EOS software evolution enabling LHC Run 3

Cedric Caffy[1,*], *Guilherme* Amadio[1], *Jaroslav* Guenther[1], *Abhishek* Lekshmanan[1], *Luca* Mascetti[1], *Andreas* Peters[1], *Manuel* Reis[1], *Michal Kamil* Simon[1], *Elvin* Sindrilaru[1], and *David* Smith[1]

[1]CERN

Abstract.
EOS has been the main storage system at CERN for more than a decade, continuously improving in order to meet the ever evolving requirements of the LHC experiments and the whole physics user community. In order to satisfy the demands of LHC Run-3, in terms of storage performance and tradeoff between cost and capacity, EOS was enhanced with a set of new functionalities and features that we will detail in this paper.
First of all, we describe the use of erasure coded layouts in a large-scale deployment which enables an efficient use of available storage capacity, while at the same time providing end-users with better throughput when accessing their data. This new operating model implies more coupling between the machines in a cluster, which in turn leads to the next set of EOS improvements that we discuss, targeting I/O traffic shaping, better I/O scheduling policies and tagged traffic prioritization. Increasing the size of the EOS clusters to cope with experiment demands, means stringent constraints on the data integrity and durability that we addressed by a re-designed consistency check engine. Another focus area of EOS development was to minimize the operational load by making the internal operational procedures (draining, balancing or conversions) more robust and efficient, to allow managing easily multiple clusters and avoid possible scaling issues.
All these improvements available in the EOS 5 release series, are coupled with the new XRootD 5 framework which brings additional security features like TLS support and optimizations for large data transfers like page read and page write functionalities. Last but not least, the area of authentication/authorization methods has seen important developments by adding support for different types of bearer tokens that we will describe along with EOS specific token extensions. We conclude by highlighting potential areas of the EOS architecture that might require further developments or re-design in order to cope with the ever-increasing demands of our end-users.

1 Introduction

EOS is a distributed disk storage system providing services for storing large amounts of physics data and user files. Not only EOS is used to store physics data at CERN, but it is also used as a storage backend for CERNBox [1] - that provides cloud access storage to different

*e-mail: cedric.caffy@cern.ch

user communities - and it is used by CTA [2] - CERN Tape Archive - to store physics data on tapes.

EOS plays a major role in the LHC and non-LHC experiments physics data workflows. With all EOS instances at CERN considered, around 8 billion files are stored and more than 790 PB of raw storage capacity is deployed using 1300 storage nodes and 60000 disks.

In order to meet LHC Run 3 requirements, EOS was enhanced with several new functionalities allowing to run efficiently on the hardware deployed at CERN. After having quickly introduced the EOS architecture, we will present the use of erasure coding allowing to optimize the use of the storage space compared to normal file replication. Next, we will present the new Input/Output (I/O) optimizations put in place to mitigate the effects of erasure coding on the cluster. With the ever-increasing number of files stored on EOS, it was necessary to design a data integrity and durability tool that we will present in a fourth part. Finally, with the increasing popularity of the usage of tokens to perform data transfers, we will present the different tokens that EOS supports. We will conclude by presenting future areas of development of the different components of EOS that will be done in order cope with the future physics community needs.

2 The EOS architecture

The EOS architecture is composed of three main components. The MGM daemon is the initial point of contact for external clients, responsible for the authentication and authorization of the user, dealing with file metadata operations (registration, modification...) and client-redirection to the physical location of their file. The FST is a server responsible for managing the physical storage of the files. It is very often attached to several dozens of disks. The namespace, is responsible for the persistency of the metadata of every files stored in EOS. Finally, a MessageQueue (MQ) daemon ensures the communication between the different FSTs and the MGM.

The figure 1 presents the EOS architecture by showing the different components that interacts with each other when a user writes a file.

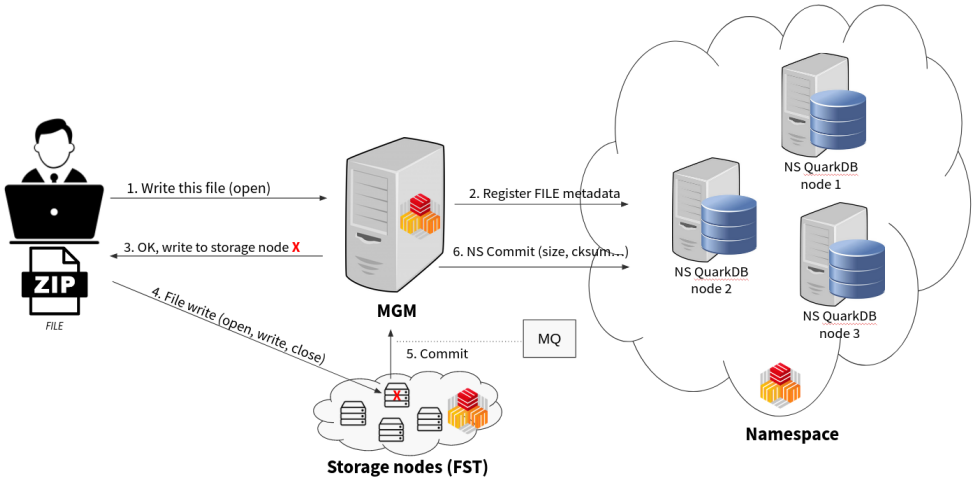

Figure 1. The EOS Architecture - a client that copies a file to/from EOS will be in contact with at least two machines

3 Store more data with less hardware - the use of erasure coding

Currently at CERN, files availability is ensured by creating and storing two replicas for each file in EOS. It is done by using the RAIN (Redundant Array of Inexpensive Nodes) layout that ensures that the two copies of each file will never land on the same FST.

This solution works well however it is expensive in terms of storage because one needs twice as much physical storage as logically needed to store the file, similar to a RAID-1 system.

This problem can be solved by using a new storage layout implemented in EOS, based on erasure coding (EC) [3]. Erasure coding consists of splitting the data blocks into different physical locations and add parity blocks to protect it. This process is executed by a FST that keeps the first stripe and sends the others to different locations. This FST is called the I/O gateway. For data read, two access pattern can be used:

- I/O gateway
- Client-side EC

The I/O gateway pattern works the same way as the I/O gateway for writing. One FST acting as the I/O gateway, it will read the stripes from the other FSTs to then reconstruct the data before sending it to the user. On the other hand, the client-side EC pattern consists of asynchronously reading all the stripes from the different FSTs and locally reconstruct the data. The tool *eoscp* [4] is used for that pattern.

The figure 2 illustrates the different access pattern for EOS EC file writing and reading.

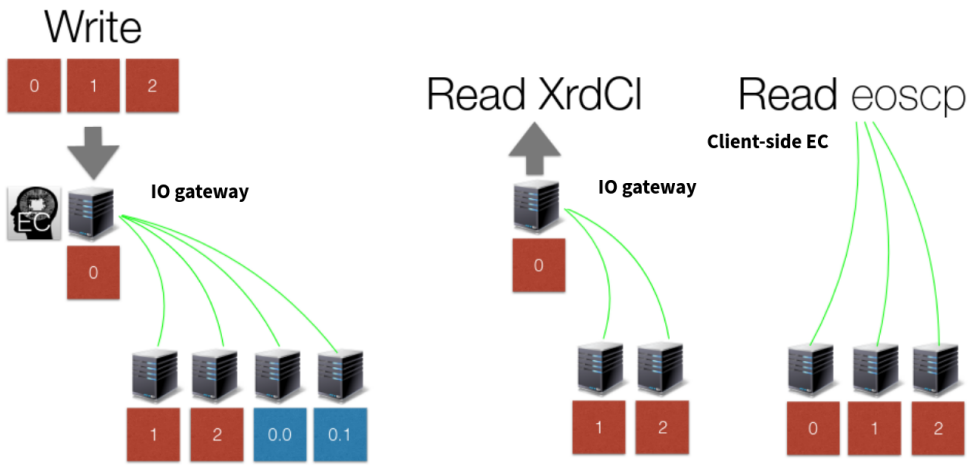

Figure 2. The different access pattern for EC file writing and reading - This model is EC(3,2) and allows to reconstruct the data even if 2 servers are not running

In terms of performance, EC has two times traffic amplification for read using the I/O gateway model compared to replication. But in general, clients benefit from the performance coming from the parallel transfers of the different disks involved in EC.

The data coming from the ALICE experiment at CERN is 100% stored using erasure coding 10+2 allowing to save around 55 PB of raw storage capacity compared to replication.

The latest benchmark performed against this EOS instance allowed to reach almost 720GB/s of data read throughput by using 6060 streams reading from 12000 disks.

4 Improving I/O performance

Erasure coding implies more coupling between the machines involved in the cluster. We will present three I/O optimization implemented in EOS in order to mitigate the impact: I/O type [5], I/O priority [5] and bandwidth regulation [5].

4.1 I/O Types

EOS can be configured to select, per data transfer, the I/O type that will be used to copy the file to/from the disk. Two I/O types can be choosen:

- Buffered I/O
- Direct I/O

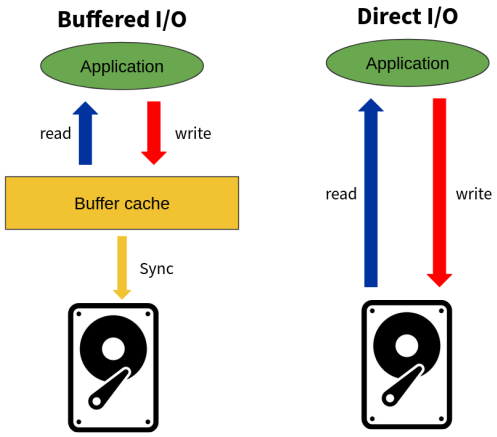

Figure 3. Buffered I/O vs Direct I/O

A file transfer using buffered I/O passes through the kernel's page cache, allowing to get very good performances in the case the same file is accessed multiple times. Clients may benefit from read-ahead - allowing to anticipate file read - and write-back - allowing to flush the file from the cache on close. The performance of this I/O type may be bad if many files are competing for the cache.

A file transfer using direct I/O, on the contrary, does not make use of the kernel's buffer cache. It reduces CPU and memory usage and the performance will be limited to that of the disk. Direct I/O demonstrated very good performance for file writes, it reduced performance tails and allowed to increase the maximum performance of a standalone XRootD [6] disk server from 7GB/s to 9GB/s.

4.2 I/O priorities

Depending on the use-cases of data transfers, we may need to balance the needs for high throughput by fairly sharing disk I/O requests among processes. For example, maintenance tasks can have lower priority than experiments data transfers.

EOS offers the possibility to set, per data transfer, the priority of the disk I/O transfer. It works only with direct I/O read and writes on devices using BFQ and CFQ scheduler. The nomenclature from the highest priority to the lowest priority is presented in figure 4.

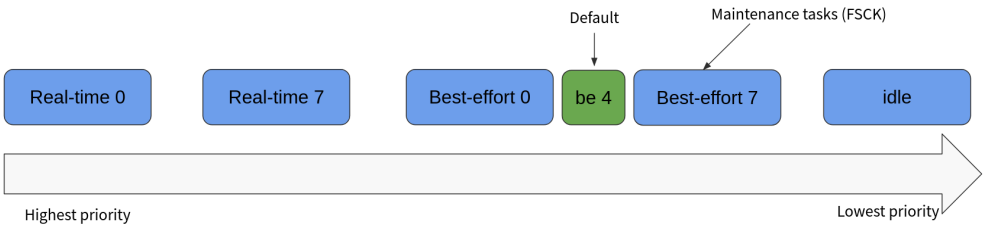

Figure 4. Nomenclature for I/O priority setting from the highest priority to the lowest

At CERN, the data coming from the experiments Data AcQuisition (DAQ) systems have a higher priority compared to other data transfers.

4.3 Bandwidth regulation

Benchmarks did show that I/O performance tails can be reduced by limiting the single transfer bandwidth of the clients. Indeed, limiting the bandwidth per transfer allow clients streams to not compete for the maximum throughput they can get from the EOS cluster. Very fast I/O being limited, other I/O can fairly share the bandwidth. The figure 5 illustrates the effect of bandwidth limitation applied to 13000 parallel transfers. The bandwidth regulation is extremely useful for DAQ systems where the client application need to ensure that each data streams finish in a certain amount of time.

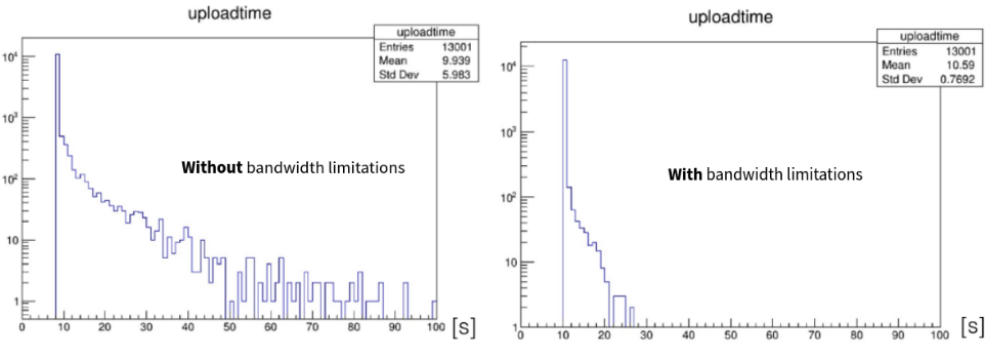

Figure 5. The effect of applying bandwidth regulation to 13000 parallel transfers

5 Improving files and data reliability

With the several billions of files currently stored in each EOS instance at CERN, it is very important to ensure the good integrity of the data.

EOS provides its own FSCK (File System CheckK) [7] functionality. FSCK is running on the MGM and all the FSTs. The main responsibilities of FSCK are:

- Detect corruptions on the disk server
- Protect against "silent" corruptions or hardware issues
- Maintain consistency between the EOS namespace view and the filesystem view

- Maintain required data redundancy
- Avoid data loss due to preventable error scenarios

The collection and the repair of the errors detected is done by the MGM. The scanning of the different disks and the metadata comparison with the namespace is done by the FST.

FSCK successfully got enabled on the EOSLHCb instance in February and the effect was immediate. The figure 6 shows the decrease of files errors that got detected and automatically repaired by FSCK after being enabled.

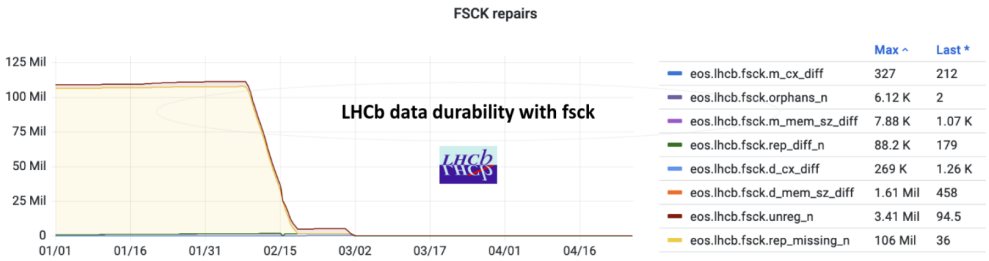

Figure 6. Files error decrease on the LHCb instance after FSCK got enabled

6 EOS authentication and authorization with tokens

Tokens are becoming more and more popular to authenticate with different storage elements. This popularity comes with the increasing usage of the HTTP protocol to perform data transfers between different storage elements.

EOS currently supports three types of tokens that we will present:

- The native token - also called *EOS token*
- Macaroons
- Scitokens [8], a subtype of WLCG JSON Web Token (JWT) [9]

6.1 EOS token

This token is the EOS native token. A client can request such a token by issuing an EOS command towards the MGM:

```
$ eos root://eos.cern.ch// token --path /path/to/file.txt --expires
1681807613 --permission rwx
zteos64:MDAwMDAyMmN4nOP6z8jFXFReIfB348[....]4PQ%3d%3d
```

In return, the client receives a token starting with *zteos64:*.

The permissions associated to this token are the ones that are associated to the resource or the ones set by the client in the request.

6.2 Macaroons

Macaroons are a type of bearer token that are signed using a secret key and that contain a list of *caveats* that are conditions that need to be respected in order to ensure its validity.

In order to request a Macaroon, the client needs to request it using their x509 certificate and issuing an HTTP request towards the MGM and the path of the resource they want to access. For example:

```
$ curl --cert [...] --key [...] --cacert [...] --capath [...] -X POST -H
'Content-Type:application/macaroon-request'
-d '{"caveats":["activity:UPLOAD,DELETE,LIST],"validity":"PT3000M"}'
https://eos.cern.ch/eos/path/to/file.txt
```

The permission associated to this token are the ones that the client has for the resource.

6.3 Scitokens

The EOS authentication relies, among other libraries, on the XRootD SciToken library [10] which was developed to support tokens compatible with the WLCG Common JWT profile. Among these tokens, the Scientific token or *sci-token* has become very popular within the WLCG[1] community. It will be used in a close future as a de facto standard and as a replacement to X509 certificate authentication for data transfers between WLCG sites.

Unlike the two tokens presented previously, the EOS namespace is not involved in the token issuance process. A client must request a sci-token from a IAM[2] provider. The IAM provider (e.g. Indigo IAM [11]) is an external entity that manages identities, group membership and authorization policies on some distributed resources within the WLCG.

A sci-token can be requested by using the *oidc-token* tool provided by the *oidc-agent* RPM package. It is associated to a user and the permissions are defined by the *scopes* that the token contains.

6.4 Token usage with EOS

These three types of token that EOS supports can be used in different ways. For HTTP access, the token has to be passed via the *Authorization* header like the following:

```
$ curl -x GET
-H "Authorization: Bearer $TOKEN" https://eos.cern.ch//path/to/file.txt
```

For XRootD access, the token has to be appended to the end of the path of the resource.

```
$ xrdcp ./file.txt root://eos.cern.ch//path/to/file.txt?authz=$TOKEN
```

7 Conclusions and future work

With the different software evolution that EOS has received, the distributed storage system is delivering fast and reliable storage for the CERN LHC and non-LHC experiments. With the use of erasure coding to optimize raw storage space, several I/O optimizations implemented, the use of FSCK to improve the data reliability and the support for tokens, EOS is ready to cope with the Run 3 needs.

In order to prepare for LHC Run 4, several areas of the EOS software will be worked on. We will start by improving the overall MGM performance by implementing a finer-grained namespace locking strategy [12]. Currently, each call to the EOS namespace done by the MGM is done under a general mutex lock which is not optimal. The idea is to use finer-grained mutexes to only lock the parts of the namespace that the client is working on. This will allow a better parallelism for MGM-namespace interactions.

[1] Worldwide LHC Computing Grid
[2] Identity and Access Management

Another source of improvement that is being discussed is to evolve the API to open files on EOS. Currently, in the case a client requests multiple files for reading, they need to perform one open call per file. We will study the feasibility of having a bulk API for opening all the files in only one call. This would enable a reduction in the number of operations required for each open call to just one. For example, the authentication would be done only once for all the files submitted in the bulk open request.

Finally, with the increase of popularity of the RN Tuple format, it is necessary to evaluate the massive scale for the RN Tuple usage and the impact on I/O performance and bandwidth.

References

[1] CERNBox - https://cernbox.cern.ch/

[2] CTA - https://cta.web.cern.ch/cta/

[3] Andreas-Joachim Peters, Michal Kamil Simon, Elvin Alin Sindrilaru, Erasure Coding for production in the EOS Open Storage system - CHEP 2019 - https://doi.org/10.1051/epjconf/202024504008

[4] eoscp code - https://gitlab.cern.ch/dss/eos/-/blob/master/fst/eoscp.cc

[5] Andreas-Joachim Peters, IO Shaping in EOS https://indico.cern.ch/event/1123214/contributions/4809924/attachments/2432068/4164961/IO%20Shaping%20in%20EOS%20-%20HEPIX.pdf

[6] XRootD - https://xrootd.slac.stanford.edu/

[7] Improving EOS availability through early error detection and recovery - FSCK https://indico.cern.ch/event/862873/contributions/3724435/

[8] Sci-tokens - https://scitokens.org/

[9] WLCG JSON Web Token profile - https://zenodo.org/records/3460258

[10] XRootD Scitokens library - https://github.com/xrootd/xrootd/tree/master/src/XrdSciTokens

[11] Indigo IAM documentation - https://indigo-iam.github.io/v/v1.8.2/docs/overview/

[12] Cedric Caffy - EOS Workshop 2023 - The EOS namespace locking, demystification and optimization https://indico.cern.ch/event/1227241/contributions/5326725/