PROCEEDINGS OF SCIENCE

PoS(ISGC 2013)015

The Dynamic Federations: federate Storage on the fly using HTTP/WebDAV and DMLite

Fabrizio Furano*
European Organization for Nuclear Research (CERN)
E-mail: fabrizio.furano@cern.ch

Ricardo Brito da Rocha

European Organization for Nuclear Research (CERN)

Adrien Devresse

European Organization for Nuclear Research (CERN)

Oliver Keeble

European Organization for Nuclear Research (CERN)

Alejandro Álvarez Ayllón

European Organization for Nuclear Research (CERN)

Patrick Fuhrmann

Deutsches Elektronen-Synchrotron (DESY)

PROCEEDINGS OF SCIENCE

Recently the importance of clustering storage nodes across site boundaries is becoming more clear, thanks also to the recent ongoing initiatives in the CMS and ATLAS experiments. These approaches are supposed to promote simplicity in accessing the data and offering new possibilities for resilience and data placement strategies, that may also lead to a better utilization of the available CPU slots.

Here we report on work that seeks to exploit the federation potential of redirectable protocols like HTTP/WebDAV and build a dynamic, scalable, persistency-free system that offers a unique view of the storage and metadata ensemble and the possibility of integration of other compatible resources such as those from cloud providers. The challenge, here undertaken by the providers of dCache and DPM, partially in the context of EMI-data, and pragmatically open to any other Grid and Cloud storage solutions, is to build such a system while being able to accommodate name translations from existing catalogues (e.g. LFCs), experiment- based metadata catalogues, or stateless algorithmic name translations, also known as "trivial file catalogues".

Other technical challenges that will determine the success of this initiative include performance, latency and scalability, and the ability to create worldwide storage federations that are able to redirect clients to repositories that they can efficiently access, for instance trying to choose the endpoints that are closer or applying other criteria.

One of the key requirements is to use standard clients (provided by OS'es or open source distributions, e.g. Web browsers) to access an already aggregated system. This was accomplished by exploiting the possibilities offered by the DMLite framework, in order to integrate with existing frontends like the Apache server.

We believe that the features of a loosely coupled federation of open-protocols-based storage elements will open many possibilities of evolving the current computing models without disrupting them, and, at the same time, can operate with the existing infrastructures, follow their evolution path and add storage centers that can be acquired as a third-party service.

The International Symposium on Grids and Clouds (ISGC) 2013
17 - 22 March 2013
Academia Sinica, Taipei, Taiwan

*Speaker.

PoS(ISGC 2013)015

1. Introduction

In this paper we describe the Storage federation system that we designed and built to match with the existing and upcoming Grid-related data management architectures. The system is able to federate storage sites and metadata endpoints that expose a suitable data access protocol, into a transparent, high performance storage federation that exposes a unique name space. The architecture can accommodate LFN/PFN algorithmic name translations without the need of catalogues. On the other hand, if catalogues are needed, several of them can be accommodated into the same federation, and their content presented as merged. In this latter case the federation mechanisms can also be used to crosscheck against the federated storage the existence of the replicas that are being requested by clients.
The idea is to allow applications to access a globally distributed repository, to which sites participate. The applications would be able to efficiently access data that is spread through different sites, by means of a redirection mechanism that is supported by the data access protocol that is used and that is fed by a dynamic system that can locate resources on the fly.
The focus is on "standard protocols" for data access, like HTTP and WebDAV, and NFS can be considered as well. In our design, the architecture and the components of such a system are detached from the actual protocol that is used.
Additional focus of our design is on the fact that a federation may be composed by distant sites, and the redirection choices have to take this into account, without imposing the need of partitioning a federation into smaller ones on a geographical basis, or partitioning the name space.
Another point that is important for our design is that such a system should be efficient also in the browsing case, e.g. allowing an user to list the content of a directory in a fast and reliable way that does not impact the performance of the whole system.

2. The goal and the available components

The purpose of the project is *being able to aggregate storage and metadata farms exposing standard protocols that support redirections and WAN data access, making them behave as a unique system, building the illusion of a unique namespace from a set of distinct endpoints, being able to accommodate also explicit, catalogue-based indexing.* The more notable examples of suitable protocols are HTTP/WebDAV and NFS 4.1.

Such a federation of sites has to be intended as a set of storage endpoints (and/or replica location catalogues or name translators) that is:

- protocol-homogeneous (a client that uses a particular protocol must be able to operate with all the members of the federation, using that protocol)
- namespace-homogeneous (each file is identified by a unique string key, which we call path/name. If two storage endpoints have a file with the same name, then they are two replicas of the same file.)

Although creating several distinct federations is always possible, the most interesting case is obviously the one for which there is only one big efficient federation, with a logical unique entry

point that eventually could be replicated in several places. For the purposes of our project it's immaterial if this big federation aggregates sub-federations or storage endpoints or name translators or replica location catalogues. As a comparison, the xrootd [2] federations support this kind of very wide setups by means of a mechanism called *peering*, which has a few constraints. We refer the reader to the xrootd documentation in the case they are not familiar with the concept.

The Grid software gives the possibility of choosing among several components the ones that are more suitable for handling the storage and metadata parts of the design of a computing model. Some of them are, for example DPM, LFC, dCache. Nowadays we have also to start considering as parts of the solution the upcoming evolutions of these components [3] [4], that are headed to supporting standard protocols like HTTP, WebDAV and NFS4.1 in the context of scientific computing. On top of this we must also consider the opportunity of acquiring storage as an external service, as an additional kind of endpoint to fit in a modern design.
The goal of the Dynamic Federations project is to give efficient tools that are able to accommodate in a coherent way the variety of storage and metadata endpoints that a distributed, heterogeneous deployment of storage farms will make available. At the same time, the guidelines of the project are to privilege the aspects that are related to performance, scalability and usability of the federation services.

3. Some use cases

We describe here a few use cases for our "storage federation engine", that we consider as clear examples of the features that our system provides and as deployment use cases that have been tested with the system we designed. These are not intended to be precise specifications of the system. Of course the list is not exhaustive, given the flexibility of the concept. Moreover, the various points do not exclude each other.

In the case of a big loosely coupled federation, the choices to redirect a client to one repository or another should be based at least on the availability of the requested resource in that endpoint. Other metrics can be considered, like the geographical location of the client with respect to the various possible servers, and/or the load of the endpoints. In other words, a client in Switzerland should not be redirected to read data from Taipei, unless the requested data is hosted only in Taipei.

We would like to recall that we are treating data access protocols that natively support redirections. The idea is that an application would use only a "standard" client (like a Web browser or an application using an HTTP client) to access the data, without additional client-side software layers that emulate the aggregation of the storage centers.

3.1 DPM and dCache via WebDAV

Given a number of storage endpoints deploying the WebDAV door of the dCache system, and the upcoming versions of DPM [3] [4], we wanted to show that a completely transparent federation of them was possible, using the WebDAV protocol. This use case has been the first one to be demoed by the Dynamic Federations project, and the first two endpoints that were added

to a working federation have been a dCache instance at DESY (Germany) and a DPM instance in ASGC (Taipei). The test did what it advertised, i.e. the users could not realize that they were browsing and using a federation of two distant sites. Moreover, the feeling of performance that the system gives is the one of a site that is hosted in the federation's frontend machine, with a fast and smooth interactivity.

3.2 Add third-party storage farms

We cite here what was the first formulation of this use case to be fulfilled by the Dynamic Federations project:
We buy from a company a service consisting in 100PB of high performance storage, located in a remote server farm. The company only allows the use of the WebDAV/HTTP protocol to access it, since its technicians do not know anything else, and do not want to internally expose their infrastructure to unknown sophisticated systems, by installing them.
We want the clients to be able to see this service through the same entry point that aggregates other similar services in a completely transparent way, at least for the data reading case. We don't want the client applications to be instrumented in order to accommodate this case. We don't want a Data Management system to treat this case as an exception of some kind.
This would allow users to browse their files using Internet Explorer without potentially being aware of the location of the items they see, and to run their personal analyses pointing their applications to the unique entry point, using the URLs that they see in the browser.

This use case may also accommodate the use case of the "Cloud storage providers". Technically, we chose to use a Cloud storage service provided by T-Mobile (Germany) through WebDAV, which then became a standard component of the various demos of the Dynamic Federations system.

3.3 Create a small local federation of close sites sharing storage

Having a flexible system that can manage storage federations opens many possibilities. One of them is being able to create a small federation of sites that share their storage, and appear as a unique storage element. In other words, a common repository (for example an instance of the so-called "conditions data" for a High Energy Physics experiment) may be distributed across collaborating sites. Doing so, the clients would not need to know the exact location of the file they need, as they would just access it through the federation frontend.

3.4 Add resources managed by one or more LFCs

We informally define an *LFC cloud* as a set of storage elements that contain file replicas that are indexed by an instance of an LFC file catalogue. In this example, without loss of generality, we suppose that the content of the LFC is accessible through an HTTP/DAV gateway, like the ones that have been recently released [3]. Let's suppose that we have two such clouds, as the example fits with no changes also the case in which there are more.

A "storage federation engine" acts as unique entry point for the two clouds, by hiding the fact that they are two. Hence, an HTTP client would contact the main federations frontend, and will

have access to all the metadata from there, because the frontend machine aggregates and caches on the fly the results of the metadata queries that are forwarded to the endpoints.
If the client issues an HTTP GET request towards the frontend, this will simply redirect it to one of the endpoints that have just advertised the availability of the requested file.
In other words, the client can be redirected to the best endpoint in the most suitable LFC cloud, with a decision based on:

- real availability of the file in the two clouds
- possibly, proximity of the client with respect to all the available endpoints.

The interesting aspect in this example is that no additional indexing of the files is needed to federate the two LFC clouds, and their internal, local workflow can remain untouched.

3.5 Federating file caches

The fact that our Dynamic Federations system applies a dynamic behavior to the problem of federating storage and metadata endpoints opens the possibility of federating storage endpoints whose content may change at a faster pace with respect to a regular storage element. This is the case, for example, of a storage cluster that acts as a file cache, hence files may appear and disappear, depending on the pattern of the file requests that it receives.
Such a storage federation that includes caches among its endpoints would have the benefits that come from both concepts:

- caches would provide their service to the site they belong to, fetching files from elsewhere and keeping them while they are being actively used
- the same caches would advertise the files that they currently contain to the federation system.

The outcome of these two points is that the content of a file cache in a given moment could be used through the federation frontend by some other external client, or, eventually, by some similar file cache system that is trying to fetch the file. As a consequence, the federation frontend would have more endpoints to choose from when asked to redirect a client to a suitable server that hosts a file resource. This aspect, coupled with some other smart endpoint choosing criteria like e.g. geographic proximity, would represent a very relevant feature for a Grid-aware setup. So far, work is foreseen to verify the usability, in the described context, of the Scalable Proxy Caches [9].

3.6 Allow the system to apply geography-aware redirection choices

A feature that we implemented in our system internally associates geographical coordinates and information to each replica that is known to the federation, by invoking a loadable "Geo" plugin. The same plugin can associate this information to each client request for locating a replica. As a consequence, the system can select the replica that is geographically closer to the client that requested it.
An easy implementation of the Geo plugin consisted in wrapping the GeoIP API [8], that seems to provide a more than adequate level of performance (on the order of one million queries per second, as output by its internal tests). The result is that the Dynamic Federation system is able to redirect a client to the replica that is the closest to it, in a very efficient way.

4. The system

The Dynamic Federations system is built around a new internal component that was called Uniform Generic Redirector (UGR). The UGR exposes an API, called *UgrConnector* that gives the functionalities of a namespace, thus including file/replica metadata information and directory listing information.

As visible in Figure 1, the UGR acts as loader of a set of plugins, which interface it to the external storage and metadata endpoints. Each kind of plugin can talk to a different kind of external endpoint (one or many in principle).
Right now we developed the following plugins:

- A WebDAV/HTTP client plugin, that is able to talk to an external WebDAV/HTTP server endpoint. This plugin is based on an implementation of an advanced wrapper client interface (called DAVIX), built using *libneon* [10]. This component supports several kinds of authentication, and supports advanced metadata primitives (e.g. getting file listings with all the metadata information of the items, in a single transaction).
- A DMLite-client plugin, that is able to use an instance of DMLite as a source of metadata information. This allows using all the possibilities of integration offered by DMLite, for instance to connect natively to an LFC database or to an HDFS cluster.
- A plugin using the GFAL2[11] framework, whose primary use case so far has been to allow connecting to legacy LHC File Catalog (LFC) services.

All the plugins that have been developed so far privilege the internal parallelism, i.e. they are able to perform N tasks in parallel, where N is a parameter that needs to be tuned in order to find the right balance between the overall system performance and the load that can be put towards the endpoints.
The set of plugins that are loaded by an UGR instance (and their configuration) is written in a configuration file. Each plugin can be loaded multiple times, with different parameters and prefix-based filename translations.

The typical use of the UGR is to be loaded by some other frontend system, like for instance the DMLite library [4]. In this form, UGR acts as a DMLite plugin, taking full benefit of its architecture and behavior. DMLite is a pluggable, thin software layer that gives abstract functionalities of file catalogue and interface to storage pools. Thanks to its architecture, a very broad range of storage systems can be accessed, through suitable plugins. DMLite can also be plugged into an Apache server, thus accessing all of its features through WebDAV.
One of the consequences of this is that, through DMLite, plugging the UGR into an Apache server becomes easily feasible, as shown in Figure 1.

Internally, the UGR acts as a sophisticated handler of parallel requests for metadata information. The basic behavior on the trigger of a metadata query is as follows:

- if the query can be satisfied by the local in-memory namespace cache, just use the cache to compute the result. Otherwise:

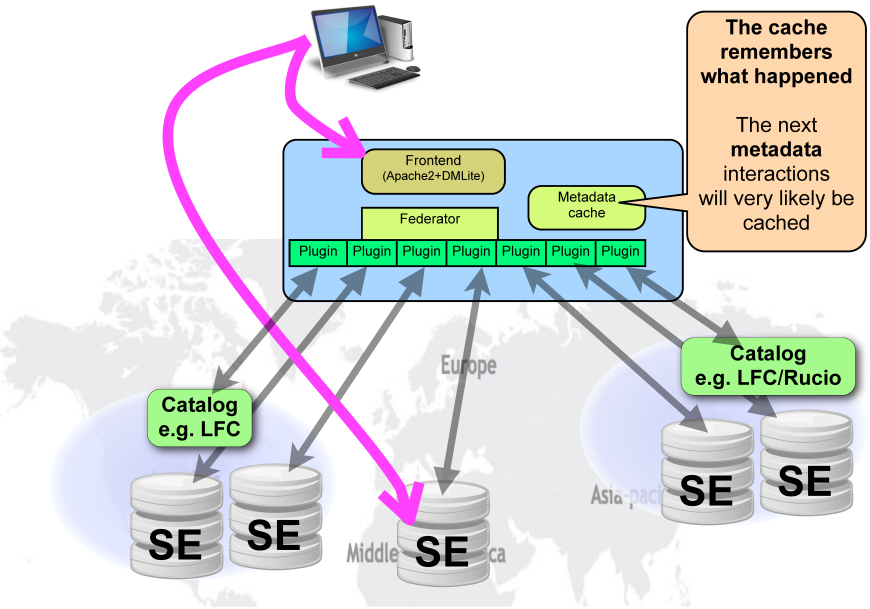

Figure 1: Exemplification of the system architecture.

- trigger, in parallel, all the plugins by queueing the query into them, then wait for the result.
- each plugin may internally decompose the query into subqueries (also parallel, depending on the plugin)
- each plugin acts independently in order to satisfy the query and write its result into the namespace cache
- when the gathered information is sufficient for that client to get the result, that client only is signalled so that it can get the desired information.
- the plugins that eventually did not finish the processing just continue, eventually updating the content of the cache with the information they may still gather.

This sequence of actions is performed for any client, in parallel, with no imposed limits to their concurrency inside UGR.

In the next sections we show sequence diagrams that explain with some more details the internal behavior of such a system, in two relevant cases.

4.1 Two clients issue a stat() request

In this example, shown in Figure 2 two clients want to know some metadata information about file X, e.g. its size. Hence, they invoke the API of the UGR service and get the response. The response is constructed by the aggregation service on the base of the responses of the various plugins.

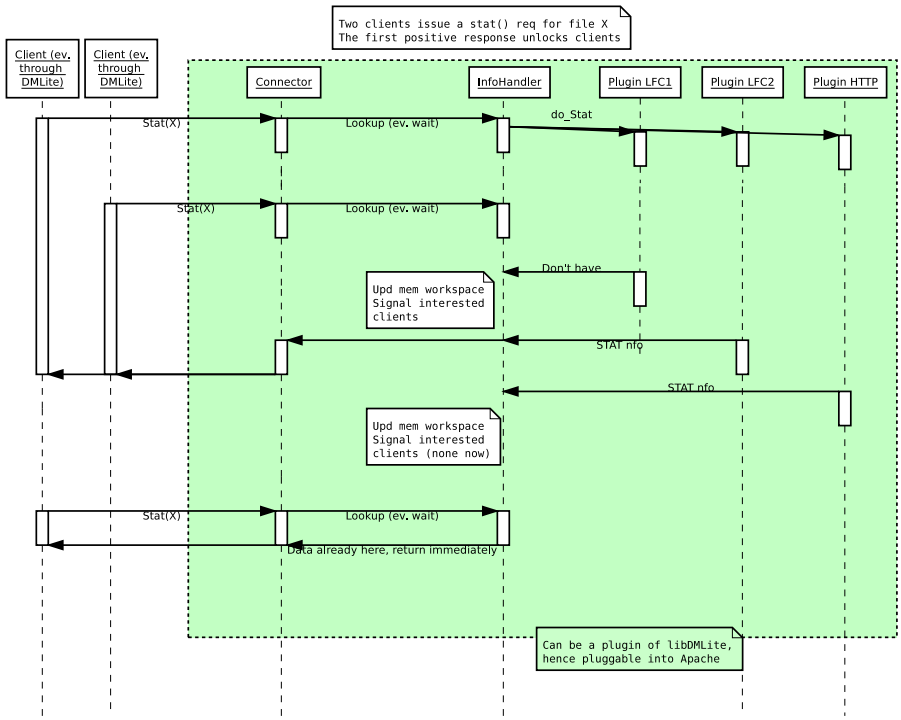

Figure 2: Clients issuing a Stat() request for the same file.

The clients get the response as soon as the service gets it from the endpoints that it aggregates. A successive query for the same information gets a cached answer, as shown in the diagram.

Any primitive describing this behavior (e.g. getting the size of a file) could give the name to this operation, should the reader be uncomfortable with the choice of *stat* of this example.

As previously said, this diagram wants to describe the basic internal interactions of such an aggregator service. The term Client refers here to any system that is able to invoke the API, single or multithreaded. As discussed, in the current deployments this client is an instance of DMLite embedded into an Apache server.

4.2 Two clients request the full list of the replicas of a file

In this example, shown in Figure 3, two clients want to know the list of the locations of file X, eventually through name translations. Hence, they invoke the API of the UGR service and get the response. The response is constructed by the aggregation service on the base of the responses of the various plugins.

Any primitive with this behavior (e.g. getting a list of replicas) could give the name to this operation, should the reader be uncomfortable with the choice of *locateall* of this example. Some implementations refer to this functionality with *getreplicas*. We wanted to use a different term in

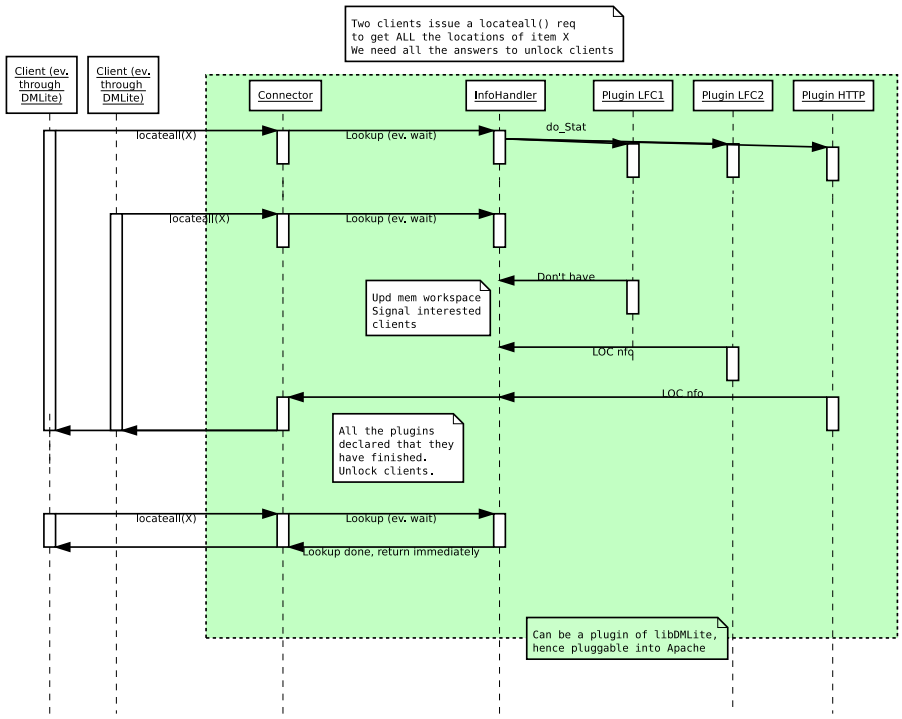

Figure 3: Clients trying to discover all the locations of the same file.

order to emphasize the fact that this specification is implementation-agnostic.

As previously said, this diagram wants to describe the basic internal interactions of such an aggregator service. The term Client refers here to any system that is able to invoke the API, single or multithreaded. In at least the DPM/LFC deployments this client will likely be an instance of the DMLite library [4].

The difference with respect to the previous case is that in order to discover all the replicas of a file the system has to wait for all the plugins to have finished. In the previous case instead, getting the size of a file just need the answer of the fastest of the endpoints.

5. Current status and conclusions

In this work we have shortly described the Grid-compatible Storage federation system that we have designed and are evaluating as a prototype. We believe that this kind of system represents a natural evolution of the current data management schemes, which are based on persistent file indexes and explicit file registration done by the applications. At the same time, the framework can accommodate the existing workflows as well. With respect to this, metaphorically we could say that, if currently data lives on distant islands of storage and catalogues are the maps, the Dynamic Federations approach is like having a realtime satellite-based, high resolution view of the world. This is confirmed by the feeling that operating with the system gives, i.e. the interaction is com-

pletely seamless and the performance is very high.
We think that this kind of fast, transparent system gives more sense to concepts that often are linked to the concept of federations, like e.g. the failover to data on another island in case of data access problems, or avoiding inconsistencies, just looking at where the files are now. Technically, today our system can federate multiple instances of: dCache, DPM, LFC and WebDAV-based Cloud storage services, and can be extended to other metadata sources. The fact of having designed the system in order to accommodate geography-based weighing of clients and replicas gives one more functionality that may be determinant for the productive usage of Wide Area Networks, for both transfers and direct, interactive access. Right now, the geographical plugin that was written uses the GeoIP libraries and the GeoLite data [8], however, others are possible, given the generality of the concept. In the informal performance tests done so far, the system core shows a good performance level, with peaks of several hundred thousands cache hits per second, including the GeoIP lookups; the demo system that was recently deployed was able to keep a steady rate of around 2K *stat* transactions per second, as seen by the client application, i.e. including all the overheads (wide area network, servers, etc.). A precise measurement of all these characteristics will be the subject of future work.

Acknowledgment

This work was partially funded by the EMI project under European Commission Grant Agreement INFSO-RI-261611.

References

[1] Scalla/xrootd WAN globalization tools: Where we are. Fabrizio Furano, Andrew Hanushevsky 2010 J. Phys.: Conf. Ser. 219 072005
http://iopscience.iop.org/1742-6596/219/7/072005/

[2] The xrootd.org homepage http://www.xrootd.org

[3] DPM: Future Proof Storage. Alejandro Alvarez, Alexandre Beche, Fabrizio Furano, Martin Hellmich, Oliver Keeble, Ricardo Rocha CHEP2012

[4] Web enabled data management with DPM & LFC. Alejandro Alvarez Ayllon, Alexandre Beche, Fabrizio Furano, Martin Hellmich, Oliver Keeble and Ricardo Brito Da Rocha CHEP2012

[5] RFC 5716 http://tools.ietf.org/html/rfc5716

[6] Data Management in HEP: an approach. Fabrizio Furano. The European Physical Journal Plus Volume 126, Number 1 (2011), 12, DOI: 10.1140/epjp/i2011-11012-2

[7] DPM components.
https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm/Dev/Components

[8] "This product includes GeoLite data created by MaxMind, available from http://www.maxmind.com/."

[9] Grid Data Access: Proxy Caches and User Views. Cristian Traian Cirstea.
Eindhoven University of Technology Stan Ackermans Institute / Software Technology
ISBN 978-90-444-1067-9

[10] *neon* is an HTTP and WebDAV client library, with a C interface.
`http://www.webdav.org/neon/`

[11] GFAL 2.0, Grid File access Library. Adrien Devresse, 2011
`https://svnweb.cern.ch/trac/lcgutil/wiki/gfal2`