A milestone for DPM (Disk Pool Manager)

Fabrizio Furano[1,*], *Oliver* Keeble[1,**], *Andrea* Manzi[1,***], and *Georgios* Bitzes[1,****]

[1]CERN - Esplanade des Particules 1, 1211 Geneva

Abstract.
The DPM (Disk Pool Manager) system is a multiprotocol scalable technology for Grid storage that supports about 130 sites for a total of about 90 Petabytes online.
The system has recently completed the development phase that had been announced in the past years, which consolidates its core component (DOME: Disk Operations Management Engine) as a full-featured high performance engine that can also be operated with standard Web clients and uses a fully documented REST-based protocol.
Together with a general improvement on performance and with a comprehensive administration command-line interface, this milestone also brings back features like the automatic disk server status detection and the volatile pools for deploying experimental disk caches.
In this contribution we also discuss the end of support for the historical DPM components (that also include a dependency on the Globus toolkit), whose deployment is now only linked to the usage of the SRM protocols, hence can be uninstalled when these are not needed any more by the site.

1 Introduction

The Disk Pool Manager (DPM) is a lightweight solution for grid-enabled disk storage management. Operated at around 130 sites, it has the widest distribution of all grid storage solutions in the WLCG infrastructure, and in total it manages around 90PB of data. DPM provides an easy way to manage and configure disk pools, and exposes multiple data access interfaces (*xrootd* [2], *GridFTP* and *HTTP/WebDAV*).

The development direction in recent years has been towards simplifying the system, while supporting all the advanced features that are needed by the Grid computing and helping sites to incrementally renew their setups. The result of this work is the creation of the *dmlite* [3] framework and the *DOME* management engine which together represent the modern DPM service. The dmlite framework and its associated plugins enabled support for the more recent data access protocols used by HEP (*HTTP with multi-range requests* and *xrootd*). DOME completes the picture as a new internal service for managing the resources which DPM exposes and is described in this contribution.

*e-mail: furano@cern.ch
**e-mail: okeeble@cern.ch
***e-mail: amanzi@cern.ch
****e-mail: gbitzes@cern.ch

2 DOME

DOME (Disk Operations Management Engine)[5] is a robust, high performance server that manages the operations of a DPM cluster. DOME communicates using HTTP and JSON with other parts of the system through the XrdHTTP plugin of the Xrootd framework. Architecturally speaking, it is primarily a service provider for the *dmlite* framework, through a dmlite plugin called *DOMEAdapter*.

DOME's adoption aims at augmenting the Disk Pool Manager (DPM) system so that its core coordination functions and inter-cluster communication paths are implemented through open protocols, and following contemporary development approaches targeting performance, scalability and maintainability.

2.1 From spacetokens to quota tokens

Historically, DPM manages space accounting through a set of individual named space reservations that are associated to pools. This follows the philosophy of the SRM specification [7], where clients' requests explicitly indicate which named space has to be accounted.

Semantically, SRM space reservations are named reservations of a part of the space of a disk pool. When writing data in workflows that involve SRM, requests to write a replica specify a spacetoken that has to host the replica, hence ultimately the replica will be subject to the space reservation rules specified by that spacetoken.

One of the weakest points in this schema is that the remote client willing to write a file has to know the name of a suitable space reservation in the destination system, to be able to write and be properly accounted for. This information represents a technical detail of the destination storage, and it can also be provided wrongly, for example accounting a file into the wrong spacetoken (e.g. "scratch" versus "production").

Another related, historical weak point of this workflow is that calculating precise results for the space occupancies can be a challenging exercise, especially if the structure of the pools has been modified in the years, following additions of new storage space or even failures.

DOME models an evolution of this mechanism towards *subdirectory-based space accounting*, instead than pool-based, in a way that is compatible with the older one and can coexist with it.

When considering subdirectory-based space accounting, every subdirectory at less than N levels from the root is kept updated with the total size of the replicas of files that reside in that directory subtree. This *subdirectory size* together with the information on free/used space in the pools associated to these subdirectory tree can then be used to compute the needed space occupancy numbers.

DOME uses the records describing spacetokens that are kept in the head node DB with minimal modification. Their meaning is slightly changed, into semantically representing a quota on one and only one directory subtree. From this point on, we will refer to them as *quotatokens*, whose behavior is similar to that of an old spacetoken associated to a directory.

Associating space reservations to directories frees clients from the need to declare for each upload the name of the spacetoken that should be used. The association will be automatic since it's based on the path of the file.

A *quotatoken* attached to a directory subtree **overrides others that may be attached to its parents**.

If the content of a directory (counting all the replicas) exceeds the quota specified by the quotatoken that influences it, then new PUT requests on that directory will be denied. In an analogous way, directories that do not have or inherit any quotatoken cannot be written into, assuming that having no quotatoken means having no space.

As a summary, the meaning of a quotatoken specifying a quota of N terabytes on pool X, associated to directory "/dir1" is *use pool X as space for hosting the files that will be written into dir1. Do not allow more than N terabytes to be hosted there*. A consequence of this is that directories that do not have or inherit a quotatoken cannot be written into.

3 DOME Architecture

Figure 1 shows the main components of DOME that are in action in a disk server.
Requests containing DOME commands come through the XrdHTTP plugin of the Xrootd framework already authenticated and referring to the DOME command path, which starts with */domedisk*. Requests related to data traffic go through the respective frontend daemon (Apache HTTPd, gridftp, xrootd). Another detail that characterizes a disk server for DOME is the ability to execute local tasks like checksum calculations and file pulls from external sources. These tasks, following a logic that is based on time, report their status to the head node, which uses this information to keep its queues updated.

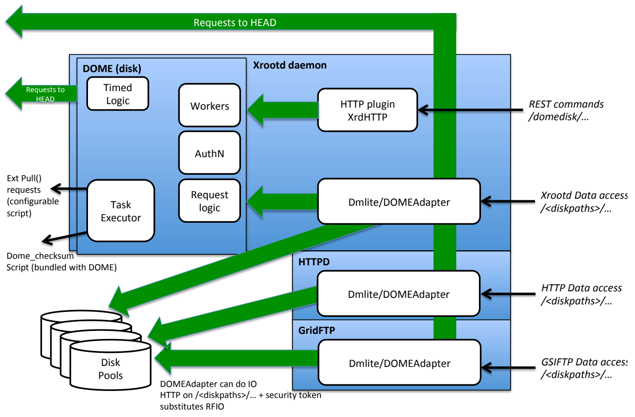
Requests to HEAD
DOME (disk)
Xrootd daemon
Requests to HEAD
Timed Logic
Workers
HTTP plugin XrdHTTP
REST commands /domedisk/...
AuthN
Ext Pull() requests (configurable script)
Task Executor
Request logic
Dmlite/DOMEAdapter
Xrootd Data access /<diskpaths>/...
Dome_checksum Script (bundled with DOME)
HTTPD
Dmlite/DOMEAdapter
HTTP Data access /<diskpaths>/...
GridFTP
Disk Pools
Dmlite/DOMEAdapter
GSIFTP Data access /<diskpaths>/...
DOMEAdapter can do IO HTTP on /<diskpaths>/... + security token substitutes RFIO

Figure 1. Simplified diagram of DOME in a disk node.

A DOME head node is slightly more complex than a disk server, and its simplified internal structure is visible in Figure 2.

Requests containing DOME commands come through the XrdHTTP plugin of the Xrootd framework already authenticated and referring to the DOME command path, which starts with */domehead*. Requests related to data traffic go through the respective frontend daemon (Apache HTTPd, gridftp, xrootd). A head node can contact external systems to get information about remote files, and queues in memory the requests for checksum calculations and remote file pulling for the whole cluster. On top of that, a head node DOME daemon also contains an embedded write-through cache for file metadata items, that considerably speeds up its operations.

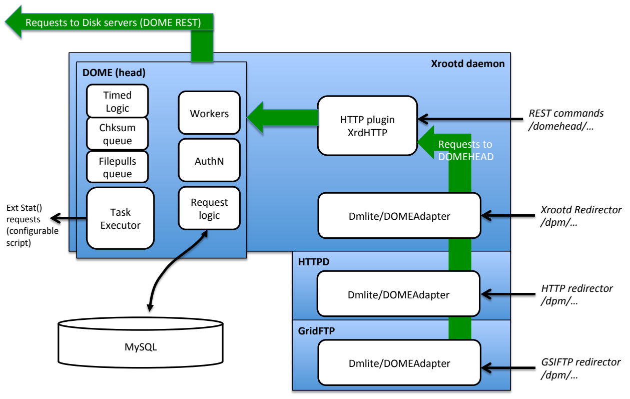

Figure 2. Simplified diagram of DOME in a head node.

4 Storage Resource Reporting

As part of DPM's policy of allowing SRM-less operation, the ability to query space occupancy has been added in accordance with the recommendations of the WLCG Accounting Task Force. The DPM distribution now includes a publisher script that can query DOME's internal accounting and copy the result as a normal file into the namespace. The file is a json document that describes the used and free space in all the independent areas (i.e. Quota Tokens) of the system. A summary for the whole system can thereby be computed simply by summing the capacities and occupancies of the individual areas. DPM systems publishing this json summary will enable WLCG accounting to include reliable storage accounting for the first time.

The distribution also contains an information provider which will publish basic system and resource metrics to the local resource BDII and which will work in the absence of the legacy components.

5 Caching with volatile pools

A major functional advantage of the DOME system is that it allows the configuration of so called "volatile pools" which enable the DPM to operate as a cache. If a Quota Token is defined on a volatile pool and then assigned to a path in the namespace, accesses via any of the supported protocols (GridFTP, HTTP or Root) to that path will trigger a file pull if the object requested is not already resident locally.

This behaviour allows DPM to operate as a caching proxy, fetching data as required from upstream storage, which could be a nearby large custodial system or a data federation. The DPM administrator supplies a script which is triggered on the disk server when a pull is required to get a non-resident file. The pull script has the responsibility to retrieve the full file. If there is insufficient available space, DPM will remove the oldest files from the cache before invoking the pull script.

Pull requests are queued in memory and dispatched to disk nodes that match the request and become available. The disk nodes instances constantly update the head node on the

running callouts, and the system will self-heal on restarts of the head node. When finished pulling a file, a disk node will notify the head node and pass the result (or failure).
The system can be configured with the following limits

- No more than L pulls will be run per disk server
- No more than M pulls will be run in total
- No more than N pulls will be run per disk mount

The pull callout in the disk server is complemented by a stat callout, which is able to get information from an external system for the presence of an offline file.

6 Admin interface: dmlite-shell

With the introduction of DOME and the deprecation of the legacy daemons, there has been also the need to replace the admin commands which currently are using the DPM and DPNS API. This activity has started some years ago with the implementation of a new python component, the *dmlite-shell*, which in its first incarnation implemented a CLI using underneath the dmlite interface to perform DPM namespace operation only. With the time the component has grown and new commands have been added (48 commands are available at the moment) in the following areas:

- namespace management: acls, checksums, replicas, files, folders
- user management: user, groups
- storage management: pool, filesystems, drain
- quota management: quotatokens

The introduction of DOME has also required the implementation of a REST client (which uses Davix [10]) for some operations that are not part of the dmlite interface. In particular every command dealing with Quotatokens is interacting with the DOME server via its REST APIs, using as identity the x509 certificate of the DPM hosts. As a result, all DPM management operations can now be performed through the dmlite-shell.

6.1 Drain via HTTP

The drain and replication subsystems have been totally rewritten in the dmlite-shell. In this case not only the interface has changed, but also the way the data replication is performed.

In particular, the drain mechanism has historically used rfiod([8]) to move data between diskservers; due to the end of support for the old rfiod client and server we decided to use the LCGDM-DAV component instead and perform the replication via the HTTP protocol.
The *drainpool, drainserver, drainfs* and the *replicate* commands are now contacting the LCGDM-DAV server running on the DPM headnode in order to put a new replica of the file to be drained or to be replicated. The process uses the internal *PoolManager* interface to request a new location of the replica, eventually hinting the pool/server/fs where to put the new replica into.
Then, as for a normal HTTP third-party copy, the process issues a *COPY* request disabling the credentials delegation to speed up the process (as we are moving data within a DPM instance this is not needed). The COPY process is monitored via Performance markers and in the case of draining the original replica is removed from the source filesystem only if the copy succeeds.

7 Configuration tool: Puppet

During the last years a lot of effort has been spent to improve the configuration subsystem for DPM, with the goal to use a more industry oriented product compared to the YAIM tool [11], which was developed for the Grid world. Puppet([12]) was selected as configuration tool, for various reasons including:

- Puppet is one of the most used Configuration Management tools, also by sites already running DPM
- Puppet allows DPM configuration integration for sites already running a Puppet infrastructure, but also it can run as a local script like YAIM
- CERN moved the whole infrastructure to Puppet, therefore it was possible to make use of the local expertise to code the DPM Puppet modules
- There is high availability of Puppet modules already developed and shared via a common portal (PuppetForge ([13]))

Given that DPM is composed by several components, the decision was to implement Puppet modules for each of them, plus a meta module (*puppet-dpm* [14] that makes use of the modules to setup a DPM instance.

The DPM team developed the following modules: *puppet-gridftp, puppet-dmlite, puppet-lcgdm, pupet-xrootd, puppet-dpm*, other modules have been developed by other teams at CERN for their needs, (puppet-voms, puppet-bdii, puppet-fetch-crl) and third-party modules are included (puppet-mysql, puppet-memcached, puppet-firewall).

All necessary Puppet modules are distributed in one of the RPM packages that belong to a DPM release, in order to make sure that the sysadmin has always the correct version of them, i.e. the one that is known to work with the DPM version he has installed. For development purposes, the modules are also available via PuppetForge.

The DPM puppet modules simplify considerably the transition from the legacy DPM deployments to new DOME based installations. The site admin can easily switch to DOME an existing installation, or directly install DPM with DOME from scratch without the legacy daemons being present.

8 Transitioning steps

As we remarked, the new components are totally compatible with the older ones and can coexist with them, if properly configured. This was an important design decision. Care has been taken to allow sites to schedule the activation of DOME at their convenience, decoupled both from the deployment of the relevant update (DOME can remain dormant if the system is configured in "legacy mode") and from synchronising with experiments.

The transition steps enumerated below each represent a working system which could be maintained indefinitely.

- Starting with the release of DPM 1.9 (which includes DOME) in Q4 2016, the sites can start upgrading head nodes and disk servers at the pace that they prefer. This allows deployment, in a dormant state, of DOME.
- Dome can be configured, while disabled, giving access to a more powerful *dmlite-shell* administration utility.
- The space reporting counters can now be primed. From this moment on, the site will precisely keep track of the disk space used (not the free space yet).

- The dmlite-shell can be used to associate each of the existing spacetokens to a suitable directory in the logical name space of the Virtual Organization they belong to.
- DOME can now be fully enabled. The system is running both the historical software stack and the new one, at the same time. All the recent features are enabled, including the ability of producing sophisticated per-directory free/used space reports. The older stack is used only for SRM services.
- Once the SRM services are no longer used, the system administrator can safely choose to uninstall the older components.

9 End of support for legacy components

The transition to the components described in this paper was motivated in part by the difficulty of maintaining software libraries that were written in the 80s and 90s [9], on which the oldest core components of DPM are based.

Since SRM has been designated an optional service for WLCG storage systems, a date has been agreed for the end of support for the full legacy stack. These components, often referred to as "lcg-dm", will be supported until end of May 2019, after which date DPM support will just advise sites to reconfigure DPM in order to use the newer components based on DOME.
This includes advising user communities to stop using unsupported data protocols like SRM, rfio, dpns, dpm-daemon. The "lcg-dm" components will remain in the public EPEL repositories for an undefined time, as the DPM team will not take any action to remove them.

10 Conclusion

The introduction of DOME represents the final component in DPM's definitive architecture. The project's platform for the future has now reached its final form and foresees just component consolidations. With a fully modernised, maintainable stack, scaleable architecture and rich protocol support, DPM is able to protect the investment of the large number of sites who have entrusted their data to it and can expect to absorb future data volumes without major architectural modifications.

References

[1] Furano F, Hanushevsky A *2010 Scalla/xrootd WAN globalization tools: Where we are* 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] Alvarez Ayllon A, Beche A, Furano F, Hellmich M, Keeble O and Brito Da Rocha R *DPM: Future Proof Storage* CHEP 2012 J. Phys.: Conf. Ser. 396 032015 DOI: 10.1088/1742-6596/396/3/032015 https://iopscience.iop.org/article/10.1088/1742-6596/396/3/032015/meta

[4] Alvarez Ayllon A, Beche A, Furano F, Hellmich M, Keeble O, Brito Da Rocha R *Web enabled data management with DPM & LFC* CHEP 2012 J. Phys.: Conf. Ser. 396 052006 DOI: 10.1088/1742-6596/396/5/052006 https://iopscience.iop.org/article/10.1088/1742-6596/396/5/052006/meta

[5] Manzi A, Furano F, Keeble O, Bitzes G *DPM evolution: a disk operations management engine for DPM* October 2017 J. Phys.: Conf. Ser. 898(6) 062011, DOI: 10.1088/1742-6596/898/6/062011

[6] Heinlein P *FastCGI* November 1998 Linux J. 1998, 55es, Article 1

[7] *Storage Resource Management (SRM) Working Group* https://sdm.lbl.gov/srm-wg/

[8] Kalmady R, Tierney B *2001 A comparison of GSIFTP and RFIO on a WAN* Proceedings of CHEP'01, September 3-7, Beijing, China,

[9] Baud J-P et al. *SHIFT, the Scalable Heterogeneous Integrated Facility* Proc. of the Int. Conf. on CHEP'91, Univ. Acad. Press, Tokyo 571-82

[10] *Davix documentation* https://dmc-docs.web.cern.ch/dmc-docs/davix.html

[11] Aiftimiei, C. (2013, May 28). *EMI YAIM CORE V. 5.1.1.* Zenodo. http://doi.org/10.5281/zenodo.6824

[12] Plummer, Shawn and Warden, David, *Puppet: Introduction, Implementation, and Inevitable Refactoring* Proceedings of the 2016 ACM on SIGUCCS Annual Conference, SIGUCCS '16, ISBN: 978-1-4503-4095-3, DOI: 10.1145/2974927.2974950 http://doi.acm.org/10.1145/2974927.2974950

[13] *PuppetForge* https://forge.puppet.com/

[14] Manzi A, Sartirana A (2018, September 4). *cern-it-sdc-id/puppet-dpm: puppet-dpm v0.6.1 (Version v0.6.1)* Zenodo. http://doi.org/10.5281/zenodo.1408556