DON QUIJOTE - DATA MANAGEMENT FOR THE ATLAS AUTOMATIC PRODUCTION SYSTEM

Miguel Branco[#], CERN, Geneva, Switzerland

Abstract

As part of the ATLAS Data Challenges 2 (DC2), an automatic production system was introduced along with a new data management component.

Previously data management tools used for the Data Challenges were built as separate components from the Grid middleware. These tools relied on a database of its own which acted as a replica catalog.

With the extensive use of Grid for the entire DC2 production it is no longer possible to have a data management tool that is designed to be independent of the Grid middleware. Each Grid relies on a replica catalog of its own along with Grid-specific data management software.

ATLAS Data Challenges 2 has used uniformly the resources provided by three Grids: NorduGrid, US Grid3 and LCG-2.

The solution we present was to build a data management proxy service – Don Quijote - which consists of a common high-level interface, whose implementation depends on each Grid's replica and metadata catalog as well as the storage backend.

Don Quijote provides management of replicas in a services oriented architecture, across the several "flavours" of Grid middleware used by ATLAS.

With a higher-level interface common across several Grids a user such as the new automatic production system can seamlessly manage replicas independently of their hosting environment. Given the services-based architecture a lightweight set of command line tools is available for users to interact uniformly with ATLAS file-based data.

Users can therefore uniformly manage files within or across Grid flavours, e.g: moving files from LCG-2 to US Grid 3 while maintaining attributes such as the Global Unique Identifiers.

INTRODUCTION

Don Quijote [1], or DQ, is the data management service for the ATLAS Production System.

Using Don Quijote users can locate replicas of files, add, remove or modify both logical and physical entries in grid replica catalogs. In addition a transfer service is provided to replicate files between storage locations as well as query each grid information system regarding data storage facilities. The client tools integrate replication of files with synchronization of replica catalogs, providing a single command to move data within or across Data Grids.

Don Quijote is built using a Service Oriented Architecture providing an uniform interface to interact with Data Grids.

DQ is modular by design; by plugging different implementations of all required modules a DQ server can be built against the grid middleware. There are currently three[*] deployed DQ flavours: DQ-LCG for the LCG-2 Data Grid, DQ-NG for NorduGrid ARC and DQ-Grid3 for US Grid3 VDT.

USE CASES

A typical usage of DQ from the ATLAS Production System [2] is as follows[†]: Windmill (the software that acts as the automatic production supervisor) queries DQ to discover where replicas of the input data files are located. The job is then steered to the grid with most input data available. The job executor on each grid [3] will trigger a DQ replication to get the input data to a location within that grid if the data is not available there. The job is executed and the resulting output data is registered in the native grid replica catalog. Later Windmill will validate the actual presence of the files, both in the grid catalog and on disk if necessary, by making a call to the DQ server. After this last validation step has been done a rename request is sent to DQ to give the output data its final name.

A typical usage of DQ for a physicist is as follows: a user wishes to perform analysis on part of a dataset produced by the ATLAS Data Challenges. Using DQ client, the physicist searches for a file named `"dc2.003011.simul.A6_dijet600._000*"`. The result shows that most of the partitions are in NorduGrid storage (actually there is no distinction on which grid the storage element belongs to, since it makes no difference - this information is only provided on verbose mode). The user then triggers the replication of some of those files to a storage element on his site. Later after the replication is finished, he decides to get a local copy of these files to his local working directory using the DQ end-user client tool. Finally after doing analysis he produces a new file which he wants to enter into a grid storage element for later use.

[#] miguel.branco@cern.ch

[*] A fourth one is under development to interface with EGEE gLite.

[†] In reality, the current Data Challenges doesn't use this fully integrated approach as the production is pre-split between grids – there is no need to steer jobs to different grids. It is expected that the usage of DQ by the production system will soon match what is described.

Using DQ end-user tools, he puts the file into a grid storage element. Since the generated file was a POOL file with the corresponding PoolFileCatalog.xml, the new logical entry is registered into the grid native catalog maintaining all attributes notably the POOL Globally Unique Identifier (GUID).

DESIGN AND IMPLEMENTATION

Data and Storage Organization

In the ATLAS Data Challenges data is distributed among three different Grids as it is produced. These grids are LCG-2, NorduGrid and US Grid3. Within each grid, data is distributed across many different sites.

In DQ the concept of a Storage Element is used. A Storage Element is a location where data is stored and is referenced by hostname[‡].

A DQ server must be setup per grid flavour to communicate with the underlying grid middleware.

Typically ATLAS has three different DQ servers running, matching one DQ server per grid. For the end-user the number of DQ servers or grids that he is interacting with is transparent. Only by convenience or usually performance is this known and the queries are steered to a single DQ server, matching a single grid. The Storage Element "host" is the distinguishing element for DQ. Don Quijote seamlessly sees the storage elements of all Grids by querying the underlying grid information system to discover those storage elements.

The ATLAS Production uses the concept of Logical Path Names (LPNs) to organize file-based event data. A logical path name is an extension of the logical file name (LFN). In addition to the LFN, a logical collection name - a hierarchical structure similar to the path of a typical *nix file system - is used. Together, the logical collection name and the logical file name form a fully qualified ATLAS production file which consists of a logical path name (LPN) : e.g. `/datafiles/dc2/simul/dc2.003011.simul.A6_dijet600/dc2.003011.simul.A6_dijet600._00043.pool.root`[§]. In addition, all files have associated a Globally Unique IDentifier (GUID). If this GUID was previously generated by a separate tool as is the case of POOL data, it is preserved.

The concept of logical collections has proven useful both to organize data and allow migration of existing catalog entries into future replica catalogs which may natively support collections. Currently, in existing replica catalogs the logical collection name is a meta-attribute attached to the LFN.

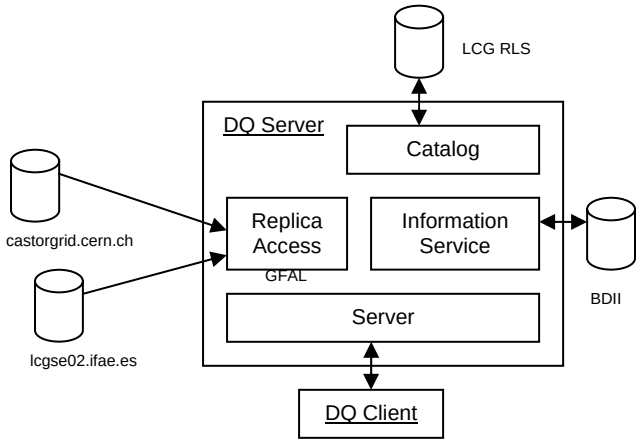
Early in the design process there was the intention to restrict the underlying file transport protocols to a small number. Therefore it was decided that transfers between storage elements either within a grid or across grids are done using GridFTP [7] as the underlying transport protocol. Hence DQ only supports GridFTP as the file *transfer* protocol.

End-user interface

DQ client is accessible using an API. Two distinct user interfaces have been built in Python using this API. One is `dms.py` which provides a command line interface to the full Don Quijote API. It is meant mostly for management of data by production managers.

A second user interface, especially designed for end-users is `dms2.py`. This encapsulates typical use-cases into a more user-friendly interface. In addition more verbose descriptions of the actions being taken as well as some level of end-user protection of data are included.

Architecture



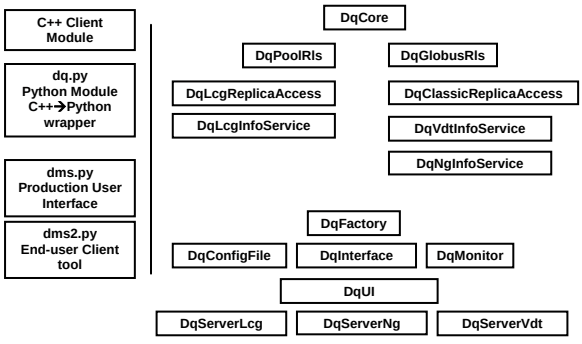
(1) Overview of DQ-LCG server and two LCG SEs

DQ provides a common interface to manage replicas. Client tools were built on top of the layer interface to help users deal with replica movement and registration within and across grid flavours.

DQ does not contain a replica database. Past experience proved that having a database which replicated the contents of grid replica catalogs was difficult to maintain. In a possibly chaotic environment such as the Grid these catalogs tended to become unsynchronized.

Therefore DQ acts as a proxy service, by forwarding the requests to the grid middleware it is bound to.

Server



(2) Overview of DQ modules

[‡] Similarly to the EDG/LCG Storage Element (SE) concept.

[§] In this example the LFN is just: dc2.003011.simul.A6_dijet600._00043.pool.root

A DQ server is made of four main modules: Catalog module, Replica Access module, Information Service module and the Server Request handling module.

The catalog module deals with all interactions with the native grid replica catalog. This includes searching logical files, renaming, adding, removing both logical and physical file entries, as well as built-in metadata attributes. This module has two distinct implementations sharing the same common interface: one implementation for Globus RLS 2.x [4] and another for the EDG RLS 2.x [5] using POOL 1.6.5 File Catalog [6] interface.

The replica access module provides support for data transfer. There are two implementations of this service: one for LCG and another one shared by both NorduGrid and US Grid3. The LCG implementation supports "classic Storage Elements" (plain GridFTP [7] servers) plus SRM [8] storage elements. It uses the LCG_util library and GFAL [9]. NorduGrid and Grid3 implementations support only plain GridFTP servers.

The information service module provides basic functionalities to query each grid's information system regarding storage elements. The interface includes calls to find grid storage elements, to verify if a host name is a valid storage element within a grid, as well as to resolve full storage URLs given a hostname. There are three different implementations of this interface since all supported grids use a different information schema.

The server module deals with the network server and the handling of client requests. This provides a common server interface with two distinct implementations: one using SOAP web services (implemented using gSOAP [10]) and another using plain sockets. Initially DQ was built as a server-side application using plain sockets and a custom-made communication protocol. Later the SOAP-based server was introduced and there is the intention to gradually move to supporting web services only. The server module is also responsible for handling a pool of worker threads meant to process client requests. This pool supports: priorities (short requests are processed first); queuing of requests; different queues depending on the expected time to process a request (e.g. a file transfer request which may have no pre-defined timeout is processed by a special queue handled by a subset of the worker threads). This allows for better response times and server availability when some users are querying the catalog, others inserting or modifying entries and others moving files around.

Security

The DQ servers can be ran both in secure or insecure mode.

On secure mode, the connections to DQ use GSI [11], which also requires a GSI-enabled DQ client. The user certificate is delegated by the server and all the requests to each grid information system, replica catalog or storage are done on behalf of the user.

On insecure mode, the connections to DQ are not protected. A limited subset of actions is available. All actions are performed by the server on behalf of the user using a service certificate running on the server side.

The insecure mode was deployed at a later phase, as it proved difficult for ATLAS to coincide all grids' VOs and policies, in time for the Data Challenges. This also allowed a more convenient migration of users to a grid environment for analysis.

The security model is under revision as it is an area for substantial improvement – depending also on the existence of standards for secure web services.

Client

The client is a shared library built in C++. SWIG/Python wrappers have been built along with a python module (`dq.py`) which implements a Python class (`DonQuijoteClient`). This allowed for quick development of scripts that use DQ from Python. This scripting ability proved to be very useful during the ATLAS Data Challenges where mass-registration, removal or replication of files was often necessary.

Using the Python/SWIG wrappers and the shared C++ library the two clients referred on "End-user interface" section have been built.

USAGE TO DATE

DQ has been used since the start of the ATLAS Data Challenges 2. Its main client is Windmill - the ATLAS automatic production supervisor. Windmill uses DQ mostly for validation of finished jobs - to rename the output data to its final name - as well as to find replicas and locate existing data. The phase of job definition is also partially plugged into DQ allowing jobs to be released for production only after their input data has been produced and verified that is accessible from DQ. In addition, DQ has been used outside of the automatic ATLAS production to steer data within and across grids. By replicating data, the production managers can submit a set of jobs to a different grid or to spread data between sites within a grid.

End-users are now using DQ to locate and retrieve data as well as to generate POOL XML File Catalogs for their analysis.

Finally, overviews of the location of ATLAS data across all sites and all Grids are regularly taken by making extensive queries using the DQ API.

CONCLUSION

The ATLAS Data Challenges 2 was the first Data Challenges to use a full-scale Grid production. A production system was introduced and with it a new data management component.

DQ was subject to changes during this period mostly as the Data Challenges load increased and additional functionalities were required.

Since the beginning of DC2 on early July, DQ servers for NorduGrid, US Grid3 and LCG have processed more than half a million requests - the vast majority operations on the grid catalogs without data movement. To date,

around 6 TB of data have been moved using DQ servers. This is expected to increase substantially with the Tier0 exercise.

As grid middleware becomes more interoperable, it is likely that the number of DQ implementations per module is reduced. The focus on the development of DQ shall be on providing end-users with user-friendlier tools and increased functionality in terms of data management for the production managers.

ACKNOWLEDGEMENTS

The author would like to acknowledge the Faculdade de Engenharia da Universidade do Porto and CERN, European Organization for Nuclear Research, where DQ was first developed as senior thesis project.

Finally, the author would like to acknowledge the Brookhaven National Laboratory for the support on the development of the DQ end-user tools.

REFERENCES

[1] Don Quijote is described at http://cern.ch/mbranco/cern/donquijote
[2] L. Goossens [501], this conference.
[3] M. Mambelli [503], D. Rebatto [364], O. Smirnova [499], this conference.
[4] Globus Replica Location Service is described at: http://www.globus.org/rls/
[5] EDG Replica Location Service is described at: http://edg-wp2.web.cern.ch/edg-wp2/
[6] POOL File Catalog interface is described at: http://lcgapp.cern.ch/project/persist/catalog/
[7] GridFTP Protocol is described at: http://www.globus.org/datagrid/gridftp.html
[8] SRM is described at: http://sdm.lbl.gov/srm-wg/
[9] J-P. Baud [278], this conference.
[10] gSOAP is described at: http://www.cs.fsu.edu/~engelen/soap.html
[11] GSI is described at: http://www.globus.org/security/