Enabling Access to Mass Storage

J Gordon, J Jensen, O Synge
e-Science department
CCLRC–Rutherford Appleton Laboratory

11 August 2003

Abstract

A number of different mass storage systems are in use in the scientific communities. The Storage Element (SE) is a Grid interface to storage systems being developed as a part of the European Union DataGrid project [EDG]. It enables Grid clients to access data in the storage systems in a uniform way, i.e. without knowing which storage system they are talking to. The SE provides a flexible interface to various storage systems, and can provide additional functionality not provided by the storage itself, such as access control, or automatic replication across sites. We describe the current status of the SE, how experiments are using or will be using the SE, and future developments. We describe in detail some aspects of single sign-on authentication and delegation, and an overview of message queueing systems for the future development of the SE. This work will be of interest to anyone with large scale storage requirements, e.g. the UK Grid community, but this work will also be of interest to individual Grid developers.

1 Introduction

Storage requirements in the scientific community are serviced by disparate storage systems, ranging from the simple disk system to large systems with tapes and robots. As scientific computation moves toward a Grid environment, a need arises for accessing those storage systems through Grid protocols. Some files in the storage systems will then be made available to users on the Grid, others will not. Conversely, Grid users will be able to write files into the storage systems, and the files may later be replicated to other storage systems. Of course, access to the storage must be granted to authorised users only, and access must be auditable by storage administrators.

To address these problems, the European Union DataGrid project [EDG] has built the Storage Element (SE), a robust and extensible interface to mass storage systems. This work is being done by the GridPP and e-Science groups at RAL.

2 Overview

We have designed and implemented an SE software system. The role of the SE is primarily to sit between the client and the Mass Storage System (MSS) (for the purpose of this paper, a *disk* is also a MSS); to hide the MSS differences from the client and to allow access to the MSS using protocols that it does not naturally support. In addition to this role, the SE will also provide other Grid functions as applied to data access. For example: security; access control; quotas; monitoring; logging; network transfer estimation. To the outside world, the SE provides three types of functions:

- For data transfer, it supports existing protocols such as RFIO, and GridFTP, and will be extensible to new protocols that may appear. It will allow these protocols to access the MSS. There will also be an API to access files in an SE and replicate files between SEs.
- For control, it provides different namespaces for different VOs, and a range of functions such as staging files into and out of tertiary storage, and will in the future provide additional control functionality such as space reservation, fine grained (file level) access control, and pinning. An API to these functions has been defined and implemented. In the near future the SE will also support the SRM protocol (see below).
- For information, it acts as information providers to the DataGrid Information Service, providing metadata about the SE, the underlying MSS, and the files therein.

The design of the SE follows a layered model with a central core handling all paths between Grid client and the MSS. This approach was chosen as the most maintainable and flexible provided all or most functionality is implemented in the core.

3 Experiments using SE

3.1 Particle Physics

PP experiments today routinely handle thousands of files totalling tens of Terabytes of data. They distribute this around the world for use by large international collaborations and to make use of the CPU and storage resources that belong to member institutes. This is just about feasible but is achieved by many different ad hoc methods; very little of the software is shared and it requires a lot of manual intervention.

The next generation of experiments [LHC] will have even bigger collaborations and will generate Petabytes (10**15 bytes) of data each year in millions of files. Solutions for managing these data are based on Grid technology and include distribution of data as replicas, moving data, mirroring and then the subsequent discovery of the relevant data for analysis [RLS]. The flow can be characterised as a movement away from the data source at the accelerator laboratory to remote institutes.

Storage of the physical files in mass storage systems is at the lowest level of this data hierarchy but movement of data around the world requires direct access from the Grid to the data.

3.2 Earth Observation

EO raw data are observations of land, ocean, and land from satellite mounted instruments. EO data products are derived from this raw data through a number of processing steps. Since the satellites and instruments are run by many organisations, the data reside in many data archives around the world. In contrast to the PP data problem, the EO scientist wishes to bring together data from a variety of data products at distributed centres. Here the format and metadata are more important. Here too, the Grid is seen as providing a solution to different access methods at different data archives. For mass storage the main requirement is the same; to have remote, authenticated access using common protocols to data stored in different mass storage systems. In addition the EO data can be held as very large numbers of small files which tend to be stored inefficiently in large data systems. Solutions are under investigation which allow the bundling of many small files into larger containers which are handled more efficiently.

4 SRM

The Storage Resource Manager (SRM) is a control interface specification. There are two versions; version 1 is an older version that emerged as a collaborative effort between Lawrence Berkeley, Fermilab and Jefferson Lab. Version 2 is a more recent effort which is a collaboration between the SRM1 collaborators, Rutherford Appleton Lab, and the CASTOR and DataGrid Data Management groups at CERN. Both versions are today defined as web services (WSDL).

Like the SE, SRMs do not store data themselves but act as interfaces to MSSs. If clients wish to read a file, they will first have to send a `get` request to the SRM. The SRM will return a request id and status structure, and the client keeps querying the request until it is ready (it may take time to process a request: for example it can take hours to stage in a file from a busy tape storage system). When the request is ready, the SRM provides a Transfer URL where the client can fetch the file. When the client has finished doing transfers on the file, it must inform the SRM that it has finished.

It is worth observing that

(a) The client polls (rather than provide a callback mechanism): tests have shown that this does not in general adversely affect the system load on the server, and it is more secure and firewall friendly (the client does not have to listen on any ports for connections from the SRM).

(b) Requests are asynchronous.

(c) The SRM does not do any data transfer. In fact there is a PUSH mode where the SRM pushes the file to the client but it was never generally supported.

Creating and uploading a file to an SRM is exactly analogous to downloading: the client issues a `put` command to the SRM and the SRM eventually provides a URL. When the client has finished uploading the file to the URL the client must inform the SRM that it has finished.

The SE currently provides an SRM type interface, i.e. generally clients access files using `get` and `put` type commands. The exact arguments are not quite the same because the SRM WSDL specification was not available when the SE interface was created, and the SE also currently does not process requests asynchronously. See the section on Message Queues for further details.

At the time of writing, the SRM version 2 specification is nearly complete. The full semantics of the space and file management is beyond the scope of this paper, but an SE can be easily modified to provide partial support for SRM2 once version 1 is deployed.

5 Security

As always on the Grid, users are identified using their personal certificates, and hosts and services are authenticated similarly using their certificates. Sending commands to an SE thus satisfies the following security properties.

- Authentication: the client is clearly identified to the SE and the SE can use this as a basis for authentication.
- Integrity: no one can alter the command in transit.
- Confidentiality: other than the fact that a command was issued to the SE, an eavesdropper cannot gain any information about the command. For example, an eavesdropper may guess that the client is issuing an SRM `get` request but cannot know which file the client is requesting.
- Non-repudiation: The client cannot later deny that it sent this request to the SE because the authenticated request is logged by the SE.

Data transfers commands using the GridFTP protocol satisfy the same security properties, and in addition, the data transfer itself can be encrypted.

5.1 Single sign-on and delegation

One of the central end user requirements is to have single sign-on and delegation. In a grid context, both of these goals are achieved using Globus proxies [TFE+03] (a *proxy* in this context is a short-lived

certificate whose corresponding private key is unencrypted). Delegation is simply achieved by copying the proxy to the server. Proxies have the disadvantage that normal (non-Grid) SSL servers and clients do not accept them as valid because they are signed by the user's keys and users are not allowed to sign keys for security reasons. Another disadvantage of the proxy is that when it is used for delegation, the client has no further control over how the proxy is used.

There are other ways to achieve the same goals. Consider the way the SSL/TLS toolkits work in general [Res01], and OpenSSL in particular. When a client sets up a SSL connection, first it creates a *context* which contains the user's private key and other things that are the same for every connection made by the client. We can achieve single sign-on for SSL by recognising that the contents of the context are not just the same for every connection the client makes, but also for every *client*, and create an "agent" that remembers the context, similar to what ssh-agent does for ssh. This does mean that each SSL client will have to be modified to check for the existence of the agent before it makes an SSL connection. The greatest advantage of this scheme is that it works also with SSL servers that are not Grid-aware. Another advantage is that it works with the unmodified OpenSSL libraries, clients, and servers, which means that security fixes to those can be deployed immediately. The disadvantage, compared to the proxy solution above, is that delegating a context is not acceptable: the context contains the user's private key in unencrypted form, so it must not ever be accessible by anyone but the user.

In any case, delegation should be done "properly". Ideally, there should be a challenge from the remote server which is passed to the client, and the challenge should be signed by the remote server. Only an entity in possession of the client's private key should be able to answer the challenge. Furthermore, on the Grid the concept of single sign-on means that the user may have a non-interactive client running on the user interface machine which must then be able to answer challenges on behalf of the user. Of course the client must answer only authenticated challenges from approved remote servers. This problem is also easily solved using the context agent.

5.2 Capabilities and Groups

This area of our work will involve a series of small discrete improvements. From the users perspective it is becoming evident that the original model proposed for the Grid of distinct Virtual Organisations each with no knowledge or ability to interact with each other is insufficient for our current grid environment. What is needed by the users is the ability to choose if the files are shared between the virtual organisations and choose at runtime which VO they wish to work within.

To facilitate this requirement recent DataGrid work [vom] has gone into providing proxies which provide support with Capabilities, a mechanism of specifying a role for a certificates interaction, and groups which provide the ability to manage collections of users and their access.

6 Future plans and developments

6.1 Message Queue

The storage element is based on a message queue architecture. Message Queues are commonly used in financial, communications, and process control sectors where the outcome of failure scenarios maybe serious or costly [GR93]. Any improvement in the message queue performance and flexibility improves the storage element.

The current system is very robust and predictable. Unfortunately our message queue is not fast. We have a clean level of separation between the Storage element and the underlying message queue. Some effort is currently going on to improve this interface, particular focus is on providing a C API, and improving the performance.

Our current system may become a message queue project in its own right but we may replace it with a commercial or established solution. The current message queue will be expanded to support all the required functionality before the next improvement is decided. We shall provide out of band transaction queries, so one transaction will be able to query the status of a second transaction. This will give the storage element more flexible error handling with a potential improvement in debugging. Transaction should be able to start a series of nested transactions. This allows for reuse of transactions in the same way as a subroutine, while preserving the traceability and auditing of a single transaction.

Message queue infrastructures manage the details of concurrency, asynchronous behaviour and auditing. Concurrency is typically managed by queueing messages between each module and a resource manager ensuring that all modules in a transaction run in isolation. Asynchronous behaviour is inherent in the architecture of a message queue.

The new abstraction interface that for the storage elements message queue API is is similar to the Data Distributor interface [ddm], but extends the simple UNIX file model to add transaction forking, querying and error handling.

This API may be reused in any scenario where reliable asynchronous distributed secure infrastructure is needed. The message queue API is currently being finalised and as an ANSI C interface with a pluggable back-end queue implementation, initially only using our own message queue infrastructure.

There are many Java message queues available including the Open Source Message Queue [jmq], or the Java Message Service (JMS) [jms].

But since the majority of the SE code is in C/C++, particularly the drivers to the mass storage systems, and we expect performance to be a priority the majority of the effort has been spent looking at C/C++ solutions.

The basic POSIX message queue [pos] is not persistent, and therefore not suitable for transaction management.

The Berkeley DB [ber], does provide persistent queues but not a messaging framework.

MQS [mqs] is relatively new and as yet untested and provides little more than the Berkeley DB and event notification.

Secure spread [spr] and its parent project spread are probably the most mature messaging infrastructures available in the C/C++ world and will soon be evaluated. This project uses symmetric cryptography for security and so may not suitable for a grid environment.

The Data Distributor [ddm] package is a general purpose buffer manager. Although the current version remains unchanged since December 96, it does appear to be in production on several physics experiments.

None of these open source projects appear to match the hot swap functionality and enterprise levels of reliability expected in projects such as IBM MQ products [ibm] or Falcon Q [fal] but open source message queue components are relatively common.

BSD kqueue which is a kernel API extension available in BSD 5 and above, that provides a single event on a single host when files, sockets or pipes are modified.

The libevent API [lib] provides a clean way of wrapping kqueue like functionality into a cross platform application, We have tested this software on the Linux platform.

Message Queue Security

We intend to integrate validation of incoming messages: the XML will be validated with XML Schemas, and the values with regular expressions or whatever else is appropriate. This will allow us to validate XML messages entering the SE's message queue from other SEs, or other clients.

6.2 A distributed SE

Thanks to the use of a message queue architecture and our previous work on delegation, very little new work is required, apart from packaging, to distribute the SE across multiple SE servers. The current focus is to remove the single points of failure from the system. Our long term goal is to use a distributed database where no single node contains the only copy of the filesystem.

7 Conclusion

We have described a Grid interface to storage systems that enables anyone with a Public Key Infrastructure (PKI) to securely share files between existing mass storage systems using common Grid tools. We gave an overview of the system design emphasising the flexibility, and described in some detail single sign-on authentication now and some possible future developments. We described Open Source message queueing systems and how they apply to the SE.

References

[ber] Berkeley db. http://www.sleepycat.com/.

[ddm] Data distributor. http://www.phenix.bnl.gov/phenix/WWW/online/oncs/dd/dd.html.

[EDG] The European Union DataGrid Project. http://www.eu-datagrid.org/.

[fal] Falcon q. http://www.level8.com/.

[GR93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[ibm] Ibm message queue. http://www-3.ibm.com/software/integration/mqfamily/.

[jmq] Java message queues. http://www.osmq.org/.

[jms] Java message service. http://wwws.sun.com/software/products/message_queue/home_message_queue.html.

[LHC] Large Hadron Collider. http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/.

[lib] libevent. http://www.monkey.org/~provos/libevent/.

[mqs] Mqs. http://www.foo.be/mqs/.

[pos] Posix message queues. http://www.tldp.org/LDP/lpg/index.html.

[Res01] Eric Rescorla. *SSL and TLS*. Addison Wesley, 2001.

[RLS] EDG Replica Location Service. http://edg-wp2.web.cern.ch/edg-wp2/replication/index.html.

[spr] Secure spread. http://www.cnds.jhu.edu/research/group/secure_spread/.

[TFE+03] Steve Tuecke, Ian Foster, Douglas Engert, Von Welch, Mary Thompson, Laura Pearlman, and Carl Kesselman. Internet X.509 Public Key Infrastructure Proxy Certificate Profile. http://www.globus.org/security/standards/draft-ietf-pkix-proxy-06.txt, May 2003.

[vom] Virtual organisation membership service. http://hep-project-grid-scg.web.cern.ch/hep-project-grid-scg/voms.html.