Protocols and Services for Distributed Data-Intensive Science

William Allcock, Ian Foster, and Steven Tuecke
Mathematics and Computer Science Division, Argonne National Laboratory

Ann Chervenak and Carl Kesselman
Information Sciences Institute, University of Southern California

Abstract. We describe work being performed in the Globus project to develop enabling protocols and services for distributed data-intensive science. These services include:

* High-performance, secure data transfer protocols based on FTP, plus a range of libraries and tools that use these protocols
* Replica catalog services supporting the creation and location of file replicas in distributed systems

These components leverage the substantial body of "Grid" services and protocols developed within the Globus project and by its collaborators, and are being used in a number of data-intensive application projects.

INTRODUCTION

We describe work being performed in the Globus project[1] to develop enabling protocols and services for distributed data-intensive science. We will begin with a discussion of the differences between protocols, API's, and services and why each is important. The features of our data transfer technology, GridFTP, and our replica catalog services will then be discussed. A full discussion of the protocols and API's is beyond the scope of this paper.

PROTOCOLS, API'S, AND SERVICES

There is a great deal of confusion regarding these terms. They are frequently used interchangeably, though incorrectly. Each plays a different role and provides distinct and important advantages.

A protocol defines the format of data that is sent between two systems, including the syntax of messages, character sets, and sequencing of messages. It does not specify whether this was accomplished by invoking a Java method, calling a C language function, or by someone sitting at a terminal typing the replies. The value of a protocol is that it provides *interoperability*. It allows a Java Application running on a Sun Workstation to work with a C application running on a Cray T3-E supercomputer, just as TCP/IP has enabled a heterogeneous collection of operating systems and computers to communicate over the Internet.

An Application Programmers Interface (API) specifies the interface to which an application programmer can write code. It specifies functionality, names, data types, parameter sequences, and return types. It is not an implementation, since several implementations of the same API can exist. The advantage to having a common API is *efficiency*. If the same API is supported on a variety of heterogeneous platforms, for example, as interfaces to different storage systems, then the programmer's task is greatly simplified.

Services are processes that are usually started automatically, always running, and provide basic functionality for other processes. They are also known as user agents and daemons. The E-Mail service is probably the most well known. It runs in the background. An E-Mail program (client) simply communicates with the server using a specified protocol and provides it with the data (address list, text, images, attachments, etc). The email service takes care of actually transferring the mail over the network and the service on the other end informs the appropriate client that mail has arrived. The advantages of a service are *faster application development*, *smaller programs* since each client does not need to have the service code present, and *more stable code* since the services can be more complicated to write.

DATA TRANSFER PROTOCOL: GRIDFTP

In Grid environments, access to distributed data is typically as important as access to distributed computational resources[2]. Distributed scientific and engineering applications require 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. Unfortunately, the lack of standard protocols for transfer and access of data in the Grid has led to a fragmented Grid storage community. Users who wish to access different storage systems are forced to use multiple protocols and/or APIs, and it is difficult to efficiently transfer data between these different storage systems.

We propose a common data transfer and access protocol called GridFTP[3,4] 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.

In order to make a common data transfer protocol attractive to users and developers of existing storage systems, we must provide a transfer protocol that offers a superset of the features offered by systems currently in regular use. In addition, the protocol must be extensible, in order to support future innovations by storage system users and developers.

We have observed that the FTP protocol is the protocol most commonly used for data transfer on the Internet, and the most likely candidate for meeting the Grid's needs. It is attractive in particular for the following reasons.

- It is a widely implemented and well-understood IETF standard protocol.
- There is a large code base and expertise from which to build.
- It provides a well-defined architecture for protocol extensions, and supports dynamic discovery of the extensions supported by a particular implementation.
- Numerous groups have added various extensions through the IETF. Some of these extensions are particularly useful in the Grid.
- In addition to client/server transfers (i.e. "put/get"), it also supports transfers directly between two servers, mediated by a third party client (i.e. "third party transfer").
- The separation of data and control channels onto different sockets allows for easier extensibility for parallel and striped transfers, efficiently transiting firewalls, etc.

Most current FTP implementations support only a subset of the features defined in the FTP protocol and its accepted extensions. Some of the seldom-implemented features are useful to Grid applications, but the standards also lack several features Grid applications require. We have selected a subset of the existing FTP standards and further extended them, adding the features described below. We believe that the resulting protocol is a suitable candidate for the common data transfer protocol for the grid.

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 GSI and Kerberos authentication, with user controlled setting of various levels of data integrity and/or confidentiality.

Third-party control of data transfer: In order to manage large data sets for large distributed communities, it is necessary to provide third-party control of transfers between storage servers. GridFTP provides this capability by adding GSSAPI security to the existing third-party transfer capability defined in the FTP standard.

Parallel data transfer: On wide-area links, using multiple TCP streams can improve aggregate bandwidth over using a single TCP stream. This is required both between a single client and a single server, and between two servers. GridFTP supports parallel data transfer through FTP command extensions and data channel extensions.

Striped data transfer: Partitioning data across multiple servers can further improve aggregate bandwidth. GridFTP supports striped data transfers through extensions defined in the Grid Forum draft.

Partial file transfer: Many applications require the transfer of partial files. However, standard FTP requires the application to transfer the entire file, or the remainder of a file starting at a particular offset. GridFTP introduces new FTP commands, to support transfers of regions of a file.

Support for reliable 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 transfer that are not widely implemented. The GridFTP protocol exploits these features, and substantially extends them.

REPLICA MANAGEMENT

In this section, we present the Globus Replica Management architecture[4]. Replica management is an important issue for a number of scientific applications. For example, consider the petabytes of experimental data that will be generated by the LHC[5]. 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.

Replica management system services 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. 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.

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.

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.

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 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.

REFERENCES

1. The Globus Project, www.globus.org
2. 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*.
3. Grid Forum GridFTP Introduction: http://www.sdsc.edu/GridForum/RemoteData/Papers/gridftp_intro_gf5.pdf
4. Grid Forum GridFTP Specification DRAFT: http://www.sdsc.edu/GridForum/RemoteData/Papers/gridftp_spec_gf5.pdf
5. 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.