Making the most of cloud storage - a toolkit for exploitation by WLCG experiments

To cite this article: Alejandro Alvarez Ayllon et al 2017 J. Phys.: Conf. Ser. 898 062027

View the article online for updates and enhancements.

Related content
- FTS3: Quantitative Monitoring
  H Riahi, M Salichos, O Keeble et al.
- Towards an HTTP Ecosystem for HEP Data Access
  Fabrizio Furano, Adrien Devresse, Oliver Keeble et al.
- Resilient FTS3 service at GridKa
  T. Hartmann, J. Bubelienè, B. Hoeft et al.
Making the most of cloud storage - a toolkit for exploitation by WLCG experiments

Alejandro Alvarez Ayllon, Maria Arsuaga Rios, Georgios Bitzes, Fabrizio Furano, Oliver Keeble, Andrea Manzi
CERN IT Dept
E-mail: oliver.keele@cern.ch

Abstract. Understanding how cloud storage can be effectively used, either standalone or in support of its associated compute, is now an important consideration for WLCG. We report on a suite of extensions to familiar tools targeted at enabling the integration of cloud object stores into traditional grid infrastructures and workflows. Notable updates include support for a number of object store flavours in FTS3, Davix and gfal2, including mitigations for lack of vector reads; the extension of Dynafed to operate as a bridge between grid and cloud domains; protocol translation in FTS3; the implementation of extensions to DPM (also implemented by the dCache project) to allow 3rd party transfers over HTTP.

The result is a toolkit which facilitates data movement and access between grid and cloud infrastructures, broadening the range of workflows suitable for cloud. We report on deployment scenarios and prototype experience, explaining how, for example, an Amazon S3 or Azure allocation can be exploited by grid workflows.

1. Introduction

With the successful exploitation of cloud CPU for simulation and the increasingly attractive price of cloud resources, attention is turning to how data-intensive workflows can be supported. While wide area data access is possible, and indeed can be desirable in the case where data is only read once, for many workflows there is substantial benefit in putting data in the cloud. Possible scenarios include:

- Data which is accessed multiple times by CPU (e.g. analysis data or minimum bias, as reported in [1] and [2])
- Staging of job output to improve reliability or to retain intermediate workflow products within the cloud
- Import or export operations before or after the provisioning of CPU, to minimise total cost

While establishing a conventional grid storage system in the cloud is possible, typically through use of block devices, the overheads are considerable and, if workflows allow, it is preferable to use the provider’s own storage systems, generally an object store delivered over HTTP, such as AWS S3 [3].

Multi-science computing facilities are interested in providing resources in a uniform manner to all their users, and the S3 de-facto standard represents an attractive possibility. Conventional grid sites would enjoy cost savings if they were able to offer storage through this kind of interface.
This contribution describes a number of solutions which facilitate the task of integrating S3-like storage into WLCG workflows, thus broadening the possible uses of cloud CPU and reducing infrastructure costs.

2. Cloud Object Stores and HEP Workflows
S3-like storage represents a particular set of design tradeoffs, relaxing certain assumptions and functions in order to promote scalability and simplicity. This means that they present a new type of interface, and applications must be adapted accordingly. In this contribution we use ”cloud storage” as a generic term for the family of S3-like object stores, including Amazon S3, Google Cloud Storage, Azure, Ceph Object Gateway and Openstack Swift.

Two of the design choices, a flat namespace and eventual consistency, are unlikely to present any fundamental problem to standard HEP workflows. In general, the namespace of interest is implemented at a higher level in experiment frameworks, and the write-once read-often behaviour does not trigger inconsistent behaviour.

There are other changes which are more disruptive. The object model does not promote access to the substructure of the object, the authentication model is different, and a client must also be able to speak the new API.

This contribution describes a set of existing tools, widely used on the WLCG infrastructure, which have made the necessary adaptions and can therefore be used to interact with cloud storage with minimal modification to the application. We thus present a toolkit, with each element addressing a particular aspect of cloud storage. The integration challenges are here broken down into three main areas, summarised in Table 1.

3. Transfer
In this section we describe several solutions for data movement. FTS [4] is the file transfer service used to distribute the majority of LHC data. It uses the gfal2 [5] library for interaction with storage systems and for transfer management. The gfal2 library itself is plugin based, and for the family of HTTP protocols, including cloud object stores, it uses the Davix [6] library.

3.1. Third party copy
A key to scalable data transfer is avoiding passing data through the client. This requires support from at least one of the two endpoints involved in the transfer. The DPM [7] and dCache [8] teams have agreed on an interface and have implemented this functionality in their products, and the FTS/gfal2/Davix stack can exploit it. The interface includes gridsite delegation, which will be used automatically to delegate credentials to the endpoint, which will then itself perform the requested PUT or GET.

3.1.1. Davix invocation A transfer using the low-level Davix cli. The davix utility will use 3rd party copy by default.
davix-cp -P grid --s3secretkey <secret> --s3accesskey <access>
   davs://storage.org/path/file01
   s3s://objbkt1.s3.amazonaws.com/file01

3.1.2. gfal2 invocation
   In the example below, S3 credentials have been placed in the gfal2 configuration file. Note
   also that in gfal, a 3rd party copy must be requested explicitly by modifying the protocol.
   gfal-copy davs+3rd://storage.org/path/file01
   s3://objbkt1.s3.amazonaws.com/file01

3.2. Transfer modes with FTS
   FTS offers three methods of transferring files to and from grid storage.

![Diagram of data transfer modes with FTS]

**Figure 1.** Data Transfer Modes with FTS

3.2.1. 3rd party copy
   FTS will use third party copy (described in Section 3.1) when requested, using the adapted protocol specification described above.

3.2.2. Data pass-through
   If third party copy is not available, the FTS server can stream the data through itself transparently. While this has obvious scalability limitations, it can work well for modest data rates, or indeed if a dedicated FTS has been colocated in the destination/origin cloud. This option could be used to migrate data between cloud providers.

   The streaming of data through FTS can be used as a kind of protocol translation. In this scenario, source and destination can use different protocols, allowing for example the transfer of files from a gridftp endpoint into the cloud.

3.2.3. Multi-hop transfers
   FTS supports multi-hop transfers, where the client requests an $A \rightarrow B \rightarrow C$ transfer. A tactical storage system such as DPM or dCache, which supports third party copy, can be used as $B$ thus avoiding data passing through the FTS.

3.3. Keys and pre-signed URLs
   The examples above all involved knowledge of the S3 keys for the service, either supplied to the client on the commandline or by configuration file. The keys may also be stored on the FTS, in which case any transfers initiated by the relevant VO will be signed with the corresponding keys so that transfer can proceed. The VO of the submitter is verified by FTS using the standard grid X509/VOMS technology.

   All the above will work equally using pre-signed URLs, for example;
fts-transfer-submit --strict-copy -s https://fts3.cern.ch:8446
https://storage.org/path/file01
https://objbkt1.s3.amazonaws.com/file01?Signature=<sig>

The --strict-copy is required because FTS will otherwise try to stat the destination file (using an HTTP HEAD operation), which will fail as the signature is associated with a single method, in this case PUT.

These URLs come with their own problems, however. They can be shared and reused, within their window of validity. A lifetime which is too short can cause transfer failures, while a lifetime which is too long can be a security risk. Because of these issues, another solution to the authentication problem is discussed in section 4.

3.4. Personal cloud storage
While not part of the "S3 family", the ability to transfer files in and out of personal cloud storage is attractive. The WebFTS project has completed Dropbox integration, which allows the user to delegate rights to an FTS instance through the usual OAuth mechanism, thus allowing the FTS to manage import and export of content. A demonstration is available at https://webfts.cern.ch/.

4. Authentication
The Dynafed [9] service is an HTTP data federator, able to give seamless access to diverse data stores as backends, including traditional HTTP-enabled grid storage and S3 endpoints. By caching relevant metadata it presents a high-performance, unified namespace with a "grid standard" HTTP frontend, with X509 authentication, VOMS support and an extensible authorisation model.

As previously noted, cloud object stores have a particular authentication system and a flat namespace. This can be hidden from clients by configuring Dynafed as a gateway.

4.1. X509 support
Dynafed can be configured to hold the keys for the object store and, when accessed by a client, it creates a signed URL to which the client is transparently redirected (in standard HTTP fashion). The result is that clients can be unaware of the S3 authentication scheme and can run without modification (beyond their existing HTTP support).

Dynafed’s design as a federator means that one can easily grow the cloud storage behind by adding new endpoints. The service’s architecture is highly concurrent and horizontally scaleable, allowing it to grow with the infrastructure.

4.2. Namespaces
While object stores such as S3 do not offer a namespace as it is usually understood, they do have support for pattern matching in object names, which allows Dynafed to present, if desired, a familiar namespace for read-only clients. This allows such operations as the following

$ gfal-copy file:///home/file01 davs://federation.desy.de/myfed/dir01/
$ davix-ls -l davs://federation.desy.de/myfed/dir01/
-rwxr-xr-x 0 426454 2014-09-05 04:04:17 file01
drwxr-xr-x 0 0 1970-01-01 01:00:00 dir02

Note the directory metadata - this is indicative of the fact that this is not a genuine filesystem object but just a presentation layer offered by Dynafed.
4.3. Example use case

The CERN volunteer computing platform requires a way to allow untrusted volunteer resources to access data and to upload results, without distributing sensitive credentials such as grid proxies to them. The solution selected was to use the Ceph object store as a backend with Dynafed in front, thus providing a tactical storage system referred to as the "data bridge". This system offers the grid-standard interface to the experiment frameworks, which can place input data and retrieve results (perhaps with FTS or gfal2), and presents an alternative username/password interface to the volunteer clients, whose rights are strictly limited and who cannot use their credentials on any other storage system.

This "data bridge" is a production service which currently manages 150 TB of storage and around 1000 active volunteers performing HEP workflows (typically simulation). Further details on this use of Dynafed are reported in [10].

5. Access

Access to data stored in S3, whether from within the cloud or remotely, can be performed using the client tools previously mentioned. Data can be "gfal-copied" for example. However, for high throughput computing, the performance of data access is critical, especially when considering random, partial access to the object. This access pattern is typical of HEP analysis using ROOT, and thus support for this scenario would considerably enhance the attractiveness and utility of cloud storage.

Much effort has been invested in both ROOT and the HEP applications to achieve high performance i/o, and a lot of this has relied on the latency-hiding vector read, whereby numerous chunks of data are requested in a single GET, incurring only one round-trip time. The majority of S3 variants do not support this feature, allowing only a single "Range request", i.e. chunk, per operation. Furthermore, different S3 flavours behave differently when presented with a multirange request - some will return the whole file, some will return the first range, and some will return an error 400 (Bad Request). Thus without mitigation on the client side, these issues will severely compromise application performance and stability.

The Davix library has been integrated into the standard ROOT [11] codebase, accessible through the TDavixFile class. This alone enables cloud access for a large number of applications with minimal modification. But what about performance? Here we describe two functions of Davix designed to maximise performance. Note that these benefits are independent of authentication solutions discussed in section 4. Davix is capable of "redirection caching", meaning that even if access is initiated through Dynafed all further requests go directly to the object store without the client re-authenticating each time.

The following discussion describes a ROOT analysis on a 267MB file which is held in an object store and accessed through TDavixFile. To aid comparison, we note that ROOT’s behaviour when using the xroot [12] protocol is to generate around 30 vector reads, of roughly 180 elements each, resulting in a total read of 5MB. Tests were run 4 times, latency was introduced using Linux traffic shaping utilities (tc).

5.1. Concurrency

Davix is able to perform operations concurrently, which can radically improve application performance. Figure 2 shows the runtime of the application for different concurrency settings in Davix, under different latency conditions.

Such a tactic can reduce by factors the amount of time the application spends waiting for data to arrive. The gains available will be limited by the number of operations required, concurrency available on the client, and how the storage system is able to handle the increased load from multiple parallel requests.
5.2. Range Coalescence
In addition to the well-established use of concurrency to hide latency, Davix is also able to reduce the number of operations it performs by coalescing nearby byte-ranges, which would normally produce multiple serial requests, into a single operation.

Figure 3 shows how application runtime (i.e. wall clock) is affected by differing choices of coalescence, under varying network conditions.

The reference performance, where multi-range requests are supported and coalescing isn’t needed, is shown on the plot (the lowest, almost horizontal line). The absence of vector support can be considerably mitigated through the coalescence feature, up to the point where, in this
case, the entire file is read. This behaviour is still preferable to deciding to perform a “copy to
local” as a policy, because in the Davix case a partial read may still result, and as data is not
accessed from disk the risk of contention with other processes is eliminated.

Configuration of the coalescence range is currently available to the client through a fragment
identifier, i.e. the use of a URL such as https://server.org/path/file#mergewindow=N. The
identifier is not passed on to the server.

The behaviour of any particular analysis is highly dependent upon a number of factors,
including ROOT configuration, data file layout, storage system and network conditions. Figure
4 demonstrates an alternative scenario, with a different read profile and ROOT file, and plots
runtime against coalescence. For lower latencies, an optimum coalescence value is visible, beyond
which application run time is penalised by the download of excess, unused data.

![Coalescing range vs wall-clock time](image.png)

**Figure 4.** Wall time with coalescence, with turnover

6. Further work

Coalescence is triggered through filling in any holes between requested byte-ranges which are
smaller than a configurable size. In principle, there is an optimum configuration for any set of
network conditions. Jumping from one byte-range to the next incurs a single round-trip wait,
whereas downloading the data within that range incurs a wait of $\text{bandwidth/volume}$. Thus the
bandwidth-delay product is the crucial parameter in determining the optimal coalescence value.
Davix could be configured to detect this network characteristic and adapt the coalescence logic
appropriately.

The cloud storage market is extremely active, with new suppliers arriving and presenting
variations on the object model and its interface. One of the purposes of the Davix project is to
serve as the abstraction layer on top of this diversity. As such, the project intends to track such
developments and incorporate support where possible.

As FTS is built on gfal2 and Davix, its cloud storage capabilities will be updated as Davix
developments enable access to new infrastructures.
7. Conclusions
The integration of cloud storage systems into grid workflows presents particular challenges - the
diversity of the systems, their approach to authentication, omissions in their access protocols
(3rd party copy, vector operations) and their flat namespace.

We have presented a suite of independent but complementary advances in well established,
proven grid tools which directly address these challenges. As an ensemble, these updates
constitute significant progress in tackling the integration of S3-style cloud storage into HEP
workflows, opening the door to more data-intensive use of the cloud.

References
[1] Garzoglio G https://indico.cern.ch/event/505613/contributions/2230739/ CHEP 2016
[2] Hover J https://indico.cern.ch/event/346931/contributions/817790/ HEPiX Spring 2015
[3] Amazon S3 https://aws.amazon.com/s3/
[4] FTS Project Homepage http://fts3-service.web.cern.ch/
[5] DMC Project Homepage http://dmc.web.cern.ch/
[6] Davix Project Homepage http://dmc.web.cern.ch/projects/davix/home
[7] Alvarez A, Beche A, Furano F, Hellmich M, Keeble O and Rocha R 2012 J. Phys.: Conf. Ser. 396 032015
[8] dCache Project Homepage http://dcache.org
[9] Dynafed Project Homepage https://svnweb.cern.ch/trac/lcgdm/wiki/Dynafeds
[10] Furano F, Field L and Keeble O 2016 Physics of Particles and Nuclei Letters 13 629
[11] ROOT - An Object Oriented Data Analysis Framework http://root.cern.ch/
[12] Furano F and Hanshevsy A 2010 J. Phys.: Conf. Ser. 219 072005