Data Management in EGEE[*]

P. Kunszt, P. Badino, R. Brito da Rocha, Á. Frohner,
G. McCance, K. Nienartowicz, CERN, Geneva, Switzerland

Abstract

Data management is one of the cornerstones in the distributed production computing environment that the EGEE project aims to provide for a European e-Science infrastructure. We have designed a set of services based on previous experience in other Grid projects, trying to address the requirements of our user communities. In this paper we summarize the most fundamental requirements and constraints that have driven the architecture and the particular choice for service decomposition in our service-oriented architecture. The three service groups for data management services are the Storage Element, the Data Scheduling and the Catalog Services.

INTRODUCTION

In this document we present the architecture of the Data Management components of the EGEE Grid Middleware (called `gLite`) [5]. It is driven by the requirements of Grid applications [1, 2, 6], the Open Grid Services Architecture (OGSA) [11] as well as previous experience from other Grid projects. For an overview of gLite, the reader is referred to [4].

All of the `gLite` data management services are based on the concept of service oriented architectures [9]. We aim to deliver a modular set of services that call upon each other to deliver their specific functionality and semantics to the end-user. There are many services necessary to meet the needs of the end-user. All the services have to work together in an integrated manner, delivering end-to-end capabilities to the applications working with them.

The modularity aspect and service orientation also allows organizations to replace services with their own, adding specialized semantics they need. So we address actually not only the need of the end-user to have a system that 'just works' but also the need of the service provider to be able to slot in their own custom service, providing the necessary interface to it and making it work in an integrated fashion with `gLite`.

The interfaces are all described through WSDL [3], and we also make an effort to adhere to WS-I [7] in order to have the highest possible interoperability. We do not adopt at this point in time any of the emerging web service technologies (like WS-Addressing, WS-Notification, WSRF, etc) since most of them have not gone yet through the standardization process and the tooling that supports them is not mature enough for our purposes.

OVERVIEW

In this section we give an overview on the data management services present in `gLite` and motivate their architecture based on the requirements.

Requirements

The requirements for the Data Management Services may be classified into functional and non-functional requirements. Non-functional requirements and constraints include generic considerations about scalability, security, service policies, reliability and resilience. Functional requirements have to do with the service logic, semantics and interactions with the user and among themselves. The most important non-functional requirements are: Not having single points of failure in the system, providing locality of reference, i.e. important data for job execution should be available locally, and security considerations dealing with the enforcement of site policies, VO policies and accounting information.

Additional motivation for the service architecture and their interactions is based on the basic functional requirements for data management. These are simple operations as seen by the user, however, in a distributed environment they require complex well-coordinated operations on many services in order to fulfil the non-functional requirements as well. The operations are only complex inside the service architecture but the user is given the illusion of simple atomic operations. Due to the distributed nature of these operations the user will see different options for quality of service and many more failure modes.

In terms of functionality, we address first and foremost the management of files (as opposed to databases), since most of our requirements come from communities (espe-

[*] EGEE is a project funded by the European Union under contract number INFSO-RI-508833

cially High Energy Physics) whose data is stored mostly in files. Of course we also provide interfaces to enable access to data in databases and metadata catalogs. The end-user functional requirements [1, 2, 6] can be summarized as a set of operations that the data management subsystem has to support. These operations deal with

- **File I/O.** The user wants to access their data transparently through standard I/O mechanisms (POSIX), using a file name that the system understands in every environment, wherever the actual job is being scheduled. All the security mechanisms, name resolution, etc, should happen without the explicit intervention of the user. However, if there are problems, the user expects to have a descriptive error response.

- **Transfer files across the WAN.** The user wants to distribute data across many sites in the Grid. She wants to use a simple intuitive interface to instruct the Grid services to make files and sets of files available at specific locations, also based on subscription models.

- **Browse the catalogs.** The user wants to browse the file namespace similar to a virtual file system. Metadata should be attached to the catalogs and selection of files and sets of files should be possible based on metadata. File ACLs should be available and enforced by the system.

There are additional implicit requirements on the Grid, stemming from requirements on other Grid services, especially the Workload Management Services. For example, in order to schedule jobs at the correct Computing Element, the Grid Storage Element needs to be able to deal with space reservations. Such requirements are not listed in detail at this point.

Another important requirement is the security aspect of working in a Grid environment. All services need to authenticate and authorize the user based on his credentials. The access to data needs to be restricted through well-defined and well-manageable access control mechanisms. The concept of security is pervasive in the sense that the architecture and design of the services is strongly driven by security considerations. It is not possible to simply 'add security' to a complex set of services later on without massive architectural changes.

Files in the Grid

We have the following file names in the Grid:

LFN Logical File Name: A logical (human readable) identifier for a file. LFNs are unique but mutable, i.e. they can be changed by the user. The namespace of the LFNs is a global hierarchical namespace.

GUID Global Unique Identifier: A logical identifier, which guarantees its uniqueness by construction (based on the UUID mechanism [**?**]). Each LFN also has a GUID (1:1 relationship). GUIDs are immutable, i.e. they cannot be changed by the user.

SURL The Site URL specifies a physical instance of a file. Also referred to as the Physical File Name (PFN). SURLs are kept in the Storage Elements.

Usually, users are not directly exposed to GUIDs and SURLs, but only to the logical namespace defined by LFNs - the Grid system handles the rest (see also Figure 2).

Service Architecture Overview

The three main service groups that relate to data and file access are: Storage Element, Catalog Services and Data Scheduling.

The Storage Element (SE) is responsible for saving/retrieving files to/from the local storage that can be anything from a disk to a mass storage system. It manages disk space for files and maintains the cache for temporary files.

The LFN - GUID - SURL mappings are being kept in the Catalog Services.

The Data Scheduling services manage wide area data transfers between SEs and coordinate the entries in the catalogs with the actual data in the SEs.

STORAGE ELEMENT

The SE provides the following set of capabilities:

Storage resource. The SE provides the storage space to store files.

POSIX-like I/O access to files. The SE provides an interface that can be used to access the files directly through a POSIX-like protocol.

File Transfer requests. Each SE has a capability to queue and manage file transfers between the local SE and remote SEs.

Storage space management. The SE may provide a means to manage the available space and the lifetime of files. The SE may also provide additional optional mechanisms such as quotas, space reservation, namespace management, etc.

Access Control The SE secures file access through access control lists.

Accounting The SE may keep logs about the resource consumption of each user and organisation.

We distinguish logically between opportunistic storage (also referred to as a *tactical SE*) and permanent storage (*strategic SE*). The tactical SE makes very little guarantees towards the longevity of a file and can be regarded as a 'scratch work space'. The strategic SE gives the user some guarantee about their data files. Such strategic SEs usually are coupled to a mass storage device to store the data permanently.

SEs are controlled by the sites and are subject to local policy. Having the concept of the tactical SE allows a site to declare space in a local store in an opportunistic manner. It can provide storage that is currently unused by their local users and revoke it whenever necessary. This will make it attractive to sites to make resources available to grid users, knowing they can re-claim the resources whenever they want. Thus, jobs requiring local storage may be run at many more sites. Users are expected to keep only disposable data in such stores - meaning that it should not matter if the instance of the data is lost - because it can be re-generated or re-copied from a master instance for example. Important data, master copies should not be kept in such storage (only at the user's own risk). If users generate new data at such stores, they should either register a master copy at a more long-term SE or be prepared to re-generate the data if necessary.

The SE has several public interfaces: The Storage Resource Management (SRM) interface [10], a POSIX-like File I/O interface and some file tansfer protocol interface. The SE also runs internally a File Transfer Service, which is however an internal component, dealing with requests that come through the File Placement Service (described below).

The SE is tightly coupled to a local file catalog (described below) in order to resolve the logical name space used by the user to the actual datafile to be found on the given resource.

The POSIX

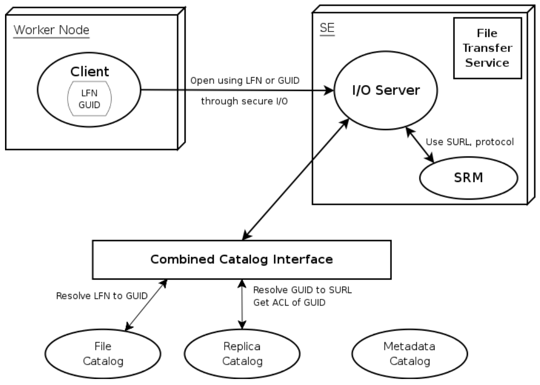

Figure 1: The SE file I/O interactions.

The actual POSIX-like File I/O interface is not a web service; the client communicates with the server through a traditional socket-based I/O library. Figure 1 shows how the I/O works in detail. The I/O client library accepts either LFN or GUID as an input to the API. The LFN or GUID is presented to the I/O server, which checks with the Catalog Service whether the user is allowed to access the file in the given way, resolving the GUID or LFN into the SURL which is then handed to the local SRM in the process. The actual file handle is acquired through SRM and will be used to access the actual file and serve the data to the client.

There is a lot of room for customisation in this model since all components are pluggable. For example, the gLite I/O is currently based on the Alien I/O [8], but it should be possible to plug in other libraries instead.

CATALOG SERVICES

We identify four interfaces that may be implemented together or separately in the `gLite Grid`. The File Catalog (FC) and Replica Catalog (RC) both make part of the GUID – LFN – SURL mappings accessible (see Figure 2). The File Catalog interface exposes operations only on the mappings that it manages, as does the Replica Catalog interface. The Replica Catalog interface gives access to the GUID – SURL mappings identifying all replicas of a given GUID. All the rest of the mappings are kept in the FC. Operations that involve both catalogs and also the Metadata Catalog and File Access Service are exposed through a Combined Catalog interface. The reason for

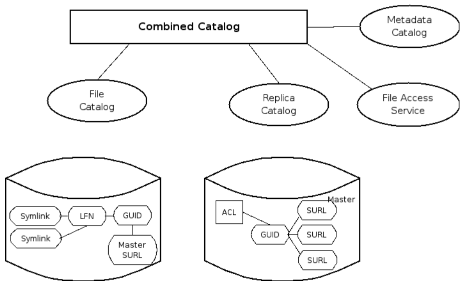

Figure 2: Catalog Services with the accessible mappings.

this interface decomposition is to have service interfaces with well-defined semantics which may be implemented by many parties. In this model, a possible scenario is that all three interfaces may be implemented over a single central database. Another possibility is that the File Catalog is central whereas the Replica Catalog is distributed over many sites and synchronised through some database replication scheme. Also, either the File Catalog, Replica Catalog or most commonly the Metadata Catalog might be implemented and controlled by the VO directly. If the implementation provides the proper interfaces, it can participate in the Data Management System and need not provide all the other functionality as well. Actual deployment also depends on the needs of the particular VO. It is possible to share services across VOs, but we expect that each VO has its own set of catalog services.

DATA MANAGEMENT

The Data Management subsystem has the following components (see Figure 3 in [4]):

Data Scheduler From the VO's point of view the Data Scheduler is a single central service. It may actually be distributed and there may be several of them, but that depends on the implementation. It is responsible for scheduling and keeping track of the data transfers and catalog operations between multiple sites.

Transfer Fetcher The Fetcher polls the available DS for a given VO and checks for transfers where the target is an SE at a local site. It takes the transfers from the DS and submits them into the FPS. (Pulling requests from DS, pushing into FPS). So it acts as the connecting service between the global DS and the local FPS.

File Placement Service FPS There is an FPS running at each site making one logical instance per VO. It coordinates the file transfers and the catalog operations, exposing atomic file replication operations to it's clients.

File Transfer Library FTL The FTL provides the API interface to the data management operations available to Grid clients. It manages the client's communication with the FPS.

The Data Scheduler is a top-level service, keeping track of data movement requests in a VO that are being submitted directly by the user through a portal or user interface or by computational jobs submitted to the Workload Management Service. The Transfer Fetcher polls the Data Scheduler and fetches transfers whose destination is the local site for the given VO, inserting new requests into the File Placement Service. The File Placement Service coordinates the transfer performed by the File Transfer Service and makes sure that the File and Replica catalogs are updated properly.

This service breakdown has been chosen to keep the services modular and dedicated to a well-defined task. Each SE has a file transfer queue and an associated request fetcher that polls each local FPS for jobs that target the given SE. This is done behind the scenes, the user controls the actual transfer solely by interaction through the FPS. The SE transfer is the lowest-level service used here. It deals only with the transfer of files between Storage Elements. The destination SE is explicitly specified in the transfer request. Each SE runs only one such queue.

The File Placement Service FPS has the task to coordinate the transfer with proper registration in the Catalogs. It monitors the progress of a transfer in the local SE queue and will make sure that the catalogs are properly updated, re-trying the operation according to well-defined VO policies if necessary. Each VO has an FPS per site. The user submits requests for transfer (implicitly including the proper catalog operations) through the File Transfer Library API to the local FPS. If the transfer request does not have a local SE as it's destination, the FPS forwards the request into the VO's global Data Scheduler. So it keeps it's task well-defined and simple.

The Transfer Fetcher is a component running at each site, periodically polling the DS whether there are any pending transfer requests that have the local site as it's destination. If yes, these requests are retrieved from the DS and forwarded to the local FPS. Once the FPS completes the request, the fetcher will update the DS to signal the completion of the request.

The DS can also be contacted directly by the user through the user interface. It keeps a persistent queue of all requests. Other processes such as policy managers and optimisers may operate on the same queue to refine the requests.

This service breakdown, which is modelled after well-understood mechanisms already in use by schedulers, ensures that the services are kept simple enough to maintain a high reliability of the overall system.

SUMMARY

We have given a brief overview of the components that make up the data management services in EGEE and described the interface breakdown and some of the internal interactions.

REFERENCES

[1] F. Carminati, P. Cerello, C. Grandi, E. Van Herwijnen, O. Smirnova, and J. Templon. Common Use Cases for a HEP Common Application Layer – HEPCAL. Technical report, LHC Computing Grid Project, 2002. `http://project-lcg-gag.web.cern.ch/project-lcg-gag/LCG_GAG_Docs/HEPCAL-%prime.pdf`.

[2] F. Carminati and J. Templon (Editors). Common Use Cases for a HEP Common Application Layer for Analysis – HEPCAL II. `http://lcg.web.cern.ch/LCG/SC2/GAG/HEPCAL-II.doc`.

[3] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1*. W3C, March 2001. `http://www.w3.org/TR/wsdl`.

[4] et.al E. Laure. Middleware for the Next Generation Grid Infrastructure. In *2004 Conference for Computing in High-Energy and Nuclear Physics (CHEP 04)*, Interlaken, Switzerland, 2004.

[5] EGEE. EGEE Middleware Architecture. EU Deliverable DJRA1.1, July 2004.

[6] EGEE Application Working Group. Biomedical Application Requirements. `https://edms.cern.ch/file/474424`.

[7] The Web Services Interoperability Organization. WS-I Documents. `http://www.ws-i.org/Documents.aspx`.

[8] Andreas-J. Peters, P. Saiz, and P. Buncic. Alienfs - a linux file system for the alien grid services. *ECONF*, C0303241:THAT005, 2003.

[9] David Sprott and Lawrence Wilkes. Understanding Service-Oriented Architecture. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmaj/%html/aj1soa.asp`.

[10] SRM 1.1. `http://sdm.lbl.gov/srm.wsdl`.

[11] GGF OGSA WG. The Open grid Services Architecture, Version 1.0 (draft 016). `https://forge.gridforum.org/projects/ogsa-wg`.