Distributed caching system for multi-site DPM storage

Alessandra Doria[1,*], *Gianpaolo Carlino*[1], *Alessandro De Salvo*[2], *Bernardino Spisso*[1], *Elisabetta Vilucchi*[3]

[1]INFN, Napoli Unit, Complesso Universitario M. S. Angelo Ed. 6, Via Cintia, 80126 Napoli, Italy
[2]INFN, Roma1 Unit, Piazzale Aldo Moro 2, 00185 Roma, Italy
[3]INFN, Laboratori Nazionali di Frascati, Via Enrico Fermi 40, 00044 Frascati (Roma), Italy

Abstract. The main goal of this work, in the context of WLCG, is to test a storage setup where the storage areas are geographically distributed and the system provides some pools behaving as data caches. Users can find data needed for their analysis in a local cache and process them locally. We first demonstrate that the distributed setup for a DPM storage is almost transparent to the users, in terms of performance and functionalities. Then, we implement a mechanism to fill the storage cache with data registered in Rucio Data Management system and we test it, running a physics analysis that gets its input data from the cache. Thus we demonstrate that the use of such a system can be useful for diskless sites with a local cache only, allowing to optimize the distribution and analysis of experimental data.

1 Introduction

The experience, gained in several years of distributed computing for large scientific communities, has shown that the Worldwide LHC Computing Grid – WLCG [1] – distributed storage infrastructure is very performing for the needs of the LHC experiments. However, it has been noted that an excessive number of storage sites leads to inefficiencies in the system administration, due to the need of experienced manpower in each site and to the increased burden on the central operations. On the other hand, user analysis is often based on clusters hosted in small sites such as Tier3s: therefore it is important to provide dynamic and efficient data access in such sites as well.

The LHC experiments, WLCG and the Funding Agencies have started a process of optimization of the storage resources and human resources needed for storage operations. Some keywords for this ongoing process are:

- Distributed storage
- Common namespaces
- Data Caching
- Redundancy
- Different Quality of Service

[*] Corresponding author: alessandra.doria@na.infn.it

In this work, we address the first three items of this list, with a prototype system based on the latest releases of the Disk Pool Manager (DPM [2]).

This study proposes a configuration where a primary site represents a single namespace and entry point for the whole storage system, including disk areas located at remote sites. Some disk pools, configured as volatile at different sites, work as local data caches with zoning access mechanisms. The proposed storage configuration has also been tested in the context of the ATLAS experiment and tested with a real user analysis job getting its input data from the cache.

2 DPM - Disk Pool Manager

The DPM storage system is used since 2006 in three out of the four Italian Tier2s of the ATLAS experiment. It offers a simple way to create a disk-based grid storage element composed by many disk servers. DPM supports the data and metadata access protocols that are relevant for file management and access in a distributed environment (HTTP, WebDAV, xrootd, SRM, gsiftp).

The system is composed of one Head Node, that hosts the metadata and manages the requests, and several Disk Nodes hosting data files and managing the file transfers with the mentioned protocols, as shown in Fig. 1. Traditionally head and disk nodes are located at the same site; leveraging the fast and reliable connections among different sites, we implemented and tested a prototype where the head node manages disk nodes located at three remote sites.

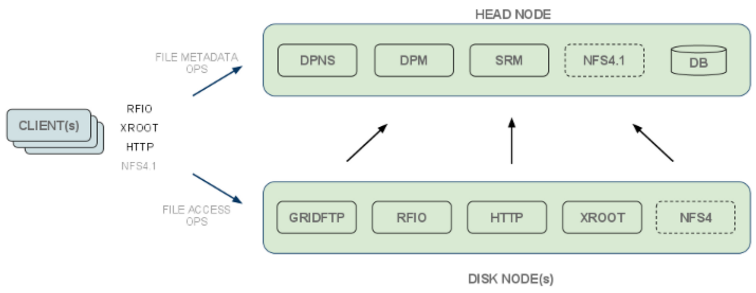

Fig. 1. DPM architecture

The latest DPM release, that relies on a new core called DOME (Disk Operations Management Engine [3]), has been used to verify how it fulfils some of the optimization requirements reported in the introduction, making it compatible with the future evolutions of models for data access and management. DPM/DOME allows to run the storage system without the SRM protocol and overrides the previous definition of SRM Space Tokens, with the introduction of Quota Tokens, associating a defined quota to a pool and to a specified path in the namespace.

The DPM/DOME system offers the possibility to implement both permanent and volatile storage pools. Volatile pools work as file caches and a custom filling mechanism can be implemented for each volatile pool. Any external storage system can be set as file source for a volatile pool with a simple customization, depending on the desired behaviour. See section 3.2 for details about the cache filling mechanism chosen for this work.

3 The configuration of the testbed system

3.1 The distributed storage configuration

We deployed a testbed setup, distributed over three Italian INFN sites, namely INFN-NAPOLI, INFN-ROMA1 and INFN-FRASCATI (NA, RM1 and LNF in the figures) that are Tier2 sites in WLCG and use DPM for their Grid Storage Elements in production.

The testbed has its DPM head node located in Naples: it is the only system front-end and manages three disk nodes located in Naples, in Rome and in Frascati. A common permanent pool and three volatile pools are defined, as shown in Fig. 2. The permanent pool is distributed and includes storage areas at each disk node, while the volatile pools, one per site, can be used as local cache at each site.

All the nodes (head and disks) of the system have a 10Gbps link to the WAN.

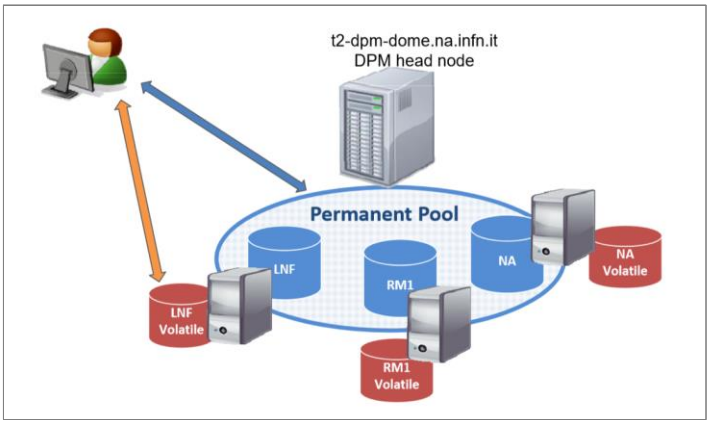

Fig. 2. Distributed DPM setup

A file of the permanent pool can physically reside in any of the disk nodes. This is completely transparent to the user and the Logical File Name for a permanent file refers to a generic path, for example:

t2-dpm-dome.na.infn.it:/dpm/infn.it/home/atlas /fileA

To get a file from one of the volatile pools instead, we have to explicitly point to the corresponding path in the namespace to be able to choose the local cache. So the paths defined for the three caches are:

t2-dpm-dome.na.infn.it:/dpm/na.infn.it/home/atlas /fileB
t2-dpm-dome.na.infn.it:/dpm/roma1.infn.it/home/atlas /fileC
t2-dpm-dome.na.infn.it:/dpm/lnf.infn.it/home/atlas/ fileD

The domain contained in the path (i.e. */dpm/lnf.infn.it/*) is just a conventional name used to make explicit the location where the cache resides. The association between volatile pools and corresponding storage paths is made in DOME by the definition of Quota Tokens, that are also defining the space quota reserved to the associated path.

The storage system made of distributed storage areas allows to use heterogeneous storage technologies for the different disk nodes. In our case, for example, while Naples and Frascati sites provide standard posix file-systems, Rome storage areas relies on a *CEPH rbd* device in replica 3 configuration, hosted on a shared facility with other grid elements.

We will show that no significant difference in performance comes from the configuration where disk nodes and head node are in different domains, given a fast and reliable network connection among the involved sites.
Another advantage of such a configuration is that system management, configuration and installation are centralized with a single Foreman/Puppet instance in Naples.

3.2 The volatile pools as file caches

In our testbed setup, one volatile pool per site is defined, in order to be able to make different tests across the sites.

In a realistic production environment, the volatile pools could be local caches at small sites like Tier3, that have computing resources to offer to local users for their analysis, but want to get rid of the burden of managing a complete storage system. The head node and the permanent pools would reside instead at the larger and better connected sites, like Tier2, whose system administrators could manage remotely the volatiles pools also. In this setup the caches would be transparent for the central operation of the experiment.

The volatile pool is filled on request with a custom caching mechanism, that can be implemented according to user needs. The first time a file is requested to a volatile pool, the corresponding disk server retrieves it with a customized script, serves the file to the requiring client and keeps it locally for the next accesses.

Complex retrieval algorithms can be implemented and the file source can be any other storage system or any kind of Data Federation as Dynafed (see for example [4]).
In our setup the cache interacts with Rucio Data Management ([5]) to get the requested file from any ATLAS site, but the ATLAS Data Management system is not aware of the existence of the local caches, so they can be added or removed according to the local needs.

The cache simply acts as a Rucio client to download files. When the cache is not yet populated, its behaviour (implemented via DOME) is driven by two scripts that are triggered by the file requests toward a volatile pool. Such requests can be *stat* (retrieve the metadata) or *get* (download or access the file):

- the *stat* call triggers a script on the head node only, checking if the file exists in the data source and retrieving the size and the metadata as well.
- the *get* request triggers a pull script running on the disk node corresponding to the chosen volatile pool. The script can address any system as file source and can use any implemented protocol for the file transfer.

While the transfer of the file from the chosen source to the volatile pool is in progress, DOME returns to the client the *accepted* status code (202 for the dav/davs protocol), asking to wait until the completion of the transfer. When the file is ready in the volatile pool, DOME returns the *succeeded* status code (200 for the dav/davs) and the transfer to the requesting client is started.

Most of the file transfer clients and libraries, used in grid environment, support the 202 status code, so the client is able to wait until the file is copied to the cache. For example, in our tests we have used the gfal client, where we can set the retry-delay and the max retry attempts in case of 202 status code.

In case the volatile pool is already populated and a cached file is requested, no script is called. The users can download or access the files directly, in the same way as for the permanent pool.

For this work we have chosen to fill the cache using as data source all the ATLAS files registered in Rucio. We implemented and tested a retrieving script that runs a Rucio client: when the script is triggered, Rucio is initialized and a download operation is started to the local physical destination directory (mapped to the volatile pool) specified by DOME.

The scope and the file name are supplied by the users through the Logical File Name used for the get request and the virtual directory named after the Rucio scope is created and populated with the required file. For example, to get from the cache the file

mc15_13TeV:DAOD_EXOT5.1053164.pool.root.1

the following request has to be issued to the volatile pool:

gfal-copy davs://t2-dpm-dome.na.infn.it/ dpm/na.infn.it/home/atlas/mc15_13TeV/DAOD_EXOT5.10531640.pool.root.1 ./

4 System Tests and results

The system has been tested in different ways, in order to verify the behaviour under different circumstances. The tests shown in the following are mainly focused on verifying functionalities rather than on measuring performance, as in real installations performance may widely vary depending on network connections, hardware performance, system load.

The following issues have been addressed:

- Verify that having disk servers distributed over different network domains does not affect the system functionality;
- Experience the use of volatile pools as caches;
- Try to use cached data as input for a real analysis job, to show the validity of the approach in terms of feasibility and execution times.

4.1 Testing the Distributed setup

As shown in section 3.1, we configured a distributed DPM pool where the storage permanent data reside, including storage areas at the three sites. The file transfers with the distributed permanent pool were performed with the three most used protocols in Grid environment: davs, xrootd and gsiftp, to compare the result in the different cases.

For each protocol and with files of different sizes, we performed writing and reading of a single file and writing of 10 files in parallel threads.

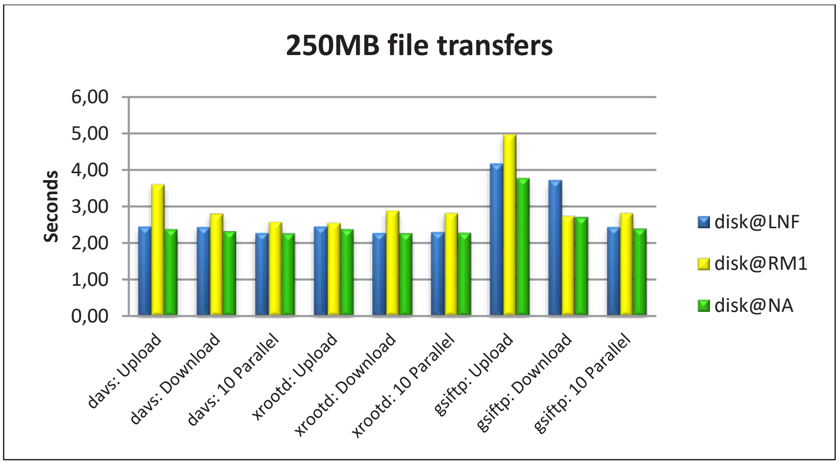

Fig. 3. Time to upload and download a 250MB file from the three disk servers with xrootd, davs and gsiftp protocols (average values over 10 measurements, statistical uncertainty ~2%)

The different locations of head and disk nodes and the different underlying file-systems (CEPH@ RM1, posix@ NA and LNF) might add an overhead in transfer times. In order to point it out, the tests were repeated with different setups: to force test files to be written at a specific site, only one site disk at a time is left as writable. Fig. 3 shows the times, in seconds, spent on writing (upload), reading (download) and parallel writing of 10 files, for a 250 MB file with the three chosen protocols.

To eliminate the effect of the file transfer over WAN, the user interfaces from which the tests were carried out were, from time to time, at the same site of the considered storage areas. We show the results at the different sites with different colours.

The comparison of the results at Naples and Frascati shows that, except for the gsiftp protocol, the overhead introduced by the remote disk server with respect to the head node, is negligible. While there is a difference in Rome where the file system relies over CEPH and the user interfaces have a different configuration.

Measurements carried out with all writable storage areas (the file is written randomly to one of the disks) and different file sizes bring to compatible results.

4.2 Testing the Cache

Cache effectiveness strongly depends on how it is used: for example, on how many times a file is reused, on how long the files are needed and on the possibility of pre-filling the cache with popular data.

In our system pre-filling is not possible, as caches are filled only on client request. The reuse and durability of data strongly depends on the type of analysis and on workflow of the local site users and groups.

Each cache area is accessible with a specific path in the namespace, that conventionally contains the domain of the site where the cache physically resides and that must be known to local users. For example, we can access a file at the Rome cache with:
davs://t2-dpm-dome.na.infn.it/dpm/roma1.infn.it/home/atlas/user.angianni/ user.angianni.14404934._000001.CxAOD.root

At the first file retrieval, the "cold" cache contacts the ATLAS Data Management system Rucio to get a file replica from the grid and make the file available to the requester. At any subsequent file access, we define the cache as "warm" since the requested file has already been downloaded from remote and it is immediately available.

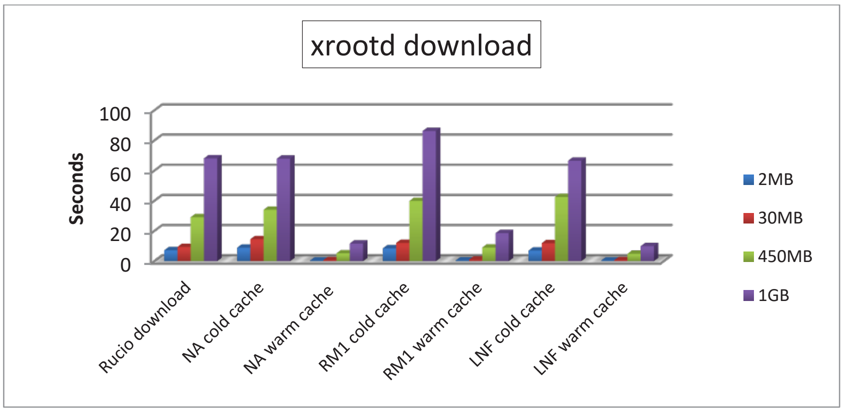

Fig. 4. Transfer time for files of different sizes (average values, statistical uncertainty ~5%)

In our tests, at each of the three involved sites, we requested files of different sizes (2MB, 30MB, 450MB e 1GB) from the local caches in both cases of "cold" and "warm" cache. Each of the measurement has been repeated 10 times as results may vary on the basis of network traffic and load of the data servers. The tests have been repeated with different protocols (davs, xrootd, gsiftp), but the results are not significantly different, so only results for xrootd protocol are shown in Fig. 4.

Comparing the transfer time of a plain "*rucio download*" command, that retrieves the file directly from the grid, with the retrieval time from the cold cache we note that time is very similar, while the retrieval time from warm cache is obviously much lower. We see again that the overhead introduced by remote disk servers is negligible, as the results using Naples or Frascati cache are very close, while there is a small increase in cold cache time at Rome due to writing time on CEPH.

4.3 File caching in a real use case

To verify the behaviour of the proposed system in a real environment, we tested it with an actual ATLAS job reading its input files from the cache with xrootd direct access.
As test job, we used a program that applies calibrations to reconstructed objects, makes an event pre-selection and saves in a TTree output a minimal set of information needed to perform offline interactive analysis.

The workflow adopted by the user to run his job was to download the input files to the local file system and then to run his job on local computing resources. The only change needed to adapt the job to run on cached files was to modify the path of the input files: from the path of the local file system to the transport URL (protocol, server, path) of the file in the local cache. As explained in section 3.1, a conventional rule has been established to determine for each site the path to access the local cache. Moreover, a very simple convention has been adopted to specify in the path all the information (scope and filename) needed to retrieve an ATLAS file in Rucio. These simple conventions have to be known by the users of the local cache. The size of the chosen input file is 1.8 GB and it contains 55k events, but we chose to run the program on a subset of 100 events only, to be able to appreciate small time fluctuations.

Each measurement was repeated 10 times and the statistical uncertainty is of about 2%. In Fig. 5 we show time needed to run the job with different configurations of the input file:

- First access: the "cold" cache gets the file from grid before the job can open it.
- Cache access when the file is already cached (warm cache).
- Old workflow: the user first downloads the file to the local file-system then runs the job on the local input.

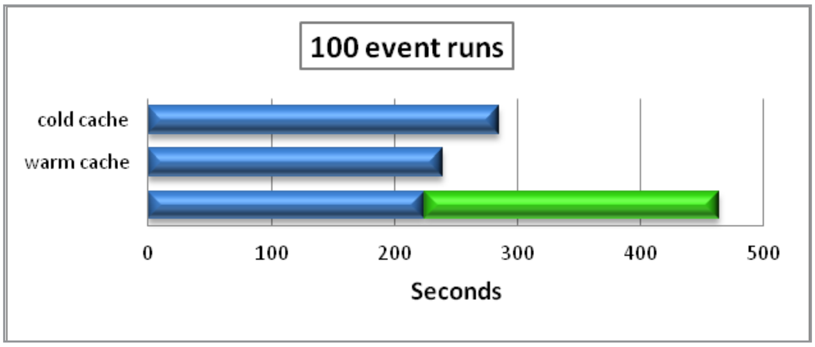

Fig. 5. Average time to run a job over 100 events in different configurations: input file downloaded locally, input file in warm cache, input file in cold cache.

Advantages of this approach:

- The user doesn't need to go though the double step of downloading the input file in a local area and running the job when the download is completed.

- Cached files can be accessed by any user and computing resource at the site.
- The time spent on running the job, even when the cache is cold, is generally lower than time needed to download the file to a local file-system plus running the job.

Drawbacks:

- Cache retrieving algorithm is not yet customized to handle full datasets. It can be done, at the cost of some latency added in download.
- No pinning mechanism keeping in the cache the most accessed files is so far available and oldest data are automatically removed from the cache when it is full. User have no control on file lifetime.

5 Conclusions

The feasibility of the proposed architecture and its functionality have been proven. The results of the transfer tests show that different storage hw (CEPH, Posix file systems) and geographical distributed storage areas don't affect significantly the data access and transfer. Any ATLAS file can be automatically retrieved in the cache: this could considerably improve local analysis on frequently used datasets and it's achieved without involving ATLAS Data Management. This system well suitable for small sites, that don't host permanent data (diskless sites) and could be unstable or poorly connected.

The user analysis performs very well reading its input data from the local cache. The user running on a machine or cluster out of the grid can thus skip the phase of downloading the input datasets on local disks.

We plan to extend and improve this study:
• Extending to larger infrastructures and to more heterogeneous storage media;
• Performing stress tests in a production-like environment.

We want to thank Francesco Cirotto, Antonio Giannini and Valerio Ippolito for providing their analysis jobs for the tests and for their help in understanding user's needs, and the DPM team for supporting us in the deployment of the new DOME features.

References

1. I. Bird et al, Update of the Computing Models of the WLCG and the LHC Experiments, CERN-LHCC-2014-014, (2014); LCG-TDR-002 http://cds.cern.ch/record/1695401
2. M. Hellmich et al, DPM efficient storage in diverse environments, 2014 J. Phys.: Conf. Ser. 513 042025; http://inspirehep.net/record/1302100/files/jpconf14_513_042025.pdf
3. A Manzi et al, DOME DPM evolution: a disk operations management engine for DPM, 2017 J. Phys.: Conf. Ser. 898 062011; http://iopscience.iop.org/article/10.1088/1742-6596/898/6/062011/meta
4. D. Michelino, S. Pardi, G. Russo, B. Spisso, An http data-federation eco-system with caching functionality using DPM and Dynafed, CHEP 2018 proceedings (to be published)
5. M.S. Barisits, T. Beermann, V. Garonne, T. Javurek, M. Lassnig, C. Serfon, The ATLAS Data Management System Rucio: Supporting LHC Run-2 and beyond, ATL-SOFT-PROC-2017-064, (2017); http://cds.cern.ch/record/2291104/files/ATL-SOFT-PROC-2017-064.pdf