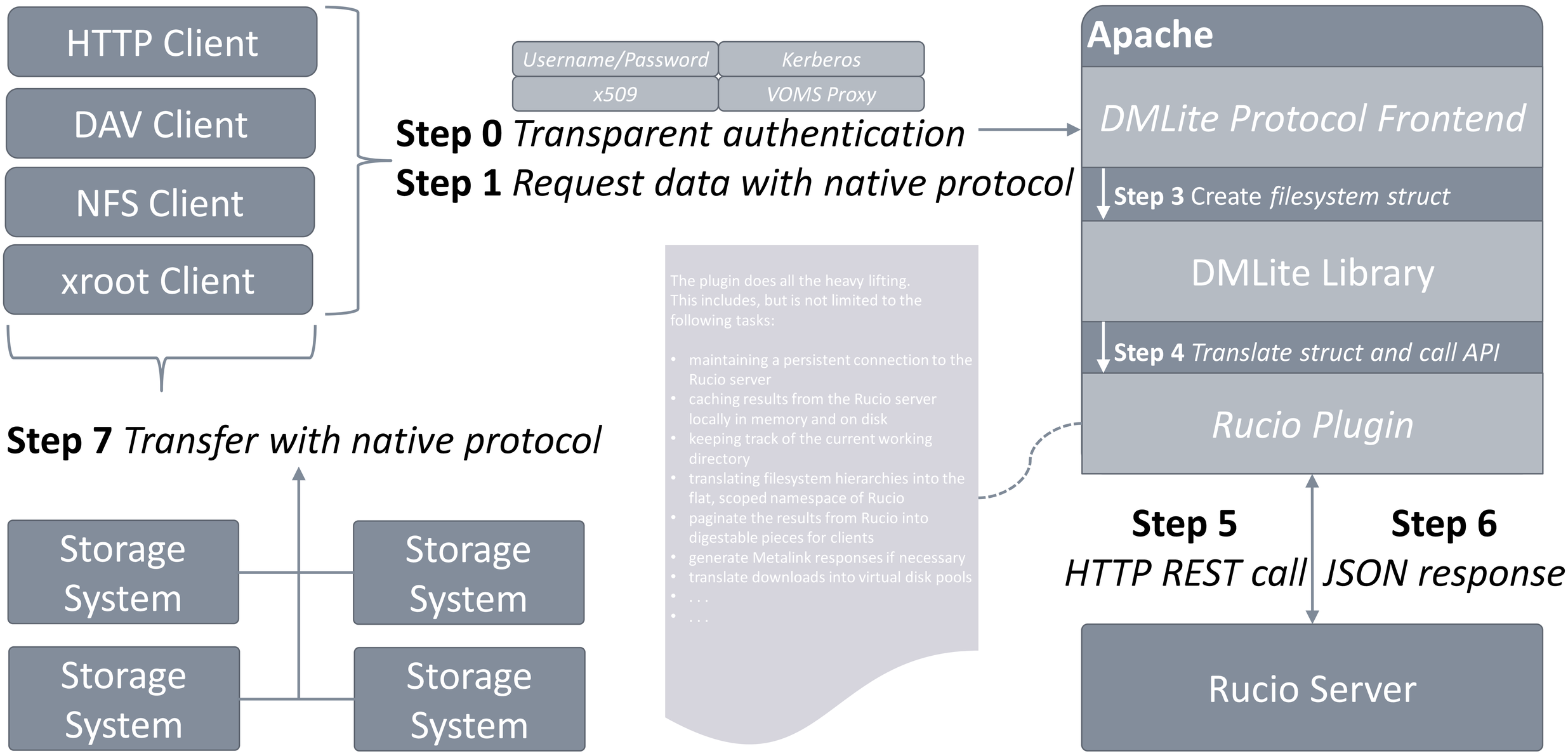
Federated Rucio Storage with DMLite

ATLAS data in a filesystem

Mario Lassnig, Daan van Dongen, Ricardo Brito da Rocha, Alejandro Alvarez Ayllon, Philippe Calfayan

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.



Component interaction of the Federated Rucio Storage

The basic design of the Rucio Federated Storage is illustrated above. A client requests the location or content of a data identifier from one of the available frontends of the DMLite software stack. This request is then internally remodelled into a POSIX-like data structure for filesystem calls, with certain extension 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.

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 dav2fs 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. dav2fs 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. 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.

Evaluation

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.

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 physics 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.

Future work includes creation of datasets and files through DMLite, and an intelligent filter of the available replicas from the Rucio server side, to distribute the load server-side whenever possible. The ROOT data analysis framework also supports access to experimental data via HTTP, and an integration is foreseen.

Bibliography

[1] Miguel Branco, et al., *Managing ATLAS data on a petabyte-scale with DQ2*, J. Phys.: Conf. Ser. 119, 062017, 2008
[2] Vincent Garonne, et al., *The ATLAS Distributed Data Management project: Past and Future*, J. Phys.: Conf. Ser. 396, 032045, 2012
[3] Alejandro Alvarez Ayllon, et al., *DPM: Future Proof Storage*, J. Phys.: Conf. Ser. 396, 032015, 2012
[4] Internet Engineering Task Force (IETF), *The Metalink Download Description Format*, RFC 5854, 2010
[5] Werner Baumann, *WebDAV Linux Filesystem*, http://savannah.nongnu.org/projects/davfs2, 2009