File and Object Replication in Data Grids

Heinz Stockinger[1,2], Asad Samar[3], Bill Allcock[4], Ian Foster[4,5], Koen Holtman[3], Brian Tierney[1,6]

1) CERN, European Organization for Nuclear Research, CH-1211 Geneva 23, Switzerland
2) Inst. for Computer Science and Business Informatics, University of Vienna, A-1010 Vienna, Austria
3) California Institute of Technology, Pasadena, CA 91125, USA
4) Mathematics and Computer Science Division, Argonne National Laboratory, IL 60439, USA
5) Department of Computer Science & The Computation Institute, The University of Chicago, IL 60637, USA
6) Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
Heinz.Stockinger@cern.ch

Abstract

Data replication is a key issue in a Data Grid and can be managed in different ways and at different levels of granularity: for example, at the file level or object level. In the High Energy Physics community, Data Grids are being developed to support the distributed analysis of experimental data. We have produced a prototype data replication tool, the Grid Data Management Pilot (GDMP) that is in production use in one physics experiment, with middleware provided by the Globus Toolkit used for authentication, data movement, and other purposes. We present here a new, enhanced GDMP architecture and prototype implementation that uses Globus Data Grid tools for efficient file replication. We also explain how this architecture can address object replication issues in an object-oriented database management system. File transfer over wide-area networks requires specific performance tuning in order to gain optimal data transfer rates. We present performance results obtained with GridFTP, an enhanced version of FTP, and discuss tuning parameters.

1 Introduction

Data replication is an optimization technique well known in the distributed systems and database communities as a means of achieving better access times to data (data locality) and/or fault tolerance (data availability) [Bres99, Karg99, Tewa99]. This technique appears clearly applicable to data distribution problems in large-scale scientific collaborations, due to their globally distributed user communities and distributed data sites. As an example of such an environment, we consider the High Energy Physics community where several thousand physicists want to access the Terabytes and even Petabytes of data that will be produced by large particle detectors around 2006 at CERN, the European Organization for Nuclear Research.

In: Proc. 10th Intl. Symp. on High Performance Distributed Computing, IEEE Press, 2001.

The computing model of a typical next generation experiment at CERN foresees the use of a distributed network of regional centers, each equipped with computing and data storage facilities and linked with wide area network connections [Newm00]. Since these sites are intended to be used in a coordinated fashion, there is a natural mapping to a Grid environment [FoKe99a] and the High Energy Physics community is building a Data Grid [Cher00] to support the distributed management and analysis of its data. Recently, the European Data Grid Project ("EU DataGrid" [EDG01]) project has been initiated and a prototype project GDMP (Grid Data Management Pilot) [SaSt01] has been used in a production environment in an experiment involving the secure replication of database files between several sites in Europe and the U.S. GDMP provides file replication services and some preliminary storage management functionality. Although it is not yet a fully functional replication manager (e.g., see [Hosc00]), it does provide useful services and is extensible to meet future needs.

GDMP uses services provided by the Globus Toolkit [FoKe99b] for security and other purposes. An initial version, GDMP version 1.2 [GDMP01], was limited to transferring Objectivity [Obje01] database files. In more recent work, we have significantly extended GDMP capabilities by integrating two new Globus Data Grid tools [Allc01], available as an alpha release as of early 2001: the Globus Replica Catalog, which we use to store replica location metadata, and the GridFTP high-performance wide area transport library, which we use as our transport engine.

In this article, we describe how GDMP uses these new services to develop a significantly improved architecture. We provide performance results on GridFTP data transfers and also describe how GDMP is extended with an object replication feature that can be used for distributed data analysis.

The article is organized as follows. Section 2 gives background on the application domain (High Energy Physics), file vs. object replication and related Data Grid projects. In the next section we discuss Globus Data Grid tools: the Globus Replica Catalog and GridFTP. Section 4 elaborates on architectural aspects of GDMP and discusses the new components in detail. Object replica issues are presented in Section 5. Finally we present performance tests on GridFTP and make some concluding remarks.

2 Background and Related Work

In order to provide some background to our specific Data Grid domain, we discuss briefly the software engineering processes used in High Energy Physics (HEP) and the difficulties that these projects face in the future. We also review the various Data Grid projects that are exploring potential solutions.

2.1 High Energy Physics

In many next-generation HEP experiments, object-oriented software engineering tools and languages are used to develop the software infrastructure for the final physics analysis. To store the experiment's data, currently an object-oriented database management system or an object data store is assumed as the data persistency solution. At the highest level of abstraction in the experiment’s data models, all data are persistent objects and can be accessed through an object-oriented navigation mechanism.

The experiment's physics detector makes observations of high energy physics collisions. Each observation is called an “event” and has a unique event number. For each event, a number of objects are present. There are raw data objects which hold the data directly taken from the detector, and reconstructed objects which hold processed versions of this raw data.

The high level experiment's data view contains neither the concept of files nor the concept of data replication: all objects are supposed to simply “exist” without regard to how they are stored and how many replicas exist. Files and replication appear only at lower layers of abstraction as implementation mechanisms for the experiment's object view.

A single file will generally contain many objects. This is necessary because the number of objects is so large (in the order of 10^7 to 10^{10} for a modern physics experiment) that storing each object in a single file would lead to scalability problems in the file systems and tertiary storage systems used. Moreover, the object persistency solutions used only work efficiently if there are many objects per file. To map the high-level object view of the experiment to the a lower level storage infrastructure of replicated files, we assume a three-step process supported by three catalogs, as shown in Figure 1.

As most objects are read-only after creation, access patterns show considerable repetitiveness and locality, and both the user community and the hardware resources are highly distributed, support for replication is clearly desirable. Replication is also desirable because the (current production versions of the) object persistency layers in each site do not have the native ability to efficiently access objects on remote sites [YoMo00], as they were built under the assumption that a low latency exists when accessing storage.

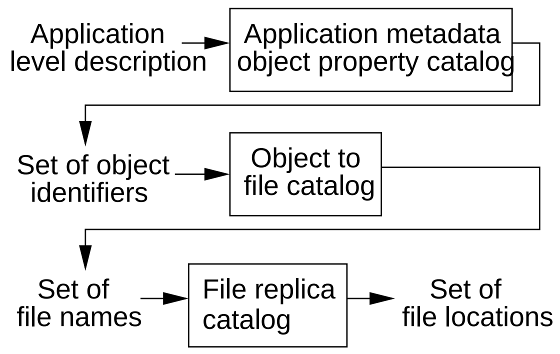
Application level description
Application metadata object property catalog
Set of object identifiers
Object to file catalog
Set of file names
File replica catalog
Set of file locations

Figure 1: Interaction between different catalogs when mapping the application-level view to lower level storage.

In this article we discuss both file and object replication. We define file replication as a mechanism that replicates data at the granularity of already-existing files. Object replication replicates data at the granularity of the individual objects, regardless of any currently existing mapping between objects and files. Figure 2 shows the difference between file replication and our approach for object replication.

In our approach, object replication is a multi-step process:

- First, on the source site, an object copier tool is used to copy the objects that need to be replicated into a new file.
- Second, the new file is moved to the destination site using a wide area file copy.
- As a final step, the new file can be deleted at the source site.

The object replication approach is considered further in Section 5. As discussed in Section 5, replication strategies with object granularity are potentially more efficient than file replication for some specific HEP workloads. This efficiency comes at the cost of greater complexity however, both in data manipulation and in the complexity of the object to file catalog [HoSt00].

The object data model of the HEP applications creates some specific difficulties for replication. If file replication is used, the replication mechanism cannot a priori treat every file as independent and self-contained, as tight navigational relations or synchronous updating constraints might couple the objects in several files. For instance, two objects in two separate files can have a navigational association between each other. If only one of these two files is replicated to a remote site, the navigation to the associated object might not be possible since the required file is not available locally too, and the object persistency layer at the remote site has no awareness of the files in other sites. Thus, the two files have to be treated as associated files and replicated together in order to preserve the navigation. A more detailed discussion on this topic can be found in [Stock01].

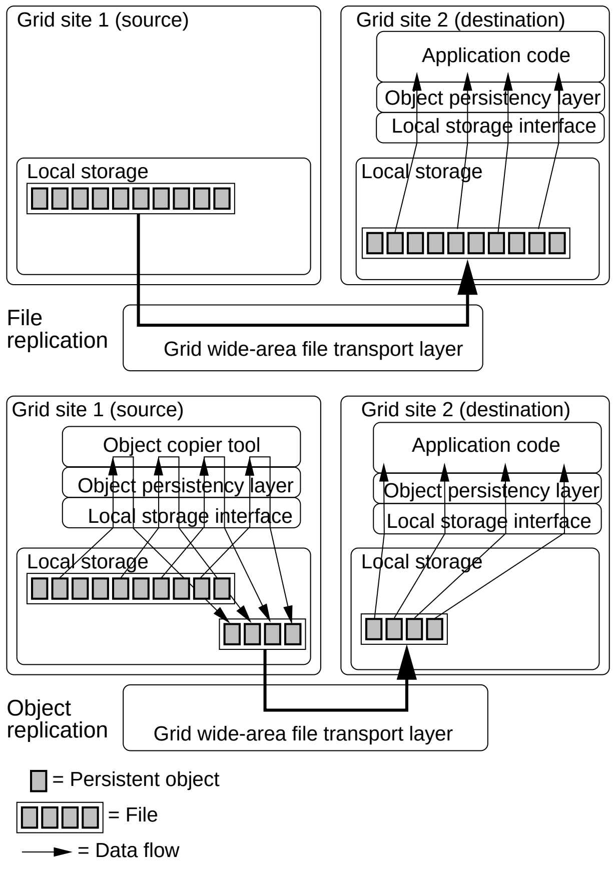

Figure 2: File and object replication in a Data Grid. File replication is shown at the top, object replication at the bottom. In both cases, the application code uses an object persistency layer to read the wanted objects from the file once it is on local storage.

If a "lazy" form of file replication is used, the mechanisms that steer replication will have to take into account any synchronous creation or update constraints between objects, and by extension between the replicated files. The model for file replication is therefore that "consistency policies", which flow from the application layer, will steer the replication layer. To minimize the constraints that need to be encoded in consistency policies, application designers must carefully think out the allocation of objects to files.

For object replication, the problem of update constraints is of course equally present, and could again be handled with application-defined consistency policies. However, to limit complexity, in our object replication solutions we are currently not considering any flexible application-defined policies. Instead we require that all objects entrusted to the object replication service are always read-only objects. This read-only requirement is no big burden for HEP applications: by using a sufficiently powerful and pervasive versioning mechanism, many (but not all) HEP persistent objects can be treated as read-only after initial creation.

2.2 Related Data Grid Projects

Data Grid concepts are being explored in a number of projects worldwide. In the U.S., the Earth System Grid (ESG) is applying Data Grid technologies to the management of climate data [Allc01b], while the Particle Physics Data Grid (PPDG) [PPDG01] and Grid Physics Network (GriPhyN) [GriP01] projects are both working in the HEP domain. Efforts have started within PPDG as well as EU DataGrid to use the GDMP code base. Since GDMP is and has been a mutual effort of EU DataGrid and PPDG, this collaboration also has the beneficial effort of encouraging these two projects to go in similar directions in the development of Grid tools. Work within the digital library community is also relevant [Moor99].

PPDG and GriPhyN are the most related projects in the HEP domain (GriPhyN is also addressing requirements of sky survey and astrophysics applications) and we state briefly how they differ from our approach.

GriPhyN is mainly addressing fundamental IT research focused on realizing the concept of Virtual Data. Virtual Data in general means that data does not necessarily have to be available in a persistent form but is created on demand and then materialized when it is requested. In this virtual data space, requests can be satisfied via direct access and/or computation, with local and global resource management, policy, and security constraints determining the strategy used.

PPDG is a project that deals only with High Energy Physics applications but focuses on more immediate issues relating to file replication, job scheduling, and so forth.

GDMP addresses a subset of the possible project scopes of EU DataGrid, GriPhyN, and PPDG, focusing on fast and efficient point-to-point file replication.

3 Globus Data Grid Tools

Since GDMP uses new Globus Data Grid tools [Allc01], we describe their features and functionality with respect to file replication.

3.1 Replica Catalog

The Globus replica catalog is intended as a fundamental building block in Data Grid systems. It addresses the common need to keep track of multiple physical copies of a single logical file by maintaining a mapping from logical file names to physical locations. The catalog contains three types of object. The highest-level object is the *collection*, a group of logical file names. Discussions with various user groups show that datasets are normally manipulated as a whole and the collection abstraction provides a convenient mechanism for doing this. A *location* object contains the information required to map between a logical filename (a globally unique identifier for a file: not a physical location) and the (possibly multiple) physical locations of the associated replicas. The final object is a *logical file entry*. This optional entry can be used to store attribute-value pair information for individual logical files. We believe that much of this type of data will be stored in a separate metadata catalog [BMRW98], but the facility is available.

The operations that can be performed on the catalog are as one might expect: creation and deletion of collection, location, and logical file entries; insertion and removal of logical file names into collections and locations; listing of the contents of collections and locations; and the heart of the system, a function to return all physical locations of a logical file. Further replica catalog documentation can be found at www.globus.org/datagrid/replica-catalog.html.

Replica catalog functions can be used directly in applications, but also form the basis (with GridFTP) for a replica management system that provides functions for the reliable creation, deletion, and management of replicas. Replica management documentation can be found at www.globus.org/datagrid/replica-management.html.

3.2 GridFTP

GridFTP is a data transfer and access protocol that provides secure, efficient data movement in Grid environments. The GridFTP protocol extends the standard FTP protocol, providing a superset of the features offered by the various Grid storage systems currently in use. We choose to work with the FTP protocol because it is the most commonly used protocol for data transfer on the Internet; of the existing candidates from which to start, we believe it comes closest to meeting the Grid's needs. The GridFTP protocol includes the following features:

- Public-key-based Grid Security Infrastructure (GSI) [FKT98] or Kerberos support (both accessible via GSS-API [Linn00])
- Third-party control of data transfer
- Parallel data transfer (one host to one host, using multiple TCP streams)
- Striped data transfer (m hosts to n hosts, possibly using multiple TCP streams if also parallel)
- Partial file transfer
- Automatic negotiation of TCP buffer/window sizes
- Support for reliable and restartable data transfer
- Integrated instrumentation, for monitoring ongoing transfer performance

Programmatic access to this functionality is provided via two primary libraries, globus_ftp_control and globus_ftp_client. These libraries have been used to develop a server, based on the Washington University FTP Daemon (wuftpd), that implements the GridFTP features listed above. A full-featured command line tool appropriate for scripting called globus_url_copy is provided. A version of the interactive ncftp client has also been developed that has GSI support and hence can communicate with GridFTP servers; however, this client does not incorporate the other features listed above. Further documentation for GridFTP and these libraries is available at www.globus.org/datagrid/gridftp.html.

4 GDMP Architecture

In this section, we briefly describe the entire GDMP architecture, focusing on the new features of our second-generation architecture, which concern namespace and file catalog management, efficient file transfer, and preliminary mass storage management. See [SaSt01] for a description of GDMP version 1.2, which is in production use in a CERN experiment.

4.1 General Architectural Issues

GDMP is a file replication software system that was initially designed to replicate Objectivity database files from one site (storage location) to one or more other remote sites. A storage location is considered to be a disk space on a single machine or on several machines connected via a local-area network and a network file system. Remote sites are connected to each other via long latency (as compared to local-area network) wide-area network connections. GDMP works as follows. A site produces a set of files locally and another site wants to obtain replicas of these files. In the case of Objectivity files, each site is running the Objectivity database management system locally that has a catalog of database files internally. However, the local Objectivity database management system does not know about other sites and a replication mechanism is required that can notify other sites about new files, efficiently transfer the files to the remote site, and integrate the filenames into the Objectivity internal file catalog. An additional server needs to be available at each site to handle replication requests and to trigger file transfers, notification messages, and updates of local catalog information. Simply put, this is done by a GDMP server running at each site where files are produced and possibly replicated.

With the new architecture and newly added components, GDMP has been extended to handle file replication independent of the file format. Note that we do not address replica synchronization issues, hence this work is useful mainly for read-only files. In GDMP 1.2, the file replication process was too tightly connected to Objectivity-specific features for naming conventions of logical and physical files and for obtaining information about the files from the Objectivity's catalog. This dependency is removed in the new version (the official release will be called GDMP 2.0) by splitting the data replication process into several steps. Other possible file types are Oracle files and flat files with particular internal structure. Thus, successfully replicating a file from one storage location to another one consists of the following steps:

- *pre-processing*: This step is specific to the file formats and might even be skipped in certain cases. This step prepares the destination site for replication, for example by creating an Objectivity federation at the destination site or introducing new schema in a database management system so that the files that are to be replicated can be integrated easily into the existing Objectivity federation.
- *actual file transfer*: This has to be done in a secure and efficient fashion; fast file transfer mechanisms are required.
- *post-processing*. The post-processing step is again file type specific and might not be needed for all file types. In the case of Objectivity, one post-processing step is to attach a database file to a local federation and thus insert it to an internal file catalog
- *insert the file entry into a replica catalog*: This step also includes the assignment of logical and physical filenames to a file (replica). This step makes the file (replica) visible to the Grid.

The GDMP replication process is based on the producer-consumer model: each data production site publishes a set of newly created files to a set of one or more consumer sites, and GDMP ensures that the necessary data transfer operations (including all the steps mentioned above) complete successfully. These services are implemented by a set of interacting servers, one per site participating in the data replication process.

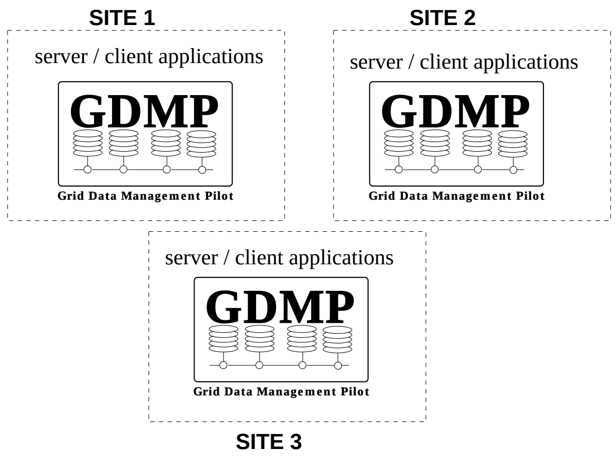
SITE 1
SITE 2
server / client applications
GDMP
Grid Data Management Pilot
server / client applications
GDMP
Grid Data Management Pilot
server / client applications
GDMP
Grid Data Management Pilot
SITE 3

Figure 3: Distributed sites and the location of GDMP servers/client applications

Figure 3 depicts a small Data Grid with only three sites where data is produced and replicated (consumed). Each of these sites deploys a GDMP server to interact with other sites and provides GDMP client commands for publishing file information to other sites (notifying other sites that new data is available) and initiating file replication requests for a set of files. In more detail, a high-level file get request is issued by a GDMP client application at one site to get files from another site and create replicas locally.

To sum up, GDMP client APIs provide four main services to the end-user [SaSt01]:

- subscribing to a remote site for getting informed when new files are created and made public,
- publishing new files and thus making them available and accessible to the Grid,
- obtaining a remote site's file catalog for failure recovery, and
- transferring files from a remote location to the local site.

Every client request to a GDMP server is authenticated and authorized by a security service. GDMP uses the Globus Security Infrastructure (GSI) [FKT98], which provides single sign capabilities for Grid resources.

Client requests are sent to the GDMP server through the Request Manager. The Request Manager is the client-server communication module, which is used to generate client requests and implement server functions for serving these requests. Using the Globus IO and Globus Data Conversion libraries, the Request Manager provides a limited Remote Procedure Call functionality.

File transfer requests are served by the GDMP Data Mover service that uses a local file transfer server such as FTP. Since file transfers must be both secure and fast, the Data Mover service has to use a file transfer mechanism that provides both features (more in Section 4.3). Once files are successfully transferred, they have to be inserted into a replica catalog. The Replica Catalog Service provides this functionality (see Section 4.2).

In an early version, GDMP was restricted to disk-to-disk file replication and it was assumed that all files are permanently available on disk. Since Data Grids deal with large amounts of data, files are permanently stored in Mass Storage Systems (MSS) such as HPSS and moved between disk to tape on demand. Thus, a disk pool is considered as a cache. GDMP provides a plug-in for initiating file stage requests on demand between a disk pool and a Mass Storage System (see Section 4.4).

Figure 4 illustrates these three principal components of the GDMP software. In the next subsections, we describe the replica catalog, data mover, and the storage management service in detail.

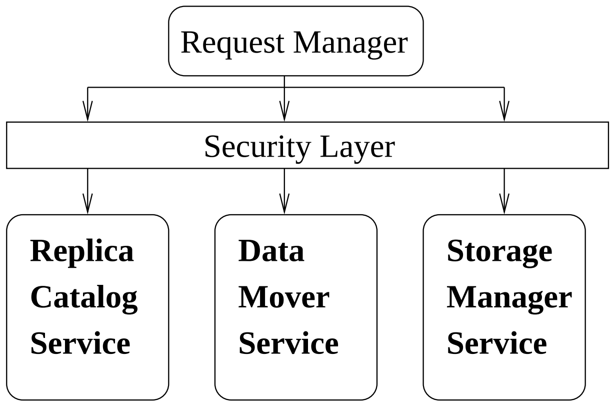

Figure 4: Overview of the GDMP architecture

4.2 Replica Catalog Service

The GDMP replication service uses a Replica Catalog to maintain a global file name space of replicas (see Section 3). GDMP provides a high-level replica catalog interface and currently uses the Globus Replica Catalog as the underlying implementation. An end-user who produces new files uses GDMP to publish information into the replica catalog. This information includes the logical file names, meta-information about the file (such as file size and modify time-stamps) and the physical location of the file. In detail, when a site publishes its files:

- These files (and the corresponding meta-information) are added to the replica catalog.
- The subscribers are notified of the existence of new files.

The Replica Catalog service also ensures a global name space by making sure that all logical file names are unique in the catalog. GDMP supports both the automatic generation and user selection of new logical file names. User-selected logical file names are verified to be unique before adding them to the replica catalog. Race conditions on the replica catalog are currently not dealt with.

Client sites interested in a new file can query the Replica Catalog Service to obtain the information required to replicate the file. Users can specify filters to obtain the exact information that they require; information is returned only about those logical files that satisfy the filter criteria. The information returned contains the meta-information about the logical file and all the physical instances of the logical file. This information can then be used as a basis for replica selection based on cost functions, which is part of planned future work. (See [VTF01] for some early ideas.)

The current Globus Replica Catalog implementation uses the LDAP protocol to interface with the database backend. We do not currently distribute or replicate the replica catalog but instead, for simplicity, use a central replica catalog and a single LDAP server for the Replica Catalog service. In the future, we will explore both distribution and replication of the replica catalog.

The GDMP Replica Catalog service is a higher-level object-oriented wrapper to the underlying Globus Replica Catalog library. This wrapper hides some Globus API details and also introduces additional functionality such as search filters, sanity checks on input parameters, and automatic creation of required entries if they do not already exist. The high-level API is also easier to use and requires fewer method calls to add, delete, or search files in the catalog.

We have already tested the new API successfully on two independent test beds involved LDAP servers at CERN (Switzerland), Caltech (California, USA) and SLAC (California). Note that each test bed only used a single replica catalog.

4.3 Data Mover Service

In a Data Grid where large amounts of data have to be transferred from one site to another (“point-to-point replication”) we require high-performance data transfer tools. This is one of the major performance issues for an “efficient” Data Grid and is the target of the Globus Data Grid Toolkit’s GridFTP system. In Section 6, we present the results of detailed performance studies conducted with the alpha GridFTP release.

The GDMP Data Mover service, like the GDMP Replica Catalog service, has a layered, modular architecture so that its high-level functions are implemented via calls to lower-level services that perform the actual data manipulation operations. In this case, the lower-level services in question are the data transfer services available at each site for movement of data to other Grid sites.

It seemed to us that the GridFTP design addressed the principle requirements for a Data Grid data transfer primitive, in particular security, performance, and robustness. Hence, we have explored the use of GridFTP as GDMP's underlying file transfer mechanism.

The large size of many data transfers makes it essential that the Data Mover service be able to handle network failures and perform additional checks for corruption, beyond those supported by TCP's 16 checksums. Hence, we use the built-in error correction in GridFTP plus an additional CRC error check to guarantee correct and uncorrupted file transfer, and use GridFTP's error detection and restart capabilities to restart interrupted and corrupted file transfers. In the future, we will exploit GridFTP's support for "pluggable" error handling modules to incorporate a variety of specialized error recovery strategies.

4.4 Storage Management Service

In order to interface to Mass Storage Systems (MSS), the GDMP service uses external tools for staging files. For each type of Mass Storage System, tools for staging files to and from a local disk pool have to be provided. We assume that each site has a disk pool that can be regarded as a data transfer cache for the Grid and that, in addition, a Mass Storage System is available at the same site but does not manage the local disk pool directly. The staging to local cache is necessary because the MSS is mostly shared with other administrative domains, which makes it difficult to manage the MSS's internal cache with any efficiency. Thus, GDMP needs to trigger file-staging requests explicitly. This is our current environment, which might change slightly in the future.

A file staging facility is necessary if disk space is limited and many users request files concurrently. If a remote site requests a replica from another remote site where the file is not available in the disk pool, GDMP initializes the staging process from tape to disk. The GDMP server then informs the remote site when the file is present locally on disk and at that time performs automatically the disk-to-disk file transfer.

In the replica catalog, physical file locations are stored and contain file locations on disk. Thus, by default a file is first looked for on its disk location and if it is not there, it is assumed to be available in the Mass Storage System. Consequently, a file state request is issued and the MMS transfers the file to the disk location stored in the replica catalog. Note that Objectivity has an interface to HPSS [Hanu01] and the file naming convention is the same: the default location is a disk location. Some other storage management systems have a tape location as a default file location.

Note that more sophisticated space management mechanisms such as reservation of disk space are currently not available but are easy to add [FRS00]. In particular, the underlying storage system needs to provide an API for storage allocation, e.g., *allocate_storage(datasize)*. In this case, the file replication transfer might be started only if the requested storage space can be allocated.

GDMP has a plug-in for the Hierarchical Storage Manager (HRM) [Bern00] APIs, which provide a common interface to be used to access different Mass Storage Systems. The implementation of HRM is based on CORBA communication mechanisms. Some initial integration tests have been performed, with promising results. Integration with HRM will provide GDMP with a flexible approach to deal with the different MSSs being used at the different regional centers where GDMP has been installed. It also provides a cleaner interface as compared to the staging script solution, which we had employed previously.

5 Object Replication

Object replication was introduced in Section 2.1 as an alternative to file replication. In this section, we cover object replication in more depth.

5.1 Motivation for Object Replication

File replication as implemented by GDMP works well for many types of data handling in HEP. However, there is an important exception: in the later stages of a physics data analysis effort, file based replication would be too inefficient. To understand why this is the case, the physics analysis process needs to be considered more closely.

The goal in a physics experiment is to observe new physics phenomena (or observe phenomena with new levels of accuracy) in the particle collisions occurring inside the detector. The physics of two colliding particles is highly stochastic. The collision creates a highly localized concentration of energy, in which new particles may be created, with different rates of probability. Most "interesting" particles will be created with extremely low probabilities. The most resource and time-consuming task in a physics analysis effort is therefore to recognize and isolate, from all events, only those events with a collision in which a particular sought-after phenomenon occurred.

For example, in one effort one might start with a set of 10^9 stored events (which corresponds to all measurements made by the detector in one year) and narrow this down in a number of steps to a smaller set. This set might then contain 10^4 events where all corresponding particle collisions display the sought-after phenomenon.

One separates the "interesting" from the "uninteresting" events by looking at the properties of some of the stored objects for each event: in the first few steps one only needs to look at a small stored object for each event. In later steps, the information content of these small objects is exhausted and one needs to look at larger and larger objects. The subsequent data analysis steps in such an effort will thus examine smaller and smaller sets (10^9 down to 10^4) of larger and larger (100 byte to 10 MB) objects.

Consider a step somewhere in the middle, where after isolating 10^6 events, the physicist will now need the corresponding set of 10^6 objects of some type X to go further. Assume that each object of this type has a size of 10 KB, given a total object set size of 10 GB. Further assume the physicist wants replicas of these objects on a specific destination site, because this site has enough CPU power available to run the necessary analysis jobs. To support this data movement efficiently with file based replication, the Grid would next need to find a set of files with all the needed objects while this set is not larger than e.g. 20 GB. However, this set of files can very likely not be found at all! Since the requested set is only 10^6 objects of type X out of the available set of 10^9, the a priori probability that any existing file happens to contain more than 50% of the selected objects is extremely low, even if every file contains only a few objects. A smart initial placement of "similar" objects together in the same files can raise the probability, but not by very much. Furthermore, the activities of other users are unlikely to create just the right files, as the physicist just selected objects related to a completely "fresh" event set which nobody else has worked on yet.

Therefore, the only way to efficiently replicate the required objects is by using object replication, in which the right files, 10 GB in total, are created with an object copier tool first. (An alternative solution is never putting more than one object in a file, but that would make the object persistency layers in Figure 2 too inefficient, and in any case this alternative can be seen as an object replication implementation that eliminates the need for the object copier tool.)

The above sparse selection effects do not only affect file replication efficiency but also local disk access efficiency. This is the context in which they have first been studied for HEP [Holt98] [Scha99]; some of the results of this prior research have been incorporated into the object replication prototype discussed below.

5.2 Architectural Choices

Architectural solutions for the object replication problem are being investigated: some earlier results are [HoSt00], [Holt01]. The use of wide-area object granularity access and replication protocols is considered unattractive, as large wide-area overheads have been observed in existing implementations of such protocols. Wide-area networks are scarce resources for HEP, and this has driven the architecture considered for object replication. Rather than trying to build high-performance specialized file granularity replication protocols, a strategic choice was made to leverage the ongoing R&D activities on maximizing throughput for moving large (>100 MB) files over the WAN.

This architectural choice has led to the use of significant parts of GDMP and the underlying Globus services in creating object replication prototypes.

A complete object replication cycle is performed as follows:

- Objects that are needed by an application on the destination site are identified, as a group, before the application starts accessing any of them.
- The objects not yet present on the destination site are identified, and a source site, or combination of source sites, for these objects is found.
- On the source site, the needed objects are copied into a new file or files, which are then sent to the destination site.

Object copying and file transport operations are pipelined to achieve a better response time and greater efficiency. In the current prototype implementation, the application on the target site can start reading the objects from a file as soon as the file has been transferred completely. After having been transferred, the files are deleted on the source site(s). The new files on the target site are first-class citizens in the Data Grid: they too are potential object extraction sources for future object replication requests.

A global view of which objects exist where is maintained in a set of index files. These files are themselves maintained and replicated on demand using file-based replication by GDMP and Globus. At the time of writing, the current prototype does not implement such a global view yet. An important future challenge is to demonstrate scalability of this global view to a huge numbers of objects [HoSt00]. We can exploit some specific properties of the HEP data model and workloads to achieve this scalability. For example, it is possible to structure most data-intensive HEP applications in such a way that each application run specifies up front exactly which set of objects are needed. These objects can then be found in one single collective lookup operation on the global view.

5.3 Prototyping Experience

Most HEP experiments involved in Data Grid activities do not yet perform physics analysis efforts that would require efficient object replication services. The reason for this is simply that their detectors are still in the building or commissioning phase, so they only have small amounts of simulated data to analyze. On the other hand, the building of the detectors and the related data analysis software is already creating a strong demand for file replication services. As a result, GDMP is a tool that is in production use whereas Grid object replication services are still in an architectural and prototyping phase.

Initial prototyping of object replication was done to validate the architectural choices and shows little surprises. As long as the object replication server is powerful enough in terms of disk I/O and CPU resources, the object copying actions in the server do not form a bottleneck, and the server has no problems maintaining several wide area parallel FTP connections with the expected efficiency. The tuning issues for the FTP connections are not changed. Overall though, compared to a file replication server dimensioned to saturate the same amount of network bandwidth, an object replication server will need more CPU and disk I/O resources. The running of the object copier tool means a significant extra load on the operating system: it needs to process more file system I/O calls and context switches per byte sent over the network. Also the amount of traffic on the machine databus per network byte sent is increased. In situations where a single box needs to drive a very high-end network card, a degradation in network traffic handling efficiency might therefore be noticeable when compared to using the box as a file based replication server. In that case, running the object copier tool on a different box (connected via a fast disk server) might be necessary.

6 Experimental Results with GridFTP

The main motivation for our performance tests is to study the impact of TCP socket buffer size tuning on parallel (multi-flow) data transfers [Tier94, QZK99, Morr97] as well as to understand the throughput that can be achieved in realistic settings. TCP uses what it calls the "congestion window" to determine how many packets can be sent at one time. In general, a larger congestion window size leads to higher throughput. The TCP "slow start" and "congestion avoidance" algorithms determine the size of the congestion window. The maximum congestion window is related to the amount of buffer space that the kernel allocates for each socket. For each socket, there is a default value for the buffer size, which can be changed by the program using a system library call just before opening the socket. The buffer size must be adjusted for both the send and receive ends of the socket. To get maximal throughput it is critical to use optimal TCP send and receive socket buffer sizes for a particular link. If the buffers are too small, the TCP congestion window will never fully open up. If the buffers are too large, the sender can overrun the receiver, and the TCP window will shut down.

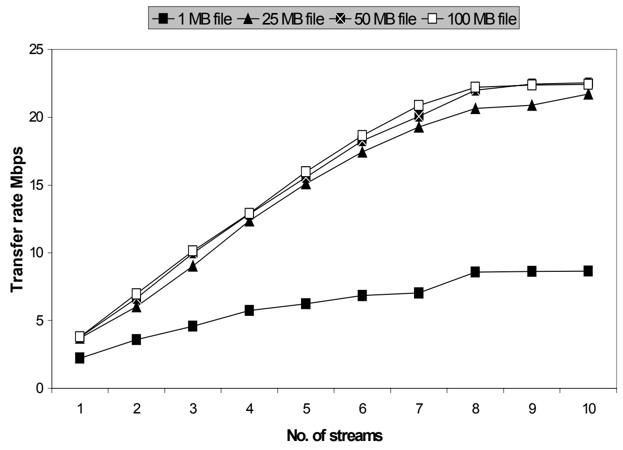

Figure 5: Transfer rates achieved for different numbers of parallel streams with GridFTP. These results are with the default TCP buffers that are typically 64 KB in the test environment. Four different files were transferred with sizes of 1 MB, 25 MB, 50 MB and 100 MB. The graph shows the curves for the larger files going up almost linearly with the number of streams, reaching a peak at around 23 Mbps for 9 streams.

TCP buffer tuning is a good way to increase throughput on a high-speed WAN link, but we face the obvious problem of having to determine the "optimal" value for the TCP window size. The optimal window size needs to be calculated by accurate measurements of link delay and bandwidth. Alternatively, we can use parallel data streams [Berk01]. The Globus GridFTP library supports both of these facilities.

We carried out a detailed study of GridFTP parallel transfer performance. The test environment consisted of a 45 Mbps link between CERN and ANL with a RTT of 125 milliseconds. The GSI enabled WU-ftpd server version 0.4b6 was used as the test server. Test programs "extended_get" and "extended_put" from the Globus distribution were the chosen clients. These programs test the parallel stream and buffer tuning features of GridFTP. Since we have seen similar behaviour for the GridFTP put and get functions, we present only results for get. The results presented in Figures 5 and 6 lead us to the following conclusions. First, proper TCP buffer size setting is the single most important factor in achieving good performance. The performance obtained from 10 streams with untuned buffers can be achieved with just 2-3 streams if the tuning is proper. Next, note that 2-3 tuned parallel streams will gain an additional 25% performance over a single tuned stream. Finally, note that it is possible to get the same throughput as tuned buffers using untuned TCP buffers with enough parallel streams.

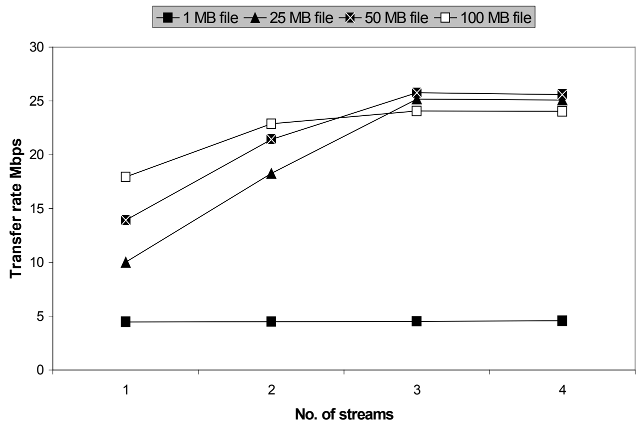

Figure 6: The same experiments as are presented in Figure 3, but with TCP buffers tuned to 1 MB. Results are similar, except that peak performance is achieved with just 3 streams.

To determine the optimal TCP buffer size, we use following standard formula, as described in [Tier00]:

optimal TCP buffer = RTT x (speed of bottleneck link)

The Round Trip Time (RTT) is measured using the Unix ping tool, and the speed of the bottleneck link is measured using pipechar [Jin01], a new tool from LBNL designed for this purpose.

A simple method to the optimal number of parallel streams is not yet known. With too many streams, there is a possibility of overloaded the receiving host, and causing network congestion. We typically run multiple iperf [IPERF] tests with various numbers of streams, and compare the results. We usually find that 4-8 streams is optimal.

7 Summary

The GDMP replication service has been enhanced with more advanced data management features, including namespace and file catalog management, efficient file transfer, and preliminary storage management. New architectural components have been discussed. A detailed study of the Globus GridFTP implementation is presented. We also analyzed more advanced object-based replication techniques and explained how these techniques can be structured in terms of file replication mechanisms.

Acknowledgements

We thank Rajesh Kalmady for his good work on WAN network tests. We also acknowledge numerous discussions on the topics covered here with colleagues, in particular Ann Chervenak, Harvey Newman, Carl Kesselman, and Steven Tuecke.

Work at ANL was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; by the National Science Foundation; and by the NASA Information Power Grid program.

References

[Allc01] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. *18th IEEE Symposium on Mass Storage Systems and 9th NASA Goddard Conference on Mass Storage Systems and Technologies,* San Diego, April 2001.

[Allc01b] W. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Lee, A. Sim, A. Shoshani, B. Tierney, B. Drach, D. Williams, High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies, Preprint, Argonne National Laboratory, 2001.

[BMRW98] C. Baru, R. Moore, A. Rajasekar, M. Wan. The SDSC Storage Resource Broker. *CASCON'98 Conference*, 1998.

[Bern00] L. M. Bernardo, A. Shoshani, A. Sim, H. Nordberg, Access Coordination of Tertiary Storage for High Energy Physics Application, *17th IEEE Symposium on Mass Storage Systems and 8th NASA Goddard Conference on Mass Storage Systems and Technologies,* Maryland, USA, March 27-30, 2000.

[Berk01] Data Intensive Distributed Computing Group, Lawrence Berkeley National Laboratory, Tuning Guide for Distributed Application on Wide Area Networks, http://www-didc.lbl.gov/tcp-wan.html, March 2001.

[Bres99] L. Breslau,, P. Cao, L. Fan, G. Phillips, S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. *Proceedings of IEEE Infocom*, 1999.

[Cher00] 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 Data Sets, *J. Network and Computer Applic*ations, 2000.

[EDG01] European Data Grid project: http://www.eu-datagrid.org

[FKT98] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. A Security Architecture for Computational Grids, *ACM Conference on Computers and Security*, 83-91, 1998.

[FoKe99a] I. Foster, C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann, 1999.

[FoKe99b] I. Foster, C. Kesselman, The Globus Toolkit, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann, 259-278, 1999.

[FRS00] I. Foster, A. Roy, V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. In *Proc. 8th International Workshop on Quality of Service*, 2000.

[GDMP01] GDMP web page: http://cmsdoc.cern.ch/cms/grid, February 2001.

[GriP01] Grid Physics Network (GriPhyN): http://www.griphyn.org, February 2001.

[Hanu01] A. Hanushevsky. Obejectivity/DB Advanced Multi_threaded Server (AMS) www.slac.stanford.edu/~abh/objy.html , April 2000.

[Holt98] K. Holtman, P. van der Stok, I. Willers. Automatic Reclustering of Objects in Very Large Databases for High Energy Physics, *Proc. of IDEAS '98*, Cardiff, UK, 1998.

[HoSt00] K. Holtman, H. Stockinger. Building a Large Location Table to Find Replicas of Physics Objects. *Computing in High Energy Physics (CHEP 2000)*, Padova, Italy, Febr. 2000.

[Holt00] K. Holtman. Object Level Physics Data Replication in the Grid. *VII International Workshop on Advanced Computing and Analysis Techniques in Physics Research ACAT'2000*, Chicago, USA, October 16-20, 2000.

[Hosc00] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, K. Stockinger, Data Management in an International Data Grid Project, *1st IEEE/ACM International Workshop on Grid Computing (Grid'2000)*, Bangalore, India, 17-20 Dec. 2000.

[IPERF] iperf: http://dast.nlanr.net/Projects/Iperf/index.html

[Jin01] G. Jin, G. Yang, G., B. Crowley, D. Agarwal. Network Characterization Service, *10th IEEE Symposium on High Performance Distributed Computing* , San Francisco, CA, August 7-9, 2001

[Karg99] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, Y. Yerushalmi. Web Caching with Consistent Hashing. *8th International World Wide Web Conference*, 1999.

[Linn00] J. Linn. Generic Security Service Application Program Interface Version 2, Update 1, IETF, RFC 2743, 2000. http://www.ietf.org/rfc/rfc2743.

[Moor99] R. Moore, C. Baru, R. Marciano, A. Rajasekar, M. Wan. Data-Intensive Computing. In I. Foster and C. Kesselman, eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 105-129, 1999.

[Morr97] Morris, R., TCP Behavior with Many Flows. *IEEE International Conference on Network Protocols*, IEEE Press, 1997.

[Newm00] H. Newman. Worldwide Distributed Analysis for the Next Generations of HENP Experiments. *Computing in High Energy Physics*, February 2000.

[Objec01] Objectivity, Inc. http://www.objectivity.com, February 2001.

[PPDG01] Particle Physics Data Grid (PPDG): http://www.ppdg.net, February 2001.

[QZK99] L. Qiu, Y. Zhang, S. Keshav, On Individual and Aggregate TCP Performance. *7th International Conference on Network Protocols*, 1999.

[SaSt01] A. Samar, H. Stockinger. Grid Data Management Pilot (GDMP): A Tool for Wide Area Replication, *IASTED International Conference on Applied Informatics (AI2001)*, Innsbruck, Austria, February 2001.

[SaMo00] H. Sato, Y. Morita. Evaluation of Objectivity/AMS on the Wide Area Network, *Computing in High Energy Physics (CHEP 2000)*, Padova, Italy, Febr. 2000.

[Sha99] M. Schaller. Reclustering of High Energy Physics Data. *Proc. of SSDBM'99*, Cleveland, Ohio, July 28-30, 1999.

[Stock01] H. Stockinger. Distributed Database Management Systems and the Data Grid. *18th IEEE Symposium on Mass Storage Systems and 9th NASA Goddard Conference on Mass Storage Systems and Technologies*, San Diego, April 17-20, 2001.

[Tier94] B. Tierney, W. Johnston, L. Chen, H. Herzog, G. Hoo, G., Jin, J. Lee. Distributed Parallel Data Storage Systems: A Scalable Approach to High Speed Image Servers. *ACM Multimedia 94*, 1994.

[Tier00]: B. Tierney. TCP Tuning Guide for Distributed Application on Wide Area Networks, *Usenix ;login*, Feb. 2001.

[Tewa99] R. Tewari, M. Dahlin, H. Vin, J. Kay. Design Considerations for Distributed Caching on the Internet, *19th IEEE International Conference on Distributed Computing Systems*, 1999.

[VTF01] S. Vazhkudai, S. Tuecke, I. Foster. Replica Selection in the Globus Data Grid, *IEEE International Symposium on Cluster Computing and the Grid (CCGrid2001)*, Brisbane, Australia, May 2001.