Journal of Physics: Conference Series **513** (2014) 042030 doi:10.1088/1742-6596/513/4/042030

The DMLite Rucio Plugin: ATLAS data in a filesystem.

M Lassnig[1], D van Dongen[2], R Brito Da Rocha[1], A Alvarez Ayllon[1], P Calfayan[3]

[1] European Organization for Nuclear Research (CERN), 1211 Geneva, Switzerland
[2] Radboud Universiteit, 6525 Nijmegen, Netherlands
[3] Ludwig-Maximilians-Universität, 80539 Munich, Germany

E-mail: mario.lassnig@cern.ch

Abstract. Rucio is the next-generation data management system of the ATLAS experiment. Historically, clients interacted with the data management system via specialised tools, but in Rucio additional methods are provided. To support filesystem-like interaction with all ATLAS data, a plugin to the DMLite software stack has been developed. It is possible to mount Rucio as a filesystem, and execute regular filesystem operations in a POSIX fashion. This is exposed via various protocols, for example, WebDAV or NFS, which then removes any dependency on Rucio for client software. The main challenge for this work is the mapping of the set-like ATLAS namespace into a hierarchical filesystem, whilst preserving the high performance features of the former. This includes listing and searching for data, creation of files, datasets and containers, and the aggregation of existing data – all within directories with potentially millions of entries. This contribution details the design and implementation of the plugin. Furthermore, an evaluation of the performance characteristics is given, to show that this approach can scale to the requirements of ATLAS physics analysis.

1. Introduction and problem statement

Historically, there have been two ways to interact with the ATLAS Data Management system Don Quijote 2 (DQ2) [1]. First, via dedicated command line clients, for example, *dq2-ls*, *dq2-get*, or *dq2-put*, or via a programmatic API in the Python programming language. This has been problematic, because users need to install these clients, use specific versions of the Python interpreter, and have a dependency on a native library that is not available for some operating systems. Additionally, the execution environments of the physics analysis software and the data management system use conflicting versions of native libraries, requiring workarounds that are troublesome to maintain. This is made worse through the deployment of many different versions of the DQ2 clients, to support the incremental conversion of the ATLAS software stack in need

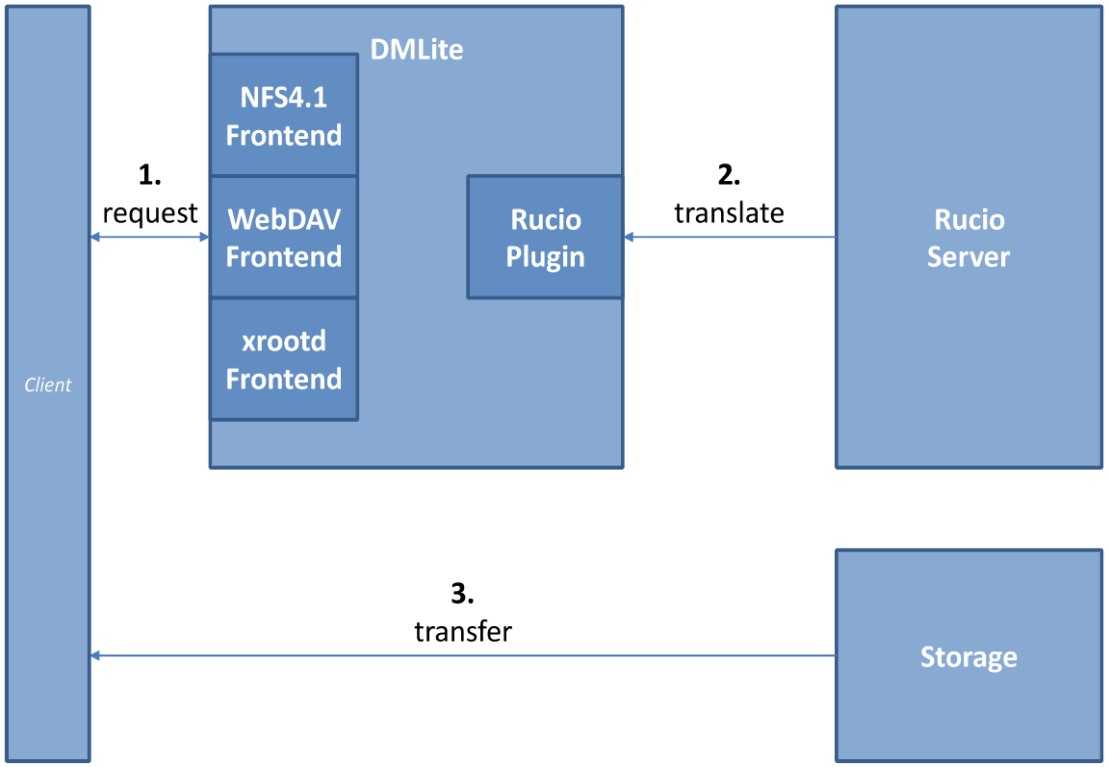

Figure 1: Component interaction of the Rucio Redirected Storage.

of data management. As a result, the client side of DQ2 is not easily portable, not user friendly, and has been shown to be an operational burden to the experiment.

The successor project of DQ2, called Rucio [2], provides a different strategy to access data, called the *Rucio Redirected Storage*. There are no specific clients to install, and existing, standardised, protocols are used. This allows operating systems to interact with the world-wide distributed data as if it were a local filesystem. This also eliminates the need for the experiment to write, maintain, and deploy data management clients. Additionally, the namespace of the data management is translated and exposed like a hierarchical filesystem, instead of its native view with potentially overlapping datasets.

2. Design

The basic design of the Rucio Redirected Storage is illustrated in Figure 1. A client requests the location or content of a data identifier from one of the available frontends of the DMLite software stack [3]. This request is then internally remodelled into a POSIX-like data structure for filesystem calls, with certain extensions required for Grid Computing, for example, the replica location. The request is then given to any available plugin of DMLite – in this case, the Rucio plugin. The Rucio plugin uses the content

of the data structure to construct a request to the Rucio server, which translates the information into a list of specific storage access URLs. Then, the data structure is filled with the translated information from the Rucio server by the plugin, and returned via the corresponding DMLite Frontend back to the client in its native protocol. The client can then use the native protocol to access the storage, and to transfer the file eventually.

3. Implementation

The plugin is written in C++, built with cmake, and linked against DMLite and the json-c libraries. The WebDAV Frontend of DMlite is a plugin to the Apache HTTP webserver, which in turn loads the DMLite library and all available plugins.

The main issue that had to be solved in the implementation is that different WebDAV clients, for example davfs2 or a web browser, use the protocol in a different fashion. Whereas a webbrowser can store a state locally, a mounted filesystem does not have that possibility. This is important when accessing subdirectories, that is, datasets containing other datasets. A web browser would send a request containing both the previous directory and the directory it would like to open, giving a straightforward mapping of the hierarchy. This translates into an bijective call to the Rucio server. davfs2 on the other hand does not have knowledge about the current directory, as this information lies with the shell process. To solve this problem, the hierarchy of the Rucio datasets is thus established through explicit naming of the current *parent* datasets through string concatenation. While this poses redundant information for the web browser, it is unobtrusive and does not incur a performance penalty.

The second implementation issue was the handling of metalink files [4]. Metalink files allow the automatic and transparent redirection of the source files. Again, webbrowsers can automatically translate metalink and download the appropriate source, but filesystems and shells do not know about this. In turn, this functionality was added by extending the WebDAV Frontend of DMLite itself.

The third implementation issue was dealing with large result sets in general. Rucio is built to deal with large sets, however, filesystems are not. The key limiting factor seem to be in the order of about 2^{16} entries per dataset, exhausting both memory on the client side, and repeated content requests to the Rucio server. This was solved by adding a pagination feature on the plugin side, that is, incremental retrieval of directory contents through a special *continuing* directory, and by overriding shell commands to be less aggressive in their prefetching.

4. Evaluation

Figure 2 and Figure 3 show the two principal ways to interact with the system. The web browser illustrates the list of all scopes under a virtual user scope. This virtual user scope is not a real Rucio scope, but instead created by the plugin, as the distribution of scopes is heavily skewed. For example, there are many more user scopes than official data scopes, therefore mixing them with official data would not be intuitive. The top entry *.++* denotes the pagination of the scopes. As there are many scopes, continuing with *.++* retrieves the next n entries from the server. This continues down from scopes to containers, to datasets, to the files. Currently, size information, modified dates, and ownership are not yet propagated, and permissions are not enforced. The shell interface

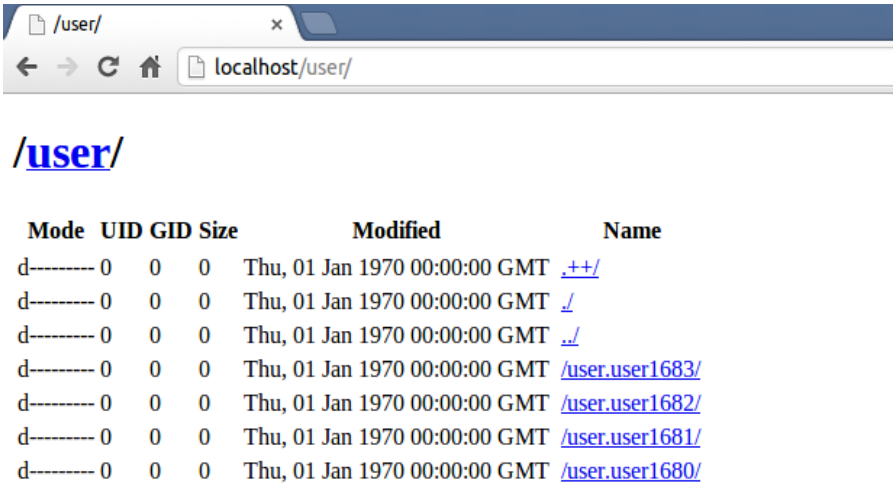

Figure 2: Browsing the WebDAV Frontend via Google Chrome. The special *.++* entry denotes the pagination of the next n entries in the list.

.

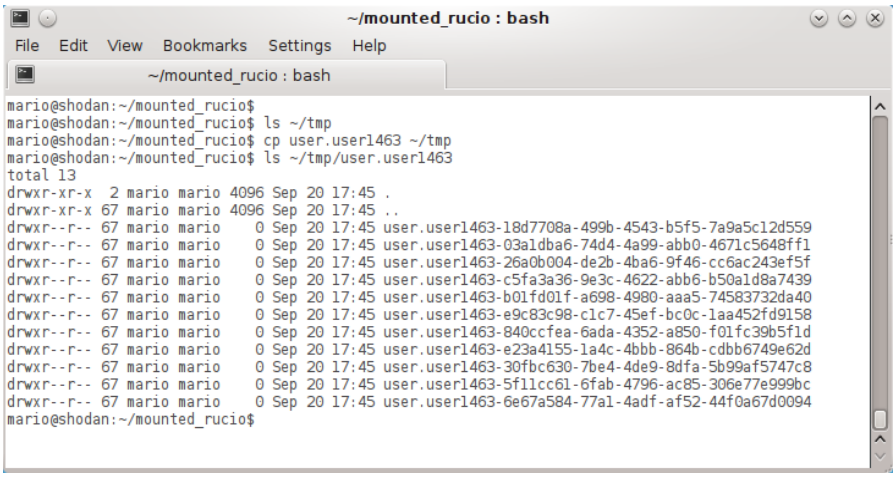

Figure 3: Mounted WebDAV Frontend via davfs2, on Linux bash. Standard GNU *cp* can be used to download the metalink files of the dataset.

has the DMLite WebDAV frontend mounted via the davfs2 FUSE filesystem [5]. Regular GNU coreutils, like *ls* or *cp* can be used. In this example, all files from a scope are copied to a local directory. These files are actually metalink files, which can then in turn be used to download the actual file from the storage. If a metalink client is used, for example, aria2c, the metalink is parsed automatically and downloads the corresponding file from the storage.

The first iteration of the tests with the plugin have been carried out against the Rucio testbed. Currently this includes 0.5 billion data identifiers (containers, datasets, files), and 1 billion replicas of the files. The plugin uses extensively the Rucio API to retrieve

the top-n entries from a given data identifier. This new API call has been implemented specifically to support this usecase for DMLite. Previously, an expensive full scan of the data identifier was used. The new API call can use the partitioned indexing structure of the database tables of the Rucio server, being bound only by network bandwidth to retrieve the results. The time it takes to retrieve the next 100 data identifiers in bulk is in the millisecond range. The IO to the database is negligible.

The main bottleneck is the parsing of the response from the Rucio server. As the communication protocol is JSON, a string-based protocol, the CPU usage to parse the strings is non-negligible. A single user from a web browser can potentially exhaust a full CPU core per instance of the plugin, by repeatedly refreshing the web browser pages of large datasets. The proxy cache keeps the database and network mostly unaffected, however, the response still needs to be parsed. It remains to be investigated how this can be avoided without having to start a non-trivial amount of DMLite instances. A potential candidate might be a custom memory cache and additional ETAG management in the plugin. An unoptimised version of a single instance of DMLite with the plugin has been stress tested on an 8-core 2.4 GHz CPU with 24GB RAM. Approximately 12 concurrent client requests for datasets with 1000 files exhaust the CPU and RAM system resources in a 5 second period. However, this includes the round-trip time to the Rucio server. If the clients are not started at the same time, but with at least 1 second serial time difference, then approximately 35 clients can be served. A potential deployment strategy therefore must take round-trip time to the Rucio server in account. An optimised version would eliminate a large fraction of that Rucio communication delay.

5. Conclusion

Rucio provides a new way to browse and access ATLAS experimental data, exposed by standard protocols without the need of specialised clients. The native scoped namespace of Rucio is automatically translated into a filesystem-like hierarchy. The implementation was built atop the DMLite framework, and communicates with the Rucio server using JSON. It is possible to browse the Rucio namespace, both via web browsers and a mounted filesystem, regardless of the size of the datasets. Additionally, retrieving the files will return a metalink file, such that the client can decide from where to download the replica. This enables additional features for free, such as transparent parallel or segmented download of files.

There are two current limitations. First, there is metadata provided by Rucio, for example, the number of events in a file, which are naturally not exposable via a filesystem. POSIX supports filesystem attributes though, so this problem might be solvable in the medium-term. The second limitation is that some access protocols to the DMLite Frontends, for example, xrootd, still require native libraries on the client side. This limitation can be circumvented by using the universally available HTTP/WebDAV protocol instead. The NFS4.1 support in DMLite is untested yet with the plugin, however, as DMLite abstracts the access protocol from all plugins. Therefore NFS4.1 should work in the same way as HTTP/WebDAV.

Future work includes creation of datasets and files through DMLite, and an intelligent filter of the available replicas from the Rucio server, to distribute the load on the server-

side whenever possible. The ROOT data analysis framework also supports access to experimental data via HTTP, and an integration is foreseen.

Bibliography

[1] Branco M *et al* 2008 Managing ATLAS data on a petabyte-scale with DQ2, *J. Phys.: Conf. Ser.* 119 062017
[2] Garonne V *et al* 2012 The ATLAS Distributed Data Management project: Past and Future *J. Phys.: Conf. Ser.* 396 032045
[3] Alvarez A *et al* 2012 DPM: Future Proof Storage *J. Phys.: Conf. Ser.* 396 032015
[4] IETF 2010 The Metalink Download Description Format RFC 5854
[5] W Baumann 2009 WebDAV Linux File System http://savannah.nongnu.org/projects/davfs2