Grid Data Mirroring Package (GDMP)

Heinz Stockinger[a], Asad Samar[b], Shahzad Muzaffar[c] and Flavia Donno[d]
[a]*CERN, European Organization for Nuclear Research, Geneva, Switzerland*
E-mail: Heinz.Stockinger@cern.ch
[b]*California Institute of Technology, Pasadena, California, USA*
E-mail: Asad.Samar@cern.ch
[c]*Fermi National Laboratory, Batavia, Illinois, USA*
E-mail: muzaffar@fnal.gov
[d]*INFN Pisa, Italy*
E-mail: Flavia.Donno@pi.infn.it

The GDMP client-server software system is a generic file replication tool that replicates files securely and efficiently from one site to another in a Data Grid environment using Globus Grid tools. In addition, it manages replica catalogue entries for file replicas and thus maintains a consistent view on names and locations of replicated files. Files to be transferred can be of any particular file format and GDMP treats them all in the same way. However, for Objectivity database files a particular plug-in exists. All files are assumed to be read-only.

1. Introduction

File replication is one of the essential features of a Data Grid [2] where potentially large amounts of data need to be stored in multiple copies (replicas) at several, world-wide distributed sites. Although replication is a well established field in the distributed computing as well as in the database communities, dealing with file replication in a Grid is a challenging task. The GDMP software system is a file replication tool satisfying several requirements of current Data Grid users and thus provides practical solutions to the file replication problem.

The GDMP [10,13,14] project (originally called Grid Data Management Pilot) was started in early 2000 as a pilot project to evaluate the Globus toolkit (tm) [5], take useful features for a file replication system and produce a prototype to be evaluated in a real production environment. In early versions (until release 1.2.2 of the software), GDMP was restricted to replicate Objectivity files but in the new release 2.0 any kind of file format is accepted.

The driving force for the research and development on GDMP was twofold: on the one hand, at CERN – the European Organization for Nuclear research located near Geneva in Switzerland – first efforts for the European DataGrid project [3] were under way and first state-of-the-art studies in Grid computing needed to be done. On the other hand, the CMS experiment, one of four next generation High Energy Physics (HEP) experiments at CERN, required data transfer tools for the distributed production of simulated data, in particular for the High Level Trigger studies. Once the DataGrid project was kicked off in January 2001, the experience gained from GDMP was used and the current version of the software system is part of the DataGrid software and in use in the first DataGrid testbed. Currently, the software development process is well advanced and the project is now a collaboration between the European DataGrid (in particular the Data Management work package) and the Particle Physics Data Grid (PPDG) [9].

With respect to DataGrid data management architecture [12], GDMP is a replica manager with additional high-level, application specific functionality like integration of Objectivity database files to an Objectivity federation, a notification system for remote sites and an interface to mass storage systems.

The paper is organised as follows: We first describe the requirements for a file replication system and give details on the GDMP architecture including all Grid components that are used from the Globus Toolkit 2.0 Alpha release. Next, we outline the data replication process in detail and describe tools and features that are used to gain the task. Section 5 elaborates on applications and interfaces and outlines the usage of the current GDMP release version 2.0. Some performance results are given in Section 6 and we report on our experience on data transfer. Finally, we conclude the paper with some concluding remarks.

2. Requirements and basic functionality

The main propose of CERN and thus the CMS experiment is to study the origin of matter. For this pur-

Scientific Programming 10 (2002) 121–133
ISSN 1058-9244 / $8.00

pose, large particle accelerators and detectors are used and several Petabytes of data will be stored a year starting from 2006. Although the CMS detector is scheduled to take data in year 2006, simulated production is done already now in order to test and prepare physics software and storage systems for the upcoming data challenge of several Petabytes of data a year. As regards the High Level Trigger study in CMS, simulated data is created in several regional centres like Fermilab (Chicago, USA), Pisa (Italy), Moscow (Russia), Lyon (France), Bristol (UK) in the CMS Data Grid and then needs to be transferred to other sites in the Grid. Data is either stored in Objectivity databases or in special, HEP specific file formats.

Although Objectivity provides a synchronous data replication option, it is not optimised for wide-area replication and thus a replication system was required that replicates Objectivity files as well as files of any other file format. Furthermore, synchronous replication bears the danger of deadlocks and the entire system might be blocked if remote replica sites are not available [8]. Instead, asynchronous replication mechanisms are required in a wide-area DataGrid and GDMP provides such a replication model.

2.1. Distributed data production in CMS

The preparations of distributed data production have been going on for a couple of years. A data transfer production system based on Perl scripts, HTTP and secure shell copy transfers has been put in place [17]. The purpose of these scripts is to allow a user to see the contents of an Objectivity's federated database catalogue from anywhere with WWW access, without the need to run any part of the Objectivity software locally. Furthermore, the script software allows for data transfer of not only Objectivity but any kind of file, for instance HEP specific Zebra-fz or Root files. These Perl scripts have been the initial input for the architecture and design of GDMP. The idea was to extend the functionality by introducing a security environment and providing a fully automated replication mechanism based on a Grid infrastructure using Globus. Originally, GDMP was a Grid-enabled successor of these Perl scripts but then has been extended with several replication features, preliminary mass storage management and meta-data management for replicated files.

Since October 2000, the first versions of GDMP are in use in the CMS production environment. We have proven that Data Grid tools can be used efficiently in a production environment and thus have made a pioneer step towards the vision of a Data Grid. What is more, GDMP is now a generic file replication tool and available in the DataGrid software to serve the HEP, Bio Informatics and Earth Observation communities.

2.2. A basic usage scenario

GDMP's new name[1] derived from the fact that the original status of a pilot project has been extended and now better identifies the functionality of the replication service.

The core functionality of the software system is to (partly) mirror one data repository to remote sites in a Data Grid. When new files are added at one site, tools have to detect which files are new in the repository and notify other sites about the availability and location of a set of new files. Figure 1 depicts a simple replication (mirroring) scenario in a Data Grid with four sites.

For instance, at Fermilab a set of three new files is created and should be made available to remote sites. A user at CERN shall now have an automatic tool that finds out about newly created files, transfers them to a specific storage location and then possibly updates local file catalogues (as it is the case for Objectivity files). Since Objectivity provides an internal catalogue for all local files, newly created files can be easily identified. For files of other data formats, a site catalogue is required which holds all files that are stored within a site and should be published to remote sites in the Grid. At each site, files normally reside on disks but may be partially available also on tapes and accessible via mass storage systems.

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 catalogue 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 catalogue. 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 catalogue information. Simply put, this is done by a GDMP server running at each site where files are produced and possibly replicated.

[1] Grid Data Mirroring Package instead of Grid Data Management Pilot

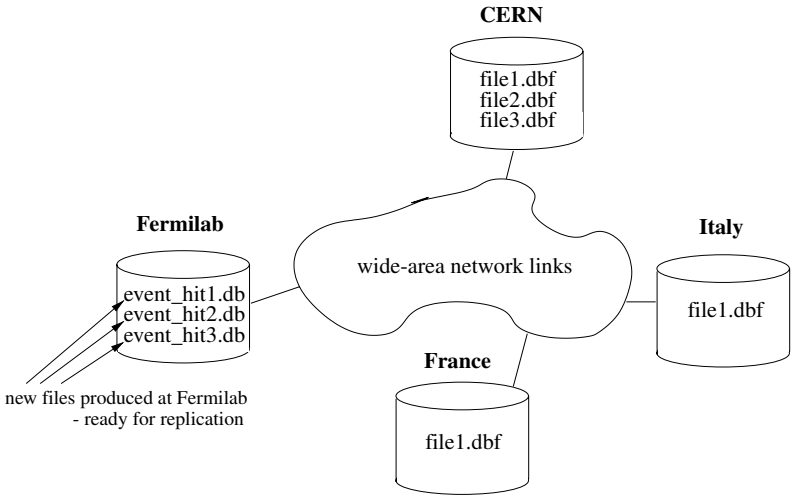

Fig. 1. A sample Data Grid with 4 sites where new data is produced and replicated.

2.3. Directories and file replication

GDMP manages files stored in a particular *storage root directory* on a local disk or mounted disk pool. It requires all physical files to be stored in this directory structure which can be configured for each GDMP installation and thus for each storage system. Note that we distinguish between "flat files"[2] (this can be any arbitrary file format like ROOT, ZEBRA, etc.) and Objectivity files which require particular replication steps. GDMP also requires two different storage root directories for these two file formats.

We start with an example. We assume that a large disk pool is mounted on a host `host1.cern.ch`. Files are stored in the directory `/data/run1/`. In the directory `run1` several subdirectories can exist and a possible directory layout is as follows:

```
/data/run1/day1/file1
/data/run1/day1/file2
/data/run1/day1/file3
/data/run1/day2/fileA
/data/run1/day2/fileB
```

GDMP requires a common root path for the directory structure since it manages several files in the replication process. In our example the common directory and thus the storage root directory for flat files is `/data/run1`.

[2]Note that the term flat file is used to distinguish between a generic file and an Objectivity file and does not say anything about the structure within the file.

The main task of GDMP is to mirror the directory structure of one storage system (called *Storage Element* in DataGrid terminology) to another. Thus, a storage root directory is required on both sites that participate in the mirroring process. Note that the storage root directory does *not* have to be identical on all Storage Elements but can be chosen and configured based on the local directory structure. If we now assume that all the files above need to be replicated to a Storage Element at Fermilab, the destination host first needs to set up a storage root directory. For example, on `host1.fnal.gov` (at Fermilab), the storage root directory has the value `/largedisk/cms/production/run1`. The GDMP replication process now can replicate files from the storage root directory on the source machine to the one on the destination machine.

2.4. Filenames

When files are replicated, identical files (replicas) exist at multiple locations and need to be identified uniquely. A set of identical replicas is assigned a *logical filename* (LFN) and each single physical file is assigned a *physical filename* (PFN). In [12] we also include a *transfer filename* (TFN) but we do not discuss it further and we will assume that all PFNs have the complete paths used by the Storage Element's file system to refer to files resident on disk. The PFN also contains the host name i.e. the domain name of the Storage Element [12] where the file is located and accessible via a Grid file transfer tool such as a GridFTP [1] server.

A typical example of a physical filename under the above assumptions is as follows:

```
pfn://host1.cern.ch/data/run1/day1
/file1
```

We observe that when PFNs are used with GDMP, the prefix "`pfn://`" is not required. Once the file is created and the PFN is available, it can be inserted to a replica catalogue that provides a global name space for file replicas. Note that for this GDMP version we use a single replica catalogue located at a single host (rather than a distributed replica catalogue as presented in [12]). In addition to the PFN, a logical filename must be assigned to the physical file. In the current version of the Globus replica catalogue [1], the LFN is equal to the last component of the PFN, i.e., to the file name including parts of the directory structure. This is a current restriction that will be removed in future versions. Thus, the logical filename is created automatically by GDMP from the physical filename complying with the implicit mapping enforced by the replica catalogue and for the physical filename in the example above it looks as follows `day1/file1`.

2.5. The file replication process

The entire file replication process does not only contain a simple file copying from one site to another but contains several steps and requires to keep track of filenames and locations. The latter is achieved through a replica catalogue service. Based on the file type or format (Objectivity file, Oracle file, plain text file, etc.) a successful file replication process consists of the following steps:

- *pre-processing*: This step is specific to 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 catalogue.

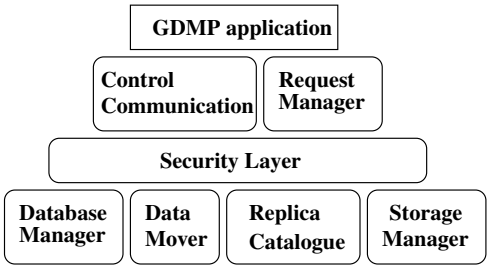

Fig. 2. The layered GDMP architecture.

- *insert the file entry into a replica catalogue*: 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 (producer) 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.

3. Architecture

We now give details on the architecture, the components used by GDMP and some design considerations. In principle, the GDMP software consists of several modules that in turn can use established Grid services like a replica catalogue service or a file transfer service.

GDMP has a layered architecture model (see Fig. 2) and uses several Globus libraries. The boxes represent single modules that are explained in more detail in the following subsections. Each of the GDMP client applications (command lines tools for publishing site catalogues, replicating files, etc.) as well as the GDMP server use several of the modules.

3.1. Control communication

This module takes care of the control communication between the clients and the servers, and uses the Globus IO library [5] as the middle-ware and builds high level functionality on top. It takes care of the intricacies related to socket communication over the wide area between nodes with heterogeneous architectures. The functionality includes starting and stopping the server, connecting and disconnecting the client to and from the

server and sending and receiving messages at both the client and server ends. This module provides services to other modules.

Globus IO is a thin layer on top of basic socket communication and provides APIs for easily managing sockets over TCP or UDP. The Globus Data Conversion library [5] is used for providing high level interfaces that are similar to RPC calls.

3.2. Request manager

Every client-server system has to have a way to generate requests on the client side and interpret these requests on the server side. The Request Manager module does exactly that. It contains several request generator/handler pairs and more can be added for customised use. This module is based on the Globus Data Conversion library which provides methods to convert data between different formats supported by variable machine architectures. The requests are generated on the client side by filling a buffer with the appropriate function handler, which is to be called on the server end, and any arguments required by that function. On the server side this buffer is unfolded and the data types are converted according to the local architecture. Finally, the respective function is called with the given arguments. The Request Manager Module basically mimics a limited Remote Procedure Call (RPC) functionality with the advantage of being light-weight and extensible.

3.3. Security layer

Security is one of the main concerns in a Grid. Allowing people from outside one's domain to use the resources is a big issue for most organisations. Sensitivity of data and unauthorised use of network bandwidth to transfer huge files are the main security issues we are dealing with. This is done by the security layer. It is based on the Grid Security Infrastructure (GSI) [6] which is an implementation of the Generic Security Service (GSS) API. It uses the Public Key Infrastructure as the underlying mechanism. The security module provides methods to acquire credentials, initiate context establishment on the client side and accept context establishment requests on the server side (context is established to authenticate the client), encrypting and decrypting messages and client authorisation. The server authenticates and authorises any client before processing its request, hence, the software protects a site from any unwanted file transfers and blocks any unauthorised requests. Consequently, every client server communication is done via secure communication. This also applies for the file transfer itself. However, data encryption for file transfer is not used.

3.4. Database manager

This module interacts with the actual database management system if database files are replicated. The module relies on the APIs provided by the particular data storage system being used. In our case the DBMS of choice is Objectivity/DB, hence we use Objectivity's command line tools to implement this module. If a new data type, i.e. a new database management system, is used, this is the only module which has to be swapped by the one which can interact with the specific DBMS. The functionality includes

- create Objectivity federation (including database schema)
- schema upgrade based on schema of a remote Objectivity federation
- retrieve the database catalogue (or file catalogue), containing information about the files currently present in the database,
- attach files to the DBMS once they arrive on the client side where they are validated.

3.5. Data Mover

The main purpose of GDMP is to replicate files over a wide-area network in an automatic, efficient and fault tolerant way. This module is responsible for the actual transfer of files from one location to another.

The Data Mover uses the GridFTP client library of Globus [1]for file transfer and relies on GridFTP servers (based on the WU-FTP server with Globus modifications for GridFTP). Since GridFTP uses the Grid Security Infrastructure (GSI), we have the same security mechanism for both the Control Communication and the Data Mover. The functionality includes client authentication and authorisation before the transfer starts, transferring files to and from the server, validating the transferred files using the file size and checksum information, resuming the file transfer from the latest checkpoint after a network failure,[3] using a progress meter

[3] In an early version of GDMP we have used a different GSI enabled FTP client library that provided this feature. It is also available in GridFTP and we plan to use it as soon as it is a public release.

to output the progress during a file transfer and finally logging all file transfers that have been completed.

Point-to-point replication is on of the major performance criteria for a large Data Grid. We require high-performance data transfer tools which is the target of the Globus Data Grid toolkit's GridFTP [1] system. GridFTP allows for parallel and striped transfer streams, TCP tuning and is based on GSI. In [14] we present the results of detailed performance studies conducted with the an early alpha GridFTP release.

The GDMP Data Mover 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.

3.6. Replica catalogue

GDMP uses the Globus replica catalogue service [1] to maintain a global file name space of replicas. GDMP provides a high-level replica catalogue interface and currently uses the Globus replica catalogue as the underlying implementation. An end-user who produces new files uses GDMP to publish information into the replica catalogue. This information includes the logical file names, logical file attributes (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 file attributes) are added to the replica catalogue.
- The subscribers are notified of the existence of new files (more details on the subscription model are given below).

The replica catalogue module also ensures a global name space by making sure that all logical file names are unique in the catalogue (this is achieved through the Globus Replica Catalogue library). 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 catalogue but we need to map the logical filenames to the ones accepted by the Globus Replica Catalogue. Race conditions on the replica catalogue are currently not dealt with.

Client sites interested in a new file can query the replica catalogue 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.

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

The GDMP replica catalogue module is a higher-level object-oriented wrapper in C++ to the underlying Globus replica catalogue library in C. 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 catalogue.

3.7. Storage manager

In order to interface to Mass Storage Systems (MSS), the GDMP server 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 externally. 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 a 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 initialises 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 catalogue, 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 stage request is issued and the MSS transfers the file to the disk location stored in the replica catalogue. Note that Objectivity has an interface to HPSS [11] 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 [7]. 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.

4. The data replication process and policies

We present the data replication process and also introduce some GDMP client applications which are then described in more detail in the next section.

4.1. File catalogues

GDMP uses a few catalogues that are used for internal book keeping and monitoring of the replication process. We elaborate on how GDMP manages the file transfer and illustrate the data flow by a producer-consumer example. The producer is the site where one or several files are written, and the consumer is the site that wants to replicate (receive) these files locally. Once the producer has finished writing a set of files (or just a single file), every single file needs to be registered in a *local file catalogue* which only contains files that are available at the local Storage Element. The catalogue contains the physical filename and logical file attributes like logical filename, file size, creation time, CRC checksum, file type. The local file catalogue is hidden from outside users and is thus only visible to the local GDMP server. The client application `gdmp_register_local_file` is used for inserting files to this catalogue.

At a certain point in time, the producer can decide to publish its local files to other Grid sites using `gdmp_publish_catalogue`. In detail, all file entries of the local file catalogue are written into the replica catalogue[4] and also sent to subscribed consumers at remote sites. A list of all newly published files and their related information is written to a local *export catalogue*. The consumer creates an *import catalogue* where it lists all the files that are published by the producer and have not yet been transfered to the consumer site. The import catalogue holds the host name of the FTP server and all related physical and logical file information for each file. Figure 3 illustrates this model graphically and shows that a GDMP installation (including client and server applications) is required on each Storage Element taking part in the data replication process.

When files are published, all required file information is read from the local file catalogue. Since the catalogue holds file attributes like size, during the execution of the publish command the files to be published do not need to reside on their physical location on disk but can already have been staged to a mass storage system. Note that files have to be at the disk location when `gdmp_register_local_file` is called since file size and CRC check sum are automatically created by GDMP and then stored in the local file catalogue.

4.2. Subscription and replication model

Although replica information is available in the replica catalogue, one GDMP site can subscribe to a remote site in order to get notified when new files are created. Thus, a site keeps a *host_list* catalogue (see Fig. 3) where all hosts are listed that are subscribed to newly created local files.

The currently implemented replication policy is an asynchronous replication mechanism with a subscription model. A producer can choose at what time new files are written into the export catalogue and thus made publicly available for consumers. Hence, the producer can decide the degree of data consistency by delaying the publication of new files. In the example above we only have one consumer for demonstration purpose. In reality, the number of consumers can be large and depends on the number of sites in the Grid. The subscription model enables that the subscribed consumers get informed immediately when new files are published in the export catalogue. Each consumer that wants be be notified about changes in the producer's export catalogue, subscribes to a producer. Depending on the degree of interest in a producer's data, consumers might

[4] An insertion to the replica catalogue can also be disabled.

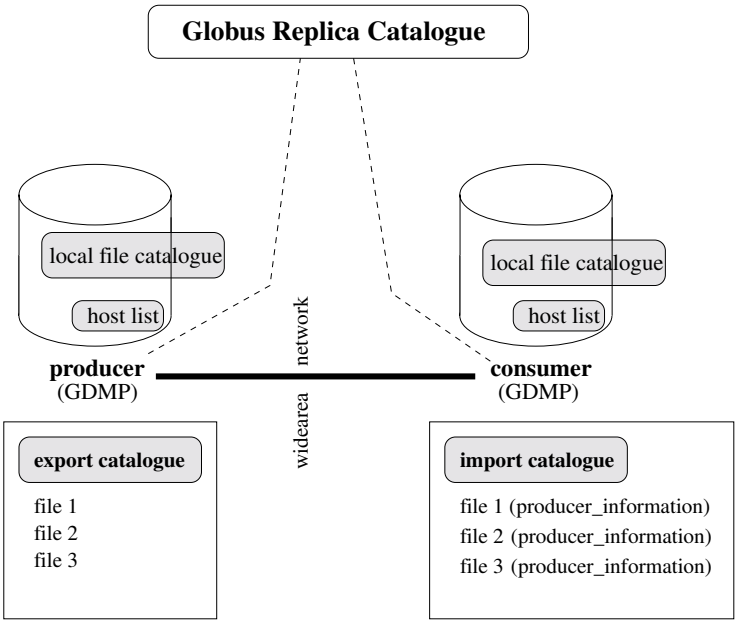

Fig. 3. The role of the local file, export, import catalogues.

want to subscribe to only some of the producers in the Grid.

Since the data exchange has to be done in a controlled and secure way, a consumer first has to be a "registered Grid user" at the producer site, i.e. the user has to be added to the grid-mapfiles[5] of the producer sites. These files contain all the users who are allowed to talk to GDMP servers running on a site. Thus, a producer has total control over who subscribes to and transfers files from its site. Once this is done, a consumer is allowed to subscribe to the producer site. The producer then adds the new consumer and its related information in the host_list catalogue.

The consumer can decide when to start the file transfer from the producer to the consumer site with the tool `gdmp_replicate_get`. The tool reads the import catalogue and starts FTP sessions and interactions with remote GDMP servers to get the necessary file.

In our example above we distinguish between a producer and a consumer site in order to illustrate the data flow. However, each site in a Data Grid can produce new files and get files from other sites, i.e. a site can be producer and a consumer. Thus, each site manages an import catalogue where it stores available file information from other sites, and an export catalogue where it publishes files created by itself.

[5]Globus specific file which is used for user authentication.

4.3. Notification system

GDMP provides a powerful notification mechanism which can be used to monitor the replication process as well as the publishing of new information. Configurable scripts are used for interfacing user specific applications.

When a user publishes a local file catalogue with `gdmp_publish_catalogue`, a remote server gets notified and calls a configurable script which can then be used by external programs to start a data transfer request. In principle, `gdmp_replicate_get` can be executed and a fully automatic replication process can be set up. On the other hand, a similar notification script is called at the producer site when a consumer has successfully replicated a file. Thus, producers can keep track of consumers requesting and replicating files and can delete files again if local storage space is required.

4.4. System states for file replication process

In the entire replication process, we distinguish several system states which can be used to check the current status of a file transfer and discover possible problems. The following file transfer states can be distinguished:

- ongoing file transfer
- file transfer has succeeded
- file has been validated (CRC checksum)

- file has been registered to the local file catalogue.
- a remote site has been notified about the correct file transfer (including validation and registration).

In addition to the successful states, we also have a few error states:

- file validation failed
- file is not registered to the local file catalogue
- file notification failed
- error during the attachment of Objectivity files

4.5. Partial replication: Filtering files

GDMP allows a partial-replication model where not all the available files in a federation are replicated. In other words, one or several filter criteria can be applied to the import and/or export catalogue in order to sieve out certain files. For instance, a site only wants to make files publicly available that contain the word "Muon" in the filename. Hence, the export catalogue, which contains all the files that a site wants to publish, has to be filtered and files that do not contain the filter criterion are deleted. A site that wants to get files from other sites can as well choose which files it wants to get. In this case the filter has to be applied on the import catalogue. This allows for a partial replication model where the producer as well as the consumer can limit the amount of files to be replicated.

4.6. Fault tolerance and failure recovery

In a distributed system we can identify several sources for errors. On the one hand, there are hardware errors like a physically broken network cable, a broken network card, processor or any other piece of hardware. On the other hand, a part of the software system can have a failure. For instance, the FTP server dies, the file system crashes or some other part of the GDMP software does not work properly. In any of these cases the connection between the distributed clients and servers is broken. We refer to such a failure as the *connection is broken* and do not distinguish between the different reasons for this failure. We emphasise that the communication is broken and the message or control flow cannot be continued. For instance, in case of a broken connection anywhere between site A and site B, site A cannot publish its catalogue to site B.

In the current version of GDMP, each site is itself responsible for getting the latest information from any other site. This is also expressed by the subscription system, where a site has to explicitly subscribe to another site in order to get the file catalogue. Furthermore, when a site recognises a local error which has caused the broken connection, this site has to request the required information from the peer site. A site which publishes information to a subscribed site does not re-send information nor logs that a site could not receive the information. A site can retry to send the message again to the destination site within a particular time frame which can be set by a timeout parameter. If the re-sending fails again, the sending site stops trying to contact the sites and hands over the responsibility to the destination site to recover from the broken connection.

To sum up, the policy is the following. Each site has to be aware of its state (connection okay or broken). Then it has to search for the origin of the broken connection. If it detects that the error is on the local site, it has to recover otherwise the peer site is responsible for failure recovery.

A site can be unavailable for several hours or even days. Meanwhile several producers can have created and published files, and the entries in the export catalogues may already have been overwritten. However, published replica information is also available in the replica catalogue. A consumer can recover from the site failure by issuing the command `gdmp_get_catalogue`. Once a producer site has published its catalogue, the catalogue is available to be transferred to any subscribed consumer. GDMP at the consumer site then compares the consumer and producer catalogue and creates the necessary information in the consumer's import catalogue. Possible multiple appearances of files are deleted in order to keep the import catalogue's entries unique.

Only the file which is currently been sent when the network connection breaks has to be resent. Since the FTP client has a "resume transfer" feature, not even the whole file has to be transfered but only the part of the file that is still missing since the last check point in the file. This allows for an optimal utilisation of the bandwidth in case of network errors.

Other possible errors: A site may publish a file several times and export it to other sites. In order to have files only uniquely transferred, each site checks automatically if the file in the local import catalogue does not already appear in the federation catalogue. Furthermore, several sites can publish the same file to a specific site two times. Consequently, on creation of the import catalogue, the system checks if the file to be entered is unique. Only if this is the case, a new file is inserted into the import catalogue.

5. GDMP applications and interfaces

In this section we go into more details on concrete GDMP tools and their usage. The client-server software consists of a GDMP server that needs to be installed and running on each site taking part in the data replication process and several client applications.

5.1. GDMP server

The GDMP server needs to run constantly and serve several user requests from GDMP client applications. We do not expect a very high load on the server (less than three requests per second) and thus do not require a high performance server, but it needs to be robust.

Currently, a GDMP server is configured to run via the Internet daemon (inetd) and can be started on any user defined port. Simply put, the inetd provides Internet service management for a network. It listens on certain ports and calls other servers or daemons for serving the request. As regards GDMP, we register a certain port, e.g. port 2000, with the Internet daemon and when a GDMP client connects to the machine via a socket connection, the Internet daemon takes the request on port 2000, starts the GDMP server and passes all the socket information to the GDMP server. The GDMP server in turn handles the client request. By default, GDMP servers are supposed to run on port 2000 but can be configured to run on any other port that is not hidden by a firewall.

All interactions between GDMP clients, the GDMP server and the underlying FTP server are done via GSI. In detail, every client request that comes over a socket first needs to be authorised and authenticated (the local grid-mapfile is used for that) and then the Request Manager module serves the client request. Like any client application using Globus, also a server requires a security proxy to be running. The server uses a dedicated server certificate which is included in the GDMP software distribution. Thus, when a server is started, the required Grid proxy is gained automatically. Any output or error message from the server is logged in a corresponding log file.

5.2. GDMP client applications

GDMP offers a set of client applications to register new files to a local catalogue, publish a set of newly created files to subscribed hosts, start the actual file replication process, and subscribe to a remote GDMP server. In addition, the software package contains a few other administration tools. We describe briefly the basic functionality and refer to the GDMP User Guide [16] for further details. We illustrate the sequence of client operations that is necessary for the replication process:

1. All GDMP client applications need to use GSI and thus require a X.509 certificate issued by a trusted certificate authority (either Globus, DataGrid or any national authority). Before any client application can be started, a client has to have a Globus proxy running[6] in order to start the authentication process handled by the GDMP security module internally. We use Globus' *single login* procedure: once a client has successfully got the proxy on one machine, requests can be sent to any GDMP server (given that the X.509 distinguished name is included in the remote server's grid-mapfile) without any further password to be entered.
2. Once the proxy is gained, the GDMP sites need to subscribe to each other. With a subscription, one site expresses its interest to get notified about newly created files. The following client application `gdmp_host_subscribe` has to be used providing remote host name as well as the remote port name as parameters. In order to be sure that a remote server is running and listening on the correct port, the tool `gdmp_ping` can be used to probe the remote server.
3. In the next step, files need to be registered in the local file catalogue using the command `gdmp_register_local_file` which registers either individual files or entire directories.
4. Once files are registered locally, they can be published to all subscribed sites with the tool `gdmp_publish_catalogue`. Note that this does not include any file transfer but only replica and location information are exchanged.
5. At the remote site, a server receives a notification message about all new files and can also further call a script locally. In principle, once a remote server is notified, the file transfer process can be started with `gdmp_replicate_get` and all files from the import catalogue are requested from remote sites and transferred to the local site. All transfer information like file size and network utilisation is logged in a log file. Several parallel

[6] A proxy can be gained via the Globus tool `grid-proxy-init`.

clients can be started too in order to optimise network traffic for un-tuned network connections. [7]

For all the client commands filters can be applied and one has to distinguish between flat files and Objectivity database files. Filters on import and export catalogues can be assigned with the tool `gdmp_filter_catalogue`.

6. Performance considerations and results

There are several ways to evaluate a file replication system and we just mention a few of them: does it satisfy the user requirements? Is it secure enough and can one trust users at distributed sites in the Data Grid? Is the performance okay? In the development process of GDMP we first put emphasis on functionality. Since Grid tools are still rather new, this question is reasonable and we showed that Grid tools can be used for file replication and at the same time satisfy basic user needs. Once the file transfer works correctly, one can start to optimise the actual transfer of data over the network.

GDMP is designed to replicate a set of files to a remote site. In principle, there are two basic methods to optimise file transfer for several files in a multi-user environment with data traffic over the Internet:

- *Optimise the transfer of a single file and get as much network bandwidth as possible:* This is achieved by using several parallel streams for a single file transfer and optimised TCP buffer sizes. We report on details of TCP buffer size tuning in [14]. It has the advantage that a single file gets transferred fast by using a large percentage of the possible throughput of a network link but has the disadvantage that other, concurrent network transfers get lower bandwidth rates.
- *Parallel client transfers:* This is the conventional and easier way to transfer a set of files efficiently. Our experience has shown that 5 to 10 parallel client transfers (using gdmp_replica_get) can also saturate the throughput of a network link. As our results indicate, this is true for wide-area connections but not really significant for local-area transfers.

[7] An un-tuned network connection is a data transfer link where default or non-optimal buffer sizes are used.

Here, we only report on performance results of the parallel clients and refer the to [1,14] for GridFTP performance results for parallel streams within a single client transfer.

GDMP 2.0 has been developed and tested on Linux RedHat 6.1 and 6.2 using the C++ compilers gcc-2.91.66 and gcc-2.95.2. We measured the latency for the whole replication process of a number of files on local as well as the wide-area networks. Note that the file replication process consists of several steps like CRC check sum calculation, registration of files in catalogues and notification of remote sites and thus has a small overhead compared to the raw transfer of the file. Our performance results here focus on the entire replication process.

For the LAN tests within CERN's computer centre we used two Linux PCs (RedHat 6.2, kernel 2.2.19-6.2.7), 800 MHz, 100 Mbps network). We transferred files with the size of 273 MB and compared the transfer speed for a single client and for two parallel clients. We obtain the following result. Note that all the tests are done with default configurations of the GridFTP servers and buffer sizes on all machines.

	single file transfer	2 parallel clients
transfer speed in s	[91–134]	[220–240]
transfer in MB/s	[2–3]	[2.2–2.4]
validation time in s	50	150
registr./notific. in s	1	1

During the tests the machine was in use by a few users who used parts of the CPU time temporarily. This is a valid assumption also for a Grid with several users. We did several tests and the transfer results are in the span given in brackets. We can derive that for fast network connections (over the LAN) there is hardly any performance gain for parallel clients since a single client utilises already much of the network link as well as of the disk I/O rate. The maximum theoretical throughput of the disk on both machines is about 4.4 MB/s. Furthermore, the GDMP client uses a rather high amount of CPU (between 10 to 40 percent) as compared to WAN tests (see below) with about 2 to 4 percent CPU time.

The WAN test was done with one of the above CERN machines (Geneva, Switzerland) and a Linux PC (Redhat 6.1, kernel 2.2.16-3, Pentium III, 863 MHz, 512 MB RAM, 100 Mbps Ethernet network card) in Fermilab, near Chicago, USA. The network link to Chicago has a theoretical maximum throughput of 45Mbps. We tested several single and parallel clients in order to get maximum throughput from the network link for a set

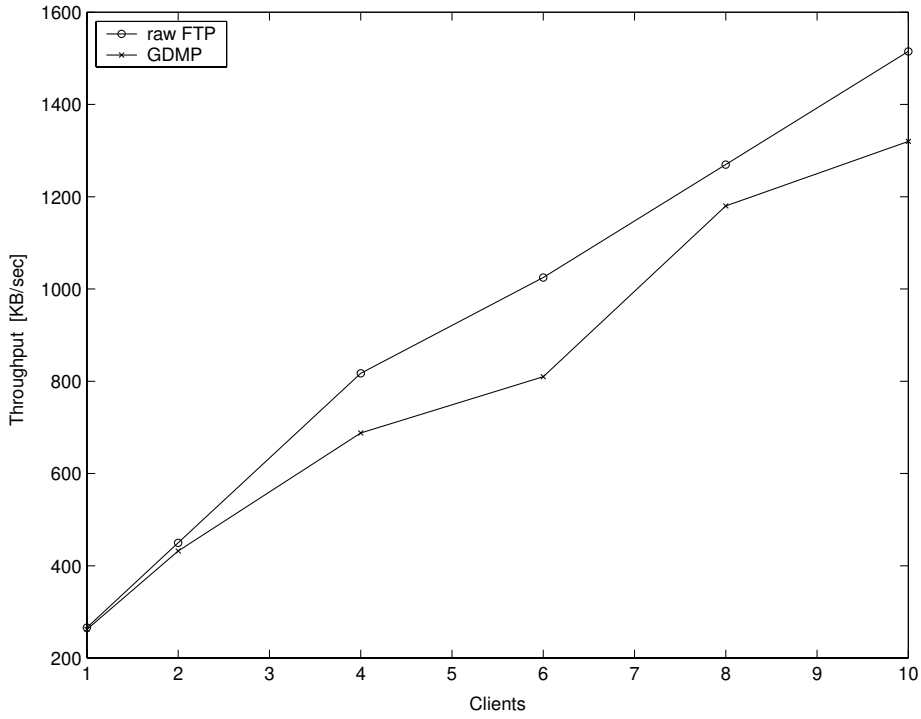

Fig. 4. Comparison of several parallel clients transferring 200 MB files over the transatlantic link Chicago-Geneva. Note that the 10 parallel clients transferred data in the morning (European time) while all other transfers were in the evening (European time). This shows again the fluctuation of network throughput. The two graphs correspond to the raw FTP transfer time (FTP) and the aggregated replication process (GDMP) including validation, registration, etc. A single GSI-FTP stream gave between 220 tp 260 KBs depending on the day time.

of files. Figure 4 shows the pure FTP rates as well as the aggregated GDMP transfer rates including validation, registration and notification of files. The graph shows that with increasing number of parallel clients the throughput of a single client decreases whereas the aggregated throughput for several parallel clients approaches to a maximum bandwidth utilisation with about 8 to 10 parallel clients. With a higher number of parallel clients, the gained throughput is only marginal.

Our results show with checksum caching and smart dummy attach methods (a common trick to attach files to an Objectivity federation), we get almost the same aggregate throughput with GDMP as with plain, single GSI-FTP transfer stream. Hence, GDMP provides much more automated and reliable replication with tolerable overhead.

To conclude, GDMP's data mover module and thus the overall transfer performance of GDMP client transfers depends on the underlying file transfer implementation, the network topology and the tuning of the network. Thus, performance tuning is not a real issue for GDMP and thus we claim that network performance needs to be provided by underling Grid tools (e.g. a file transfer service). GDMP's responsibility is to interface with a given service.

7. Conclusion and future work

GDMP is now part of the official DataGrid software system and will be used in first DataGrid testbed in the last quarter of 2001. Furthermore, GDMP has been in production use at several sites (CERN, Caltech (Pasadena, CA), Fermilab (Chicago, IL), Pisa, Moscow, San Diego, Wisconsin) in the CMS experiment to transfer almost a Terabyte of data in total.

Our experience has shown that new technology like Grid tools can be combined and used efficiently for data replication in a wide-area distributed environment. The work on GDMP is continuing and we will add more fault tolerance and failure recovery to our future releases.

Acknowledgement

Several people have contributed with useful discussions, comments and help to the success of GDMP. In

particular we want to thank our colleagues in the CMS experiment in Europe and in the US, our colleagues in the DataGrid project and in particular the Data Management work package and also our colleagues in the PPDG project as well as in the Globus team. Thank you to Andrea Domenici and James Amundson for their contribution to the software.

References

[1] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel and 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 17–20, 2001.

[2] B. Allcock, A. Chervenak, I. Foster, C. Kesselman, C. Salisbury and S. Tuecke, The Data Grid: Towards an Architecture for the Distribute Management and Analysis of Large Scientific Datasets, *Journal of Network and Computer Applications: Special Issue on Network-Based Storage Services* **23**(3) (July 2000), 187–200.

[3] European DataGrid Project: http://www.eu-datagrid.org.

[4] Data Management Work Package in EDG: http://grid-data-management.web.cern.ch/grid-data-management.

[5] The Globus Project(tm), http://www.globus.org.

[6] I. Foster, C. Kesselman, G. Tsudik and S. Tuecke, A Security Architecture for Computational Grids, *5th ACM conference on Computer and Communications Security*, San Francisco, California, November 2–5, 1998.

[7] I. Foster, A. Roy and V. Sander, A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation, *8th International Workshop on Quality of Service*, Pittsburgh, June 5–7, 2000.

[8] J. Gray, P. Helland, P. O'Neil and D. Shasha, The Dangers of Replication and a Solution, *ACM SIGMOD International Conference on Management of Data*, Montreal, Quebec, Canada, June 4–6, 1996.

[9] Particle Physics Data Grid project (PPDG): http://www.ppdg.net.

[10] M. Hafeez, A. Samar and H. Stockinger, A Data Grid Prototype for Distributed Data Production in CMS, *VII International Workshop on Advanced Computing and Analysis Techniques in Physics Research* (*ACAT2000*), Chicago, Illinois, October 2000.

[11] A. Hanushevsky, Obejectivity/DB Advanced Multi-threaded Server (AMS) www.slac.stanford.edu/∼abh/objy.html, April 2000.

[12] W. Hoschek, J. Jean-Martinez, P. Kunszt, B. Segal, H. Stockinger, K. Stockinger and B. Tierney, Data Management (WP2) Architecture Report – Design, Requirements and Valuation Criteria, *DataGrid-02-D2.2-0103-1_2*, http://grid-data-management.web.cern.ch/grid-data-management/docs/DataGrid-02-D2.2-0103-1_2.pdf, Geneva, Sept 19, 2001.

[13] A. Samar and H. Stockinger, Grid Data Management Pilot (GDMP): A Tool for Wide Area Replication, *IASTED International Conference on Applied Informatics* (*AI2001*), Innsbruck, Austria, February 19–22, 2001.

[14] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman and B. Tierney, File and Object Replication in Data Grids, *10th IEEE International Symposium on High Performance and Distributed Computing* (*HPDC-10*), San Francisco, California, August 7–9, 2001.

[15] H. Stockinger and A. Hanushevsky, HTTP Redirection for Replica Catalogue Lookups in Data Grids, to appear in *ACM Symposium on Applied Computing* (*SAC2002*), Madrid, Spain, March 10–14, 2001.

[16] H. Stockinger, A. Samar, S. Muzaffar, F. Donno and A. Domenici, Grid Data Mirroring Package (GDMP): User Guide for GDMP 2.0, http://cmsdoc.cern.ch/cms/grid, October 2001.

[17] T. Wildish, Accessing Objectivity catalogues via the web, http://wildish.home.cern.ch/wildish/Objectivity/scripts.html.