Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing

Bill Allcock[1] Joe Bester[1] John Bresnahan[1] Ann L. Chervenak[2] Ian Foster[1,3] Carl Kesselman[2] Sam Meder[1] Veronika Nefedova[1] Darcy Quesnel[1] Steven Tuecke[1]

[1] Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
{allcock, bester, foster, nefedova, quesnel, tuecke}@mcs.anl.gov

[2] Information Sciences Institute
University of Southern California
Los Angeles, CA 90292
{annc, carl}@isi.edu

[3] Department of Computer Science & The Computation Institute
The University of Chicago
Chicago, IL 60637

Abstract

After studying data-intensive, high-performance computing applications such as high-energy physics and climate modeling, we conclude that these applications require two fundamental data management services: *secure, reliable, efficient transfer* of data in wide area environments and the ability to *register and locate multiple copies* of data sets. In this paper, we present our design of these services in the Globus grid computing environment. We also describe the performance of our current implementation.

1 Introduction

Data-intensive, high-performance computing applications require the efficient management and transfer of terabytes or petabytes of information in wide-area, distributed computing environments. Examples of data-intensive applications include experimental analyses and simulations in several scientific disciplines, such as high-energy physics, climate modeling, earthquake engineering and astronomy. These applications share several requirements. Massive data sets must be shared by a large community of hundreds or thousands of researchers who are distributed around the world. These researchers need efficient transfer of large data sets to perform analyses at their local sites or at other remote resources. In many cases, the researchers create local copies or replicas of the experimental data sets to overcome long wide-area data transfer latencies. The data management environment must provide security services such as authentication of users and control over who is allowed to access the data. In addition, once multiple copies of files are distributed at multiple locations, researchers would need

the ability to find all existing copies of the data set and determine whether to access an existing copy or create a new one to meet the performance needs of their applications.

The needs of these data-intensive computing applications rest on two fundamental data management services: *secure, reliable, efficient transfer* of data in wide area environments and the ability to *register and locate multiple copies* of data sets.

This paper presents our approach to providing these fundamental data management services in the Globus grid computing environment. We begin by describing in more detail the needs of two data-intensive applications. After briefly introducing the Globus grid computing environment, we present the design and initial performance measurements of our GridFTP protocol for efficient, secure data transfers as well as the Globus Replica Management architecture.

2 Data-Intensive Computing Requirements

We focus on two data-intensive computing applications: high-energy physics experiments and climate modeling applications. Important parameters required to characterize an application include: average file sizes, total data volume, rate of data creation, types of file access (write-once, write-many), expected access rates, type of storage system (file system or database), and consistency requirements for multiple copies of data.

2.1 High-energy Physics applications

First, we consider the data requirements of high-energy particle physics experiments. These experiments are characterized by the need to perform analysis over large amounts of data. Experiments that use the Large Hadron Collider at the European physics center CERN will produce large amounts of raw and derived experimental results. Beginning in 2005, these experiments will produce several petabytes of data per year for approximately 15 years. We are working with several physics experiments (ATLAS, CMS) that will produce LHC data as well as other physics data sets (BaBar).

There are two types of data generated by physics experiments:

- ***Experimental data*** represents the information collected by the experiment. There is a single creator of this data, and once created, it is not modified. However, data may be collected incrementally over a period of weeks.
- ***Metadata*** captures information about the experiment (such as the number of events) and the results of analysis. Multiple individuals may create metadata[CFK1]. The volume of metadata is typically smaller than that of experimental data.

The BaBar, CMS and ATLAS experiments use object-oriented Objectivity databases to store experimental results. Objectivity stores collections of objects in a single file called a *database.* Databases can be grouped into larger collections called *federations.* Typical

database file sizes are approximately 2 to 10 gigabytes. Federations are currently limited to 64K files; however, future versions of Objectivity will eliminate this restriction. The BaBar experiment plans to exploit this feature to reduce database file sizes to approximately 1 gigabyte and increase the size of a federation to include millions of database files. BaBar metadata files are currently 2 gigabytes in size, with plans to reduce this size in the future.

While in principle the primary databases produced by physics experiments are read-only (once created, their contents do not change), we find in practice that during an initial data production period of several weeks, database files change as new objects are added. For example, in the BaBar experiment, objects are appended to database files in a federation over several weeks; during this period, many database files may be changing simultaneously until the files are filled, after which they do not change further.

Metadata may be either modified or augmented over time, even after the initial period of producing experimental data. For example, in the CMS experiment, metadata files change to reflect the increasing total number of events in the database.

The consumers of experimental physics data and metadata will number in the hundreds or thousands. These users are distributed at many sites worldwide.

Because of the geographic distribution of the participants in a particle physics experiment, it is desirable to make copies or ***replicas*** of the data being analyzed to minimize the access time to the data. For example, Figure 1 shows the expected replication scheme for the physics data sets generated by the CERN Large Hardron Collider. Files are replicated in a hierarchical manner, with all files stored at a central location (CERN) and decreasing subsets of the data set stored at national and regional data centers [1][2].

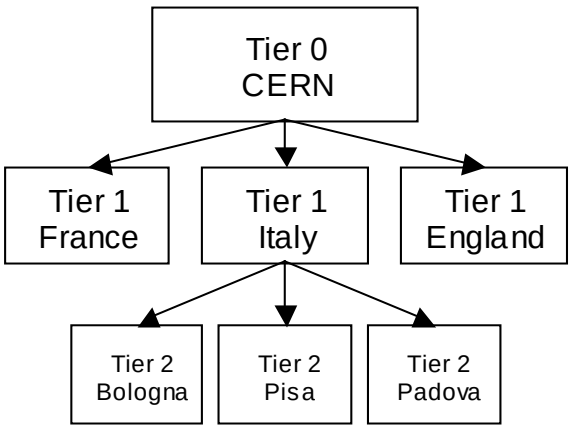

Figure 1: Scheme for hierarchical replication of Physics data

Replication of physics data sets is complicated by several factors. First, security services are required to authenticate the user and control access to storage systems. Next, because data sets are so large, it may be desirable to replicate only "interesting" subsets of the data. Finally, since experimental data and metadata files may be modified, any updates

to the databaes files must be propagated to all replicas. In the case of the BaBar experiment, experimental database files are logically appended over a period of several weeks. During this initial production period, users want access to the data as it is being produced. The experiment provides periodic batches of updates to remote replicas of the data sets, typically every few days or every week. Since metadata updates can take place over an indefinite period, these must also be propagated periodically to all replicas. BaBar uses a master-slave model to propagate modifications to replicas. This model is simple to implement in the physics environment because the physicists have partitioned the object identifier space so that each participating site has exclusive write access to a predefined subset of objects in the federation. The exclusive writer of a particular object acts as the master in update operations.

Table 1 summarizes the characteristics of high-energy physics applications.

Rate of data generation (starting 2005)	Several petabytes per year
Typical experimental database sizes	2 to 10 gigabytes
Typical metadata database sizes	2 gigabytes
Maximum number of database files in federation	Currently 64K; eventually millions
Period of updates to experimental data	Several weeks
Period of updates to metadata	Indefinite
Type of storage system	Object-oriented database
Number of data consumers	Hundreds to thousands

Table 1: Characteristics of high-energy physics applications

2.2 Climate Modeling Application

A second example application is climate modeling. Climate modeling research groups sometimes generate large *reference simulations* that are made available to a large international community. The output from these simulations can be large (many terabytes). The simulation data is typically generated at one or more supercomputer centers and is released in stages to progressively larger communities: first the research collaboration that generated the data, then perhaps to selected colleagues, and eventually to the entire community. Thus, these applications require access control to determine which users are allowed to view the collection at each stage.

In contrast to the physics community, data is not maintained in databases but rather as flat files, typically stored in a structured data format such as NetCDF, with associated metadata. In addition, files are not updated once released.

However, as in the physics application, researchers in the climate modeling community will find it convenient to store local copies of portions of the data set. Therefore, the application will have similar needs for managing copies of data sets at multiple locations, as well as higher-level services such as replica selection or automatic replica creation.

2.3 Summary of Application Requirements

We observe that these scientific applications, as well as others we have examined in such areas as earthquake engineering and astronomy, require two fundamental data management components, upon which higher-level components can be built:

- **A reliable, secure, high-performance data transfer** protocol for use in wide area environments. Ideally, this protocol would be universally adopted to provide access to the widest variety of available storage systems.
- **Management of multiple copies of files and collections of files**, including services for registering and locating all physical locations for files and collections.

Higher-level services that can be built upon these fundamental components include reliable creation of a copy of a large data collection at a new location; selection of the best replica for a data transfer operation based on performance estimates provided by external information services; and automatic creation of new replicas in response to application demands.

In the next section, we present an overview of the Globus grid computing environment and our approach to providing these fundamental data management services.

3 The Globus Architecture and Data Management

The term *grid computing* refers to the emerging computational and networking infrastructure that is designed to provide pervasive, uniform and reliable access to data, computational, and human resources distributed over wide area environments. For example, in a computational grid environment, scientists at locations throughout the world can share data collection instruments such as particle colliders, compute resources such as supercomputers and clusters of workstations, and community data sets stored on network caches and hierarchical storage systems.

The Globus project provides middleware services for grid computing environments. There are four main components of Globus. The Grid Security Infrastructure (GSI) provides authentication and authorization services using public key certificates as well as Kerberos authentication. The Globus Resource Management architecture provides a language for specifying application requirements and mechanisms for immediate and advance reservations of one or more computational components. This architecture also provides several interfaces for submitting jobs to remote machines. The Globus Information Management architecture provides a distributed scheme for publishing and retrieving information about resources in the wide area environment. A distributed collection of information servers is accessed by higher-level services that perform resource discovery, configuration and scheduling. The last major component of Globus is the Data Management architecture.

Based on the application needs presented in the previous section, the Globus Data Management architecture, or *Data Grid*, provides the two fundamental components: a universal data transfer protocol for grid computing environments called *GridFTP* and a *Replica Management* infrastructure for managing multiple copies of shared data sets. In the remainder of this paper, we present the design of these two fundamental components. We discuss our initial implementation as well as preliminary performance measurements.

4 GridFTP: A Secure, Efficient Data Transport Mechanism

Data-intensive scientific and engineering applications require both ***transfers*** of large amounts of data (terabytes or petabytes) between storage systems and ***access*** to large amounts of data (gigabytes or terabytes) by many geographically distributed applications and users for analysis, visualization, etc.

There are already a number of storage systems in use by the Grid community, each of which was designed to satisfy specific needs and requirements for storing, transferring and accessing large datasets. These include the Distributed Parallel Storage System (DPSS) and the High Performance Storage System (HPSS), which provide high-performance access to data and utilize parallel data transfer and/or striping across multiple servers to improve performance [1][2]. The Distributed File System (DFS) supports high-volume usage, dataset replication and local caching. The Storage Resource Broker (SRB) connects heterogeneous data collections, provides a uniform client interface to storage repositories, and provides a metadata catalog for describing and locating data within the storage system [4]. Other systems allow clients to access structured data from a variety of underlying storage systems (e.g., HDF5 [5]).

Unfortunately, most of these storage systems utilize incompatible and often unpublished protocols for accessing data, and therefore require the use of their own client libraries to access data. These incompatible protocols and client libraries effectively partition the datasets available on the grid. Applications that require access to data stored in different storage systems must use multiple access methods.

To overcome these incompatible protocols, we have proposed a universal grid data transfer and access protocol called ***GridFTP*** that provides secure, efficient data movement in Grid environments. This protocol, which extends the standard FTP protocol, provides a superset of the features offered by the various Grid storage systems currently in use. We argue that using GridFTP as a common data access protocol would be mutually advantageous to grid storage providers and users. Storage providers would gain a broader user base, because their data would be available to any client, while storage users would gain access to a broader range of storage systems and data.

We chose to extend the FTP protocol because we observed that FTP is the protocol most commonly used for data transfer on the Internet and the most likely candidate for meeting the Grid's needs. The FTP protocol is an attractive choice for several reasons. First, FTP

is a widely implemented and well-understood IETF standard protocol. As a result, there is a large base of code and expertise from which to build. Second, the FTP protocol provides a well-defined architecture for protocol extensions and supports dynamic discovery of the extensions supported by a particular implementation. Third, numerous groups have added extensions through the IETF, and some of these extensions will be particularly useful in the Grid. Finally, in addition to client/server transfers, the FTP protocol also supports transfers directly between two servers, mediated by a third party client (i.e. "third party transfer").

4.1 Features of GridFTP

Next, we describe the protocol extensions in GridFTP. Some of these features are supported by FTP extensions that have already been standardized in the IETF, but which are currently seldom implemented. Other features are new extensions to FTP.

4.1.1 Grid Security Infrastructure (GSI) and Kerberos support

Robust and flexible authentication, integrity, and confidentiality features are critical when transferring or accessing files. GridFTP must support Grid Security Infrastructure (GSI) and Kerberos authentication, with user controlled setting of various levels of data integrity and/or confidentiality. GridFTP provides this capability by implementing the GSSAPI authentication mechanisms defined by RFC 2228, "FTP Security Extensions".

4.1.2 Third-party control of data transfer

To manage large data sets for distributed communities, we must provide authenticated ***third-party*** control of data transfers between storage servers. A third-party operation allows a "third-party" user or application at one site to initiate, monitor and control a data transfer operation between two other "parties": the source and destination sites for the data transfer. Our implementation adds GSSAPI security to the existing third-party transfer capability defined in the FTP standard. The "third-party" authenticates itself on a local machine, and GSSAPI operations authenticate the third party to the source and destination machines for the data transfer.

4.1.3 Parallel data transfer

On wide-area links, using multiple TCP streams in parallel (even between the same source and destination) can improve aggregate bandwidth over using a single TCP stream. GridFTP supports parallel data transfer through FTP command extensions and data channel extensions.

4.1.4 Striped data transfer

Data may be striped or interleaved across multiple servers, as in a DPSS network disk cache or a striped file system. GridFTP includes extensions that initiate striped transfers,

which use multiple TCP streams to transfer data that is partitioned among multiple servers. Striped transfers provide further bandwidth improvements over those achieved with parallel transfers. We have defined GridFTP protocol extenstions that support striped data transfers

4.1 5 Partial file transfer

Many applications would benefit from transferring portions of files rather than complete files. This is particularly important for applications like high-energy physics analysis that require access to relatively small subsets of massive, object-oriented physics database files. The standard FTP protocol requires applications to transfer entire files, or the remainder of a file starting at a particular offset. GridFTP introduces new FTP commands to support transfers of subsets or regions of a file.

4.1.6 Automatic negotiation of TCP buffer/window sizes

Using optimal settings for TCP buffer/window sizes can have a dramatic impact on data transfer performance. However, manually setting TCP buffer/window sizes is an error-prone process (particularly for non-experts) and is often simply not done. GridFTP extends the standard FTP command set and data channel protocol to support both manual setting and automatic negotiation of TCP buffer sizes for large files and for large sets of small files.

4.1.7 Support for reliable and restartable data transfer

Reliable transfer is important for many applications that manage data. Fault recovery methods for handling transient network failures, server outages, etc. are needed. The FTP standard includes basic features for restarting failed transfers that are not widely implemented. The GridFTP protocol exploits these features and extends them to cover the new data channel protocol.

4.2 The GridFTP Protocol Implementation

In this section, we briefly present the implementation of the GridFTP protocol in the Globus Grid computing environment. Our current implementation is an alpha release of the gridFTP libraries, available with limited support to a small number of users. The current implementation supports partial file transfers, third-party transfers, parallel transfers and striped transfers. This implementation does not yet support automatic negotiation of TCP buffer/window sizes.

The implementation consists of two main libraries implemented in C: the globus_ftp_control_library and the globus_ftp_client_library.

The **globus_ftp_control_library** implements the control channel API. This API provides routines for managing a GridFTP connection, including authentication, creation

of control and data channels, and reading and writing data over data channels. Having separate control and data channels, as defined in the FTP protocol standard, greatly facilitates the support of such features as parallel transfers, striped transfers and third-party data transfers. For parallel and striped transfers, the control channel is used to specify a put or get operation; multiple parallel TCP data channels provide concurrent transfers. In third-party transfers, the initiator monitors or aborts transfers via the control channel, while data is transferred over one or more data channels between source and destination sites.

The **globus_ftp_client_library** implements the GridFTP client API. This API provides higher-level client features on top of the globus_ftp_control library, including complete file get and put operations, calls to set the level of parallelism for parallel data transfers, partial file transfer operations, third-party transfers, and eventually, functions to set TCP buffer sizes.

4.3 Performance of gridFTP data transfers

Next, we present performance measurements of data transfers using a prototype implementation of the gridFTP protocol.

In Figure 1, we show the performance of *parallel* GridFTP transfers as the number of simultaneous TCP streams between the source and destination hosts increases from 1 to 32. This transfer was measured between a host at Argonne National Laboratory and another host at Lawrence Berkeley National Laboratory. Bandwidth increases with the number of streams. However, as the number of streams increases, the benefit of adding a stream diminishes. From these tests, we conclude that using eight streams will give us most of the benefits possible with parallel transfers. We use these numbers in the remainder of our tests.

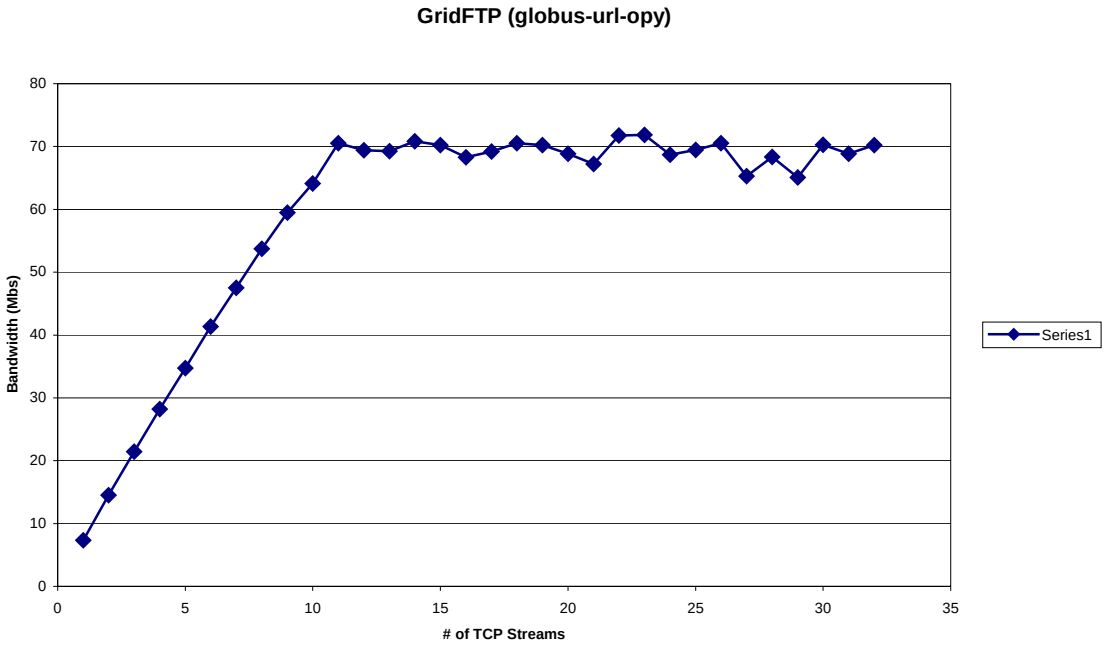

Figure 1: Bandwidth of parallel GridFTP transfer with increasing number of simultaneous TCP streams

In Figure 2, we show aggregate parallel bandwidth for a period of approximately 14 hours during the Supercomputing 2000 conference in Dallas, Texas, on November 7, 2000. This data corresponds to parallel transfers between two hosts using up to eight simultaneous streams. The graph includes drops in performance due to various network problems, including a power failure for the Supercomputing network (SciNet), DNS problems, and backbone problems on the exhibition floor. Because the GridFTP protocol supports restart of failed transfers, the interrupted transfers are able to continue as soon as the network is restored. Toward the right side of the graph, we show the increase in aggregate bandwidth as parallelism increases. One notable feature of the graph is the frequent drop in bandwidth to relatively low levels. This is due to our current implementation of GridFTP, which destroys and rebuilds the TCP connection between subsequent transfers. Our plans to address this performance issue with data channel caching are discussed below.

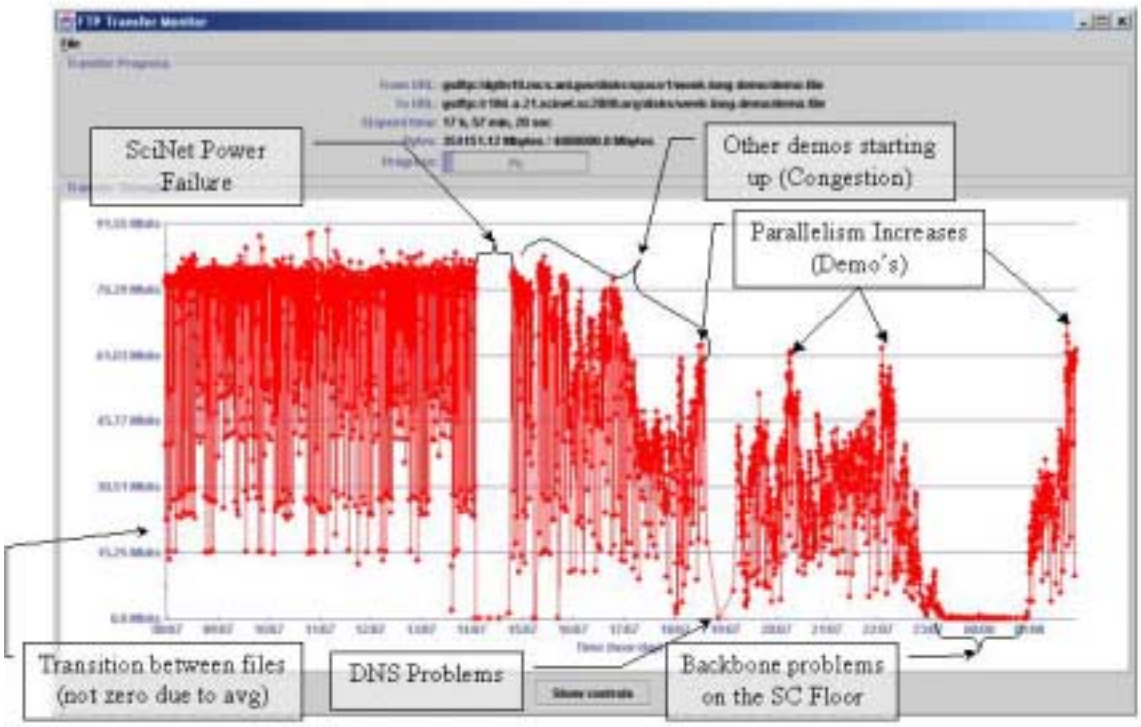

Figure 2: Supercomputing 2000 bandwidth over period

Finally, we present the bandwidth we achieved during the Network Challenge competition at the Supercomputing 2000 conference. Our configuration for this competition consisted of eight linux workstations on the SC2000 exhibition floor sending data across the wide area network to eight workstations (four linux, four solaris) at Lawrence Berkeley Laboratory. Figure 3 illustrates this configuration. We used *striped* transfers during this competition, with a 2-gigabyte file partitioned across the eight workstations on the exhibition floor. Each workstation actually had four copies of its file partition. On each server machine, a new transfer of a copy of the file partition was initiated after 25% of the previous transfer was complete. Each new transfer creates a new TCP stream. At any time, there are up to four simultaneous TCP streams transferring data from each server in the cluster of eight workstations, for a total of up to 32 simultaneous TCP streams. Because our current implementation requires that a TCP stream be broken down and restarted between subsequent transfers, there are often fewer than four simultaneous streams transferring data on each host.

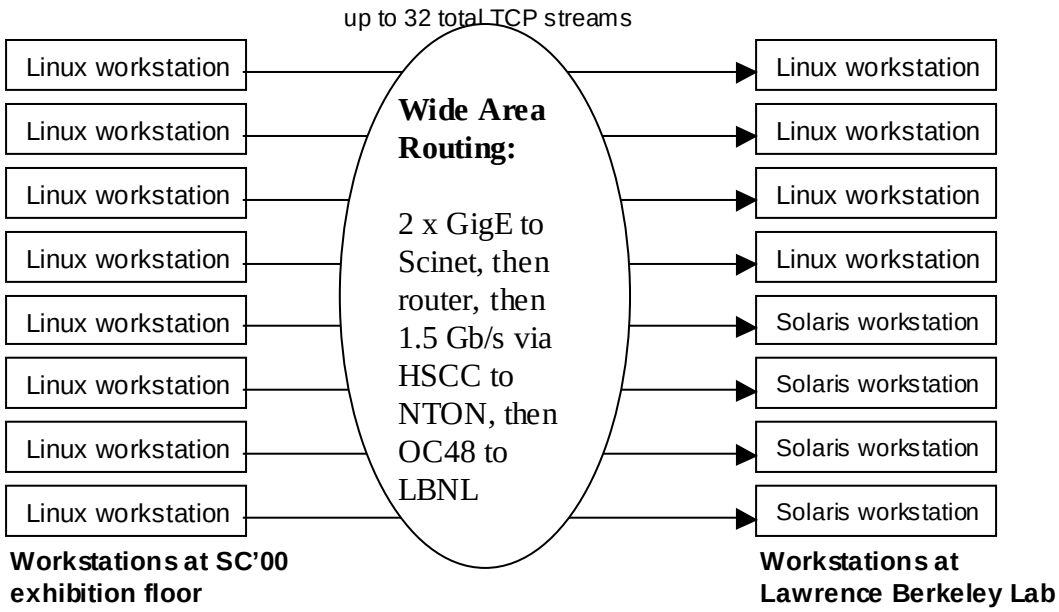

Figure 3: Experimental configuration for GridFTP experiments at SC'00 in Dallas, Texas, November 2000.

Table 2 summarizes the results of our Network Challenge competition entry. We achieved a peak transfer rate of 1.55 gigabits/second over an interval of 0.1 seconds. This striped configuration was able to transfer a peak rate of 1.03 gigabits/second over an interval of 5 seconds. Over the hour-long period of our competition entry, we sustained a data rate of 512.9 megabits per second. This corresponded to a total data transfer during that hour of 230.8 gigabytes, or a quarter of a terabyte.

Striped servers at source location	8
Striped servers at destination location	8
Maximum simultaneous TCP streams per server	4
Maximum simultaneous TCP streams overall	32
Peak transfer rate over 0.1 seconds	1.55 Gbits/sec
Peak transfer rate over 5 seconds	1.03 Gbits/sec
Sustained transfer rate over 1 hour	512.9 Mbits/sec
Total data transferred in 1 hour	230.8 Gbytes

Table 2: Network Challenge Configuration and Performance Results

The next version of the GridFTP implementation will support higher bandwidth with *data channel caching*. This mechanism allows a client to indicate that a TCP stream is likely to be re-used soon after the existing transfer is complete. In response to this hint, we will temporarily keep the TCP channel active and allow subsequent transfers to use the channel without requiring costly breakdown, restart and re-authentication operations.

5 Replica Management

In this section, we present the Globus Replica Management architecture, which is responsible for managing complete and partial copies of data sets. Replica management is an important issue for a number of scientific applications. For example, consider a data set that contains petabytes of experimental results for a particle physics application. While the complete data set may exist in one or possibly several physical locations, it is likely that many universities, research laboratories or individual researchers will have insufficient storage to hold a complete copy. Instead, they will store copies of the most relevant portions of the data set on local storage for faster access.

Services provided by a replica management system include:

- creating new copies of a complete or partial data set
- registering these new copies in a *Replica Catalog*
- allowing users and applications to query the catalog to find all existing copies of a particular file or collection of files
- selecting the ``best'' replica for access based on storage and network performance predictions provided by a Grid information service

The Globus replica management architecture is a layered architecture. At the lowest level is a *Replica Catalog* that allows users to register files as logical collections and provides mappings between logical names for files and collections and the storage system locations of one or more replicas of these objects. We have implemented a *Replica Catalog API* in C as well as a command-line tool; these functions and commands perform low-level manipulation operations for the replica catalog, including creating, deleting and modifying catalog entries. Finally, we have defined a higher-level *Replica Management API* that creates and deletes replicas on storage systems and invokes low-level commands to update the corresponding entries in the replica catalog.

In this section, we present the design of the Replica Catalog and the corresponding APIs. We describe the current state of our implementation and present performance numbers for a replica catalog implemented as an LDAP directory service.

The basic replica management services that we provide can be used by higher-level tools to *select among replicas* based on network or storage system performance or *automatically to create new replicas* at desirable locations. We will implement some of these higher-level services in the next generation of our replica management infrastructure.

5.1 The Replica Catalog

As mentioned above, the purpose of the replica catalog is to provide mappings between logical names for files or collections and one or more copies of the objects on physical storage systems. The catalog registers three types of *entries*: logical collections, locations and logical files.

A ***logical collection*** is a user-defined group of files. We expect that users will find it convenient and intuitive to register and manipulate groups of files as a collection, rather than requiring that every file be registered and manipulated individually. Aggregating files should reduce both the number of entries in the catalog and the number of catalog manipulation operations required to manage replicas.

Location entries in the replica catalog contain all the information required for mapping a logical collection to a particular physical instance of that collection. The location entry may register information about the physical storage system, such as the hostname, port and protocol. In addition, it contains all information needed to construct a URL that can be used to access particular files in the collection on the corresponding storage system. Each location entry represents a complete or partial copy of a logical collection on a storage system. One location entry corresponds to exactly one physical storage system location. The location entry explicitly lists all files from the logical collection that are stored on the specified physical storage system.

Each logical collection may have an arbitrary number of associated location entries, each of which contains a (possibly overlapping) subset of the files in the collection. Using multiple location entries, users can easily register logical collections that span multiple physical storage systems.

Despite the benefits of registering and manipulating collections of files using logical collection and location objects, users and applications may also want to characterize individual files. For this purpose, the replica catalog includes optional entries that describe individual ***logical files***. Logical files are entities with globally unique names that may have one or more physical instances. The catalog may optionally contain one logical file entry in the replica catalog for each logical file in a collection.

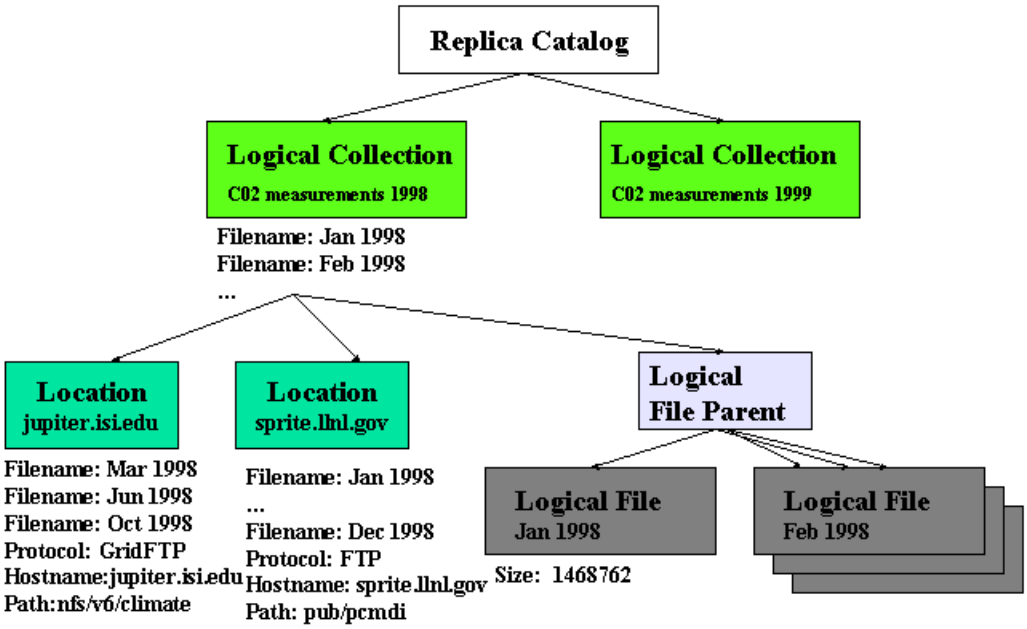

Figure 4: A Replica Catalog for a climate modeling application.

Figure 4 shows an example replica catalog for a climate modeling application. This catalog contains two logical collections with CO_2 measurements for 1998 and 1999. The 1998 collection has two physical locations, a partial collection at jupiter.isi.edu and a complete collection at sprite.llnl.gov. The location entries contain attributes that list all files stored at a particular physical location. They also contain attributes that provide all information (protocol, hostname, port, path) required to map from logical names for files to URLs corresponding to file locations on the storage system. The example climate modeling catalog also contains logical file entries for each file in the collection. These entries provide size information for individual files.

5.2 Replica Catalog API and Command Line Tool

We have implemented an API for low-level replica catalog manipulation as a C library called **globus_replica_catalog.c**. In addition, we have implemented a straightforward command-line tool that provides similar functionality.

All operations on the replica catalog being by establishing a connection to a logical collection. This may include creating a new logical collection entry if the specified collection doesn't already exist. After establishing this connection, API and command line functions operate on the collection entry or on its corresponding location or logicalfile entries. Possible operations on replica catalog entries fall into three general categories:

- *Create and delete entire entries:* When creating a logical collection, location or logicalfile entry, the client supplies all attributes to be associated with the new entry. For example, the client might specify all filenames to be associated with a logical collection entry or the hostname, protocol and path information required for mapping to physical file locations in a location entry. Deleting a catalog entry requires knowing only the name of the entry and its corresponding logical collection.
- *Add, list or delete attributes of an entry:* These operations allow the user to manipulate the attributes of an entry. For example, as new files are produced by an experimental physics application, their names might be added as attributes to a logical collection entry. If space constraints on a storage system required deleting certain files, a location entry should be updated to remove the corresponding filenames from the catalog entry.
- *List or search for specified entries:* These operations search the catalog and return a list of entries and their attributes that match the specified search criteria. For example, a simple *list* operation might return all the location entries associated with a logical collection. More complex *search* operations are used to identify all physical locations where a particular set of files is stored. These search operations are essential to higher-level tools that select among possible source and destination locations for data transfer operations.

5.3 Replica Management API

In this section, we describe the high-level *Replica Management API* that is currently being designed. The purpose of this API is to build higher-level functionality on top of the low-level Replica Catalog API. In addition to making low-level Replica Catalog API calls, the functions in the Replica Management API can manipulate storage systems directly, including copying files and checking file status, such as last modification time. This library has not yet been implemented.

The functions of the Replica Management API can be categorized as follows:

- *Session management:* These functions create, configure and destroy session handles. The use of session handles allows caching of connection states to GridFTP and LDAP servers, so that a series of operations on multiple files can be performed efficiently, without requiring separate FTP or LDAP commands for each operation.
- *Catalog creation:* These functions create and populate one or more entries in a replica catalog. There are basic functions to create empty logical collection and location objects. In addition, there are functions to *register* and *publish* filenames into logical collections and locations. A client *registers* a file that already exists on a storage system by adding the filename to collection and location entries in the catalog. Alternatively, the client *publishes* a file into logical collection and location entries by first copying the file onto the corresponding storage system and then updating the catalog entries.
- *File maintenance:* These functions include copy, update, delete operations on physical files with corresponding replica catalog updates. For example, the copy operation copies a file from one physical location to another, updating the corresponding location entry in the replica catalog. The delete operation removes a filename from a location entry in the replica catalog and optionally also deletes the file on the corresponding physical storage system.
- *Access control:* These functions control who is allowed to access and make replicas of individual files and logical collections.

5.4 Implementation and Performance

We have implemented the Replica Catalog as a Lightweight Directory Access Protocol (LDAP) directory. We have implemented the Replica Catalog API as a C library in the Globus grid computing environment. In addition, we have implemented a command line tool that calls replica catalog API functions.

[Note to reviewers: we are currently running extensive performance measurements of a test replica catalog to determine its scalability as collection size and number of collections increases. If space allows, we will include these results in the final version of the paper, but we have left them out of the current draft.]

6 Conclusions

We have argued that high-performance, distributed computing applications require two fundamental services: *secure, reliable, efficient transfer* of data in wide area environments and the ability to *register and locate multiple copies* of data sets. Upon these fundamental components, we can build higher-level services, including reliable creation of a copy of a data collection at a new location; selection of the best replica for a data transfer operation based on performance; and automatic creation of new replicas in response to application demands.

We have presented our design and implementation of these fundamental services in the Globus grid computing environment. We presented the GridFTP protocol, which implements extensions to provide GSI security, parallel, striped, partial and third-party transfers. We also presented the performance of GridFTP parallel and striped accesses. Next, we described the Globus Replica Management architecture, which is responsible for managing complete and partial copies of data sets. We described the design of our Replica Catalog, as well as APIs for manipulating the catalog and storage systems.

We are working with several high-performance, data-intensive computing applications, such as high-energy physics, climate modeling and earthquake engineering. Working with these application communities provides an ideal opportunity to evaluate the usefulness of the Globus data management architecture.

Acknowledgements

We are grateful to Marcus Thiebaux and Soonwook Hwang for their work in characterizing the performance of LDAP servers. Brian Toonen helped to optimize the GridFTP code in preparation for the SC ‘2000 conference. This work has been funded in part by the National Science Foundation and the Department of Energy.

References

[1] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, K. Stockinger, “Data Management in an International Grid Project”, 2000 International Workshop on Grid Computing (GRID 2000), Bangalore, India, December 2000.

[2] K. Holtman, “Object Level Replication for Physics”, Proceedings of 4th Annual Globus Retreat, Pittsburgh, July 2000.

[3] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, “The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets,” to be published in the Journal of Network and Computer Applications .

[4] Globus Data Management web page,www.globus.org/research/data-management.html

[5] Tierney, B. Lee, J., Crowley, B., Holding, M., Hylton, J., Drake, F., "A Network-Aware Distributed Storage Cache for Data Intensive Environments", Proceedings of IEEE High Performance Distributed Computing conference(HPDC-8), August 1999.
[6] "Basics of the High Performance Storage System", www.sdsc.edu/projects/HPSS
[7] C. Baru, R. Moore, A. Rajasekar, M. Wan, "The SDSC Storage Resource Broker," Proc. CASCON'98 Conference, Nov.30-Dec.3, 1998, Toronto, Canada.
[8] HDF5 information available at http://hdf.ncsa.uiuc.edu/

[CFK1]We need to clarify metadata update issue: single writer (as in Babar), what about other experiments. Is there updates to objects, or are new metadata objects just added.