DPM Evolution: a Disk Operations Management Engine for DPM

Andrea Manzi, Fabrizio Furano, Oliver Keeble, Georgios Bitzes
CERN IT

E-mail: amanzi@cern.ch, furano@cern.ch, okeeble@cern.ch, georgios.bitzes@cern.ch

Abstract.
The DPM (Disk Pool Manager) project is the most widely deployed solution for storage of large data repositories on Grid sites, and is completing the most important upgrade in its history, with the aim of bringing important new features, performance and easier long term maintainability. Work has been done to make the so-called "legacy stack" optional, and substitute it with an advanced implementation that is based on the fastCGI and RESTful technologies. Beside the obvious gain in making optional several legacy components that are difficult to maintain, this step brings important features together with performance enhancements. Among the most important features we can cite the simplification of the configuration, the possibility of working in a totally SRM-free mode, the implementation of quotas, free/used space on directories, and the implementation of volatile pools that can pull files from external sources, which can be used to deploy data caches. Moreover, the communication with the new core, called DOME (Disk Operations Management Engine) now happens through secure HTTPS channels through an extensively documented, industry-compliant protocol. For this leap, referred to with the codename "DPM Evolution", the help of the DPM collaboration has been very important in the beta testing phases, and here we report about the technical choices and the first site experiences.

1. Introduction

The Disk Pool Manager (DPM) is a lightweight solution for grid enabled disk storage management. Operated at around 150 sites, it has the widest distribution of all grid storage solutions in the WLCG infrastructure. It provides an easy way to manage and configure disk pools, and exposes multiple interfaces for data access (xrootd, GridFTP and HTTP/WebDAV) and control (SRM).

The development direction of the last 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. This effort is also dictated by the difficulty of maintaining software libraries that have been written in the 80s and 90s, over which the oldest core components of DPM are based.

Work had been done in the last years to create the *dmlite* [?] framework and a set of plugins that started complementing the core features of DPM by giving support for the more recent data access protocols used by HEP (*HTTP with multi-range requests* and *xrootd*), while keeping the older core components as an internal coordination layer that includes support for the SRM protocols.

The development cycle that ended in Q4/2017 has successfully removed this functional dependency, making the usage of the older cmoponents optional, and linked to the foreseen phase-out of the usage of the SRM protocol in the Grid. This was accomplished by writing a brand new core component that acts as coordination layer and can coexist with the older one in various ways.

2. Dome

DOME (Disk operations Management Engine) is a robust, high performance server that manages the operations of a DPM cluster. DOME is built on the *FastCGI* [?] technology, and uses HTTP and JSON to communicate with clients. The adoption of DOME aims at augmenting the Disk Pool Manager (DPM) system so that its core coordination functions and inter-cluster communication paths are implemented through open components, and following contemporary development approaches headed to performance, scalability and maintainability. We can summarize the main goals of DOME as:

- Making optional all the so-called legacy components that are provided by the *lcg-dm* code tree, namely *libshift*, *rfiod*, *dpm(daemon)*, *dpnsdaemon*, *CSec* and others.
- provide a software infrastructure where adding new coordination features is easier than with the historical one
- provide full support for asynchronous calculation of file checksums of multiple types
- provide support for checking the consistence of replicas through their checksums
- provide structure, hooks and callouts that allow the usage of DPM disk pools as large *file caches*
- having a unified configuration file that is readable and synthetic, as opposed to the previous approach of having several sparse configuration files, all with differently over-simplified syntax rules (or no syntax at all, e.g. */etc/NSCONFIG*)

The DOME component has the shape of a *fastCGI* daemon, and has to be triggered by the Apache instances running in the DPM head node and in all the DPM disk servers. A configuration option defines whether it is running as head node or disk server.

Architecurally speaking, DOME is primarily a service provider for the *dmlite* framework, through a new dmlite plugin called *DOMEAdapter*.

3. DOME: Main features

DOME has two modalities: headnode and disknode, which respectively represent simplified evolutions of the *dpm* daemon and of *rfiod*, together with *libshift* and *Csec*. The functionalities are roughly as follows:

- headnode: general coordination function
  spreads load (PUT, GET, checksums) towards the available disk nodes
  keeps an in memory status of the DPM disk/pool topology with disk sizes and free space
  keeps an in memory status of the ongoing asynchronous checksum calculations
  keeps an in memory status of the ongoing asynchronous file callouts
  queues and dispatches to disk nodes the requests for asynchronous checksum calculations that have to be delayed for load balancing reasons
  queues and dispatches to disk nodes the requests for asynchronous file callouts that have to be delayed for load balancing reasons
- disknode: local disk and space-related services
  Allows to *stat* individual physical files and directories
  Allows to *statfs* filesystems to get used and free space
  Allows the local submission of checksum calculations
  Allows the local submission of file callouts

The historical data tunnelling features provided by the *rfio* infrastructure (and used by gridftp in some boundary scenarios) [?] is implemented by DOMEAdapter directly on the top of HTTPS, hence namely it does not use the DOME server.

The main difference from the legacy components is that DOME does not apply user authorization again for individual internal transactions, as the task of authenticating/authorizing remote users is already accomplished by the dmlite frontends. DOME instead checks that the sender of a request (e.g. a disk server) is authorized to send requests. DOME applies strong, industry standard authentication protocols to this task.
Authentication in DOME is zero-config for the regular cases (one head node and multiple disk servers), and flexible enough to add arbitrary identities that will be allowed to send commands to it.
The activity of DOME is not linked to any particular data access protocol. Its concepts of logical file name and physical file name are not linked to a particular data/metadata transfer protocol.

3.1. 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 [?].
Semantically, SRM space reservations are named reservations of a part of the space of a disk pool. When writing data inworkflows that involve SRM, requests to write a replica specify a pool that has to host the replica, hence ultimately the replica will be subject to the space reservations.

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 *subdirectorybased 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.

A *quotatoken* attached to a directory subtree **overrides others that may be attached to its parents**.
If a directory content (counting all the replicas) exceeds the quota spacified by the quotatoken that influences it, then new PUT requests on that directory will be denied.

As a summary, the meaning of a quotatoken specifying a quota of N terabytes on poolX, associated to directory "/dir1" is *give poolX as space for hosting the files that will be written into dir1. Do not allow more than N terabytes to be hosted there.*

3.2. Open checksumming

DOME supports requests for checksums of arbitrary kind. It can:

- return the corresponding checksum that is stored in the name space
- choose an appropriate replica of the file and tell to the disk node managing it to calculate its checksum
- force the recalculation of the checksum and store it into the name space

The checksum calculation requests are queued in the head node, in memory. The architecture of the queue is designed to be self-healing in the case the checksum calculations do not end correctly, or some machines are restarted, including the head node itself.

4. Architecture

Figure 1 shows the main components of DOME that are in action in a disk server.
Requests come through Apache already authenticated and referring to the two possible paths that refer either to the filesystems or to the dome command path which starts with `/domedisk`.
Another detail that characterizes a disk server for DOME is the ability to execute 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.

A DOME head node is slightly more complex than a disk server, and its internal structure is visible in Figure 2.
Requests come through Apache already authenticated and referring to the two possible paths

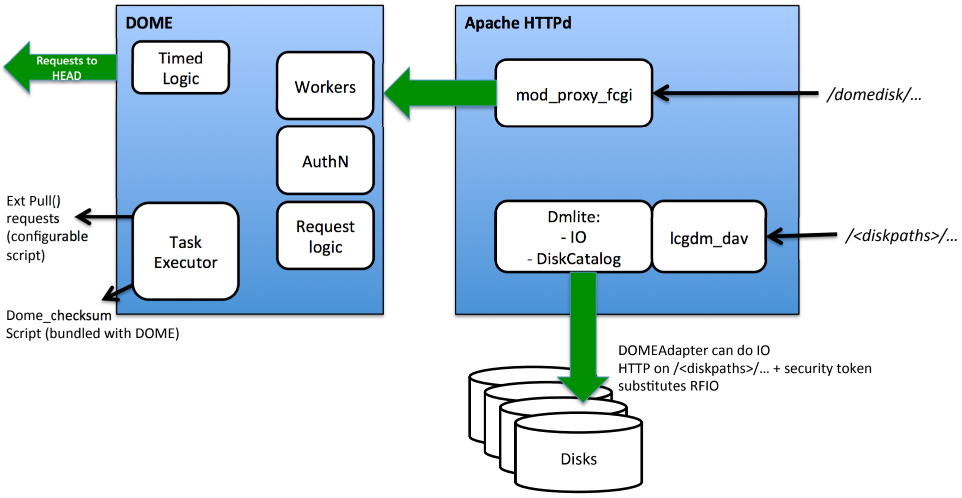
DOME
Requests to HEAD
Timed Logic
Workers
AuthN
Request logic
Task Executor
Ext Pull() requests (configurable script)
Dome_checksum Script (bundled with DOME)
Apache HTTPd
mod_proxy_fcgi
/domedisk/…
Dmlite:
- IO
- DiskCatalog
lcgdm_dav
/<diskpaths>/…
DOMEAdapter can do IO HTTP on /<diskpaths>/… + security token substitutes RFIO
Disks

Figure 1. Simplified diagram of DOME in a disk node

that refer either to the logical name space (`/dpm`) or to the dome command path which starts with `/domehead`. 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.

4.1. Checksum queuer

DOME internally queues and schedules checksum calculation requests in the head node. We can summarize the applied behaviour as:

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

Checksum requests are queued in memory and dispatched to suitable disk nodes that become available with respect to the mentioned criteria. The disk nodes instances constantly update the head node about the running checksums, hence the system will self-heal on restarts of the head node.

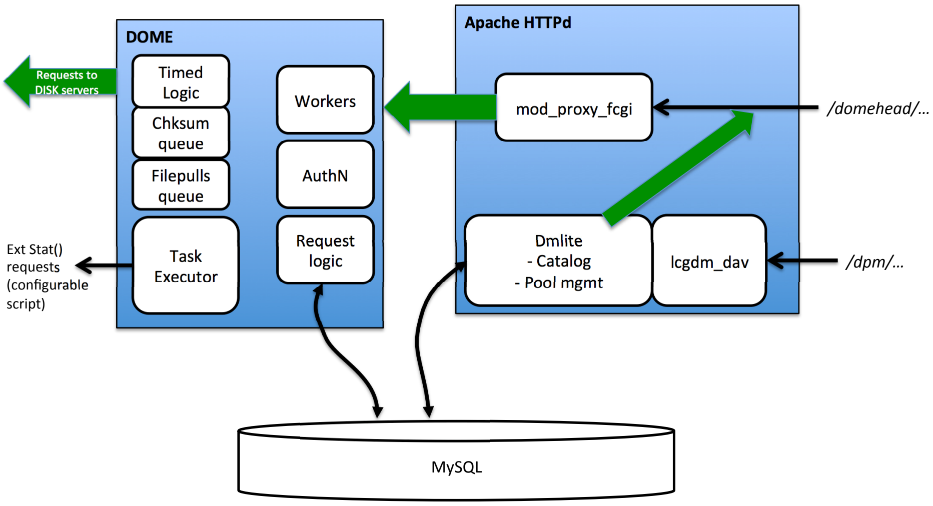

Figure 2. Simplified diagram of DOME in a head node

When finished calculating a checksum, a disk node will notify the head node and pass the result (or failure).

4.2. File pulls queuer

DOME internally on the head node queues and schedules requests for file pulls from external locations. We can summarize the applied behaviour as:

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

The fiel pull itself is implemented as a simple callout in the disk server, that can invoke any file movement mechanism. 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.

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).

5. Conclusion

technical step for the obsolescence of the legacy components has been done
the removal from production will follow the pace at which HEP experiments abandon SRM

blah
blah

References

[1] IOP Publishing is to grateful Mark A Caprio, Center for Theoretical Physics, Yale University, for permission to include the `iopart-num` BibTeXpackage (version 2.0, December 21, 2006) with this documentation. Updates and new releases of `iopart-num` can be found on `www.ctan.org` (CTAN).