In the immediate future it will also allow for fine-grained authorization to be achieved through a role based authentication/authorization management system [1].



**Figure 5:** Performance measurements of replicating files with sizes 100MB, 500MB and 1GB between a) CERN and RAL, b) RAL and NIKHEF, and c) NIKHEF and CNAF. d) Performance measurements of replicating 1GB files to NIKHEF 1) using conventional replication: IN2P3 was specified as the source and 2) using optimized replication: RAL was picked as the better source automatically.

## ACKNOWLEDGMENTS

We would like to thank first and foremost our colleagues in WP2 who have had contributions to the design and development of Reptor and for running the experimental testbed: Diana Bosio, James Casey and Leanne Guy. We would like to thank our colleagues in the EU DataGrid project who provided us with invaluable feedback on our existing replica management tools. Special thanks go to our colleagues from WP3 (Andy Cooke, Laurence Field and Steve Fisher) for their R-GMA support; and to our colleagues from WP7 (Robert Harakaly and Franck Bonnassieux) for providing the network monitoring software and their help with its integration with Reptor. We are also thankful to Ann Chervenak and Carl Kesselman for many stimulating discussions.

## REFERENCES

- [1] R. Alifieri, R. Cecchini, V. Ciaschini, L. Agnello, A. Frohner, A. Gianoli, K. Loerentey, and F. Spataro. An Authorization System for Virtual Organizations. In *European Across Grids Conference*, Santiago de Compostela, Spain, February 2003.
- [2] B. Allcock, J. Bester, J. Bresnahan, et al. Efficient Data Transport and Replica Management for High-Performance Data-Intensive 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.
- [3] Bill Allcock, Ian Foster, Veronika Nefedov, Ann Chervenak, Ewa Deelman, Carl Kesselman, Jason Lee, Alex Sim, Arie Shoshani, Bob Drach, and Dean Williams. High-Performance Remote Access

- to Climate Simulation Data: A Challenge Problem for Data Grid Technologies. In *Supercomputing 2001*, Denver, Texas, November 2001.
- [4] Chaitanya Baru, Reagan Moore, Arcot Rajasekar, and Michael Wan. The SDSC Storage Research Broker. In *CASCON'98*, Toronto, Canada, 30 November - 3 December 1998.
- [5] W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. Design of a Replica Optimisation Framework. Technical Report DataGrid-02-TED-021215, CERN, Geneva, Switzerland, December 2002. EU DataGrid Project.
- [6] 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.
- [7] A. Chervenak, E. Deelman, I. Foster, L. Guy, A. Iamnitchi, C. Kesselman, W. Hoschek, M. Rippeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Gigggle: A Framework for Constructing Scalable Replica Location Services. In *SC'2002*, Baltimore, USA, November 2002.
- [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 *10th IEEE Symposium on High Performance and Distributed Computing (HPDC-10)*, San Francisco, California, August 7-9 2001.
- [9] European datagrid project. [www.eu-datagrid.org](http://www.eu-datagrid.org).
- [10] EU DataGrid WP1. Definition of Architecture, Technical Plan and Evaluation Criteria for Scheduling, Resource Management, Security and Job Description. EU DataGrid Project. Deliverable D1.2, September 2001.
- [11] EU DataGrid WP2. EDG Replica Manager. <http://cern.ch/grid-data-management/edg-replica-manager>.
- [12] EU DataGrid WP3. R-GMA: Relation Grid Monitoring Architecture. <http://hepunx.rl.ac.uk/edg/wp3/>.
- [13] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *Proceedings of the 5th ACM Conference on Computer and Communications Security Conference*, pages 82–92, November 1998.
- [14] L. Guy, P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. Replica Management in Data Grids. Technical report, GGF5 Working Draft, July 2002.
- [15] J. Nick I. Foster, C. Kesselman and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.
- [16] 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.
- [17] Heinz Stockinger. *Database Replication in World-Wide Distributed Data Grids*. PhD thesis, Institute of Computer Science and Business Informatics, University of Vienna, December 2001.
- [18] Heinz Stockinger, Asad Samar, Shazhad Muzaffar, and Flavia Donno. Grid data mirroring package (gdmp). *Journal of Scientific Programming*, 10(2), 2002.

- [19] I. Terekhov, R. Pordes, V.White, L. Lueking, L. Carpenter, H. Schellman, J. Trumbo, S. Veseli, and M. Vranicar. Distributed Data Access and Resource Management in the D0 SAM System. In *10thIEEE Symposium on High Performance and Distributed Computing (HPDC-10)*, San Francisco, California, August 7-9 2001.
- [20] Tomcat Web Server. <http://jakarta.apache.org/>.
- [21] Gregor von Laszewski, Ian Foster, Jarek Gawor, and Peter Lane. A Java Commodity Grid Kit, Gregor von Laszewski. *Concurrency and Computation: Practice and Experience*, 13(8-9), 2001.
- [22] W3C. Web Services Activity. <http://www.w3c.org/2002/ws/>.
- [23] Rich Wolski, Neil Spring, and Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 1998.