EPJ Web of Conferences **245**, 04009 (2020) https://doi.org/10.1051/epjconf/202024504009
CHEP 2019

EOS architectural evolution and strategic development directions

Georgios Bitzes[1,*], *Fabio* Luchetti[1,**], *Andrea* Manzi[1,***], *Mihai* Patrascoiu[1,****], *Andreas-Joachim* Peters[1,†], *Michal Kamil* Simon[1,‡], and *Elvin Alin* Sindrilaru[1,§]

[1]CERN, Esplanade des Particules 1, 1217 Meyrin, Geneva, Switzerland

Abstract. EOS [1] is the main storage system at CERN providing hundreds of PB of capacity to both physics experiments and also regular users of the CERN infrastructure. Since its first deployment in 2010, EOS has evolved and adapted to the challenges posed by ever-increasing requirements for storage capacity, user-friendly POSIX-like interactive experience and new paradigms like collaborative applications along with sync and share capabilities.

Overcoming these challenges at various levels of the software stack meant coming up with a new architecture for the namespace subsystem, completely redesigning the EOS FUSE module and adapting the rest of the components like draining, LRU engine, file system consistency check and others, to ensure a stable and predictable performance. In this paper we detail the issues that triggered all these changes along with the software design choices that we made.

In the last part of the paper, we move our focus to the areas that need immediate improvements in order to ensure a seamless experience for the end-user along with increased over-all availability of the service. Some of these changes have far-reaching effects and are aimed at simplifying both the deployment model but more importantly the operational load when dealing with (non/)transient errors in a system managing thousands of disks.

1 Introduction

The EOS Open Storage System was originally targeted for the particular use case of user analysis in LHC experiments. Based on experience from previous storage projects at CERN the usage of a relational database as metadata backend was dropped and replaced by an in-memory namespace. The single file access latency was reduced to ms level. Initially EOS offered only remote access protocols (XRootD, gridFTP). Today a big part of IO is done using a FUSE filesystem interface.

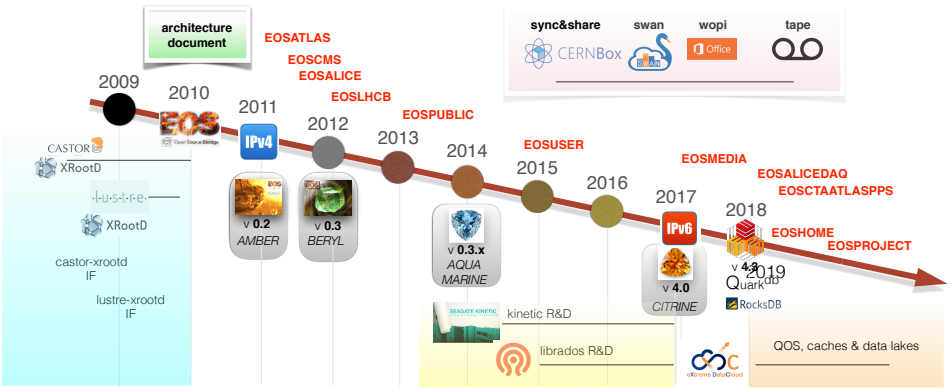
*e-mail: georgios.bitzes@cern.ch
**e-mail: fabio.luchetti@cern.ch
***e-mail: andrea.manzi@cern.ch
****e-mail: mihai.patrascoiu@cern.ch
†e-mail: andreas.joachim.peters@cern.ch
‡e-mail: michal.simon@cern.ch
§e-mail: elvin.alin.sindrilaru@cern.ch

2 Project History

The EOS project started with an architecture document written in April 2010. Since 2011 EOS releases had been named with gemstones. The major releases were called Amber (2011), Beryl (2012), Aquamarine (2014) and Citrine (2017-2019).

Three major changes within the Citrine version were the introduction of IPV6 support in 2017, the change of the namespace architecture in 2018 and the support of the Cern Tape Archive (CTA) in 2019. The number of production instances has seen a linear increase over time. First use cases were grid storage systems for LHC experiments, since 2015 use cases like CERNBox [2] as a sync&share platform had been added. The project has run two major R&D project phases, the first one integrating external scalable storage technologies like Seagate Kinetic[1] and Ceph RADOS[2] from 2014 to 2017. Since 2019 EOS was participating in the Extreme Data Cloud (XDC) project [5] implementing the ideas of Quality Of Service (QOS) and Data Lakes.

[h]



EOS Project Timeline

The project timeline is illustrated in figure 2.

3 Storage Use Cases

EOS is used today in two major areas:

- disk storage
  - raw data storage - large files with sequential IO, write once read many storage (WORM)
  - analysis use cases - files of any size, random or forward seeking access, WORM storage

[1]Kinetic drive technology has been phased out
[2]RADOS - Reliable Autonomic Distributed Object Storage

  - home and project areas for the CERNBox service - file of any size including file updates, software compilation and distribution use cases
  - cloud storage at AARNet [3] and the Joint Research Centre JRC [4] - geo-distributed deployment (60 ms latency between storage nodes) and sync&share use cases
  - Tier2 center and university storage - mainly analysis use cases and simulation output storage
  - online systems - DAQ[3] - high-throughput IO, WORM storage

- tape storage
  - Cern Tape Archive **CTA** - namespace and disk buffer in front of tape system, WORM storage

4 Development Areas

This is a brief overview of the major development areas of 2019, which we are illustrating in detail in the following sections:

- namespace architecture (implemented in the MGM[4] service)
- storage consistency (provided by the FST[5] service)
- filesystem access (FUSE client and access control lists ACL)
- tape integration (CTA)
- protocols & API (Protobuf [6],XrdHttp [7], GRPC [8])
- tokens & authorisation

4.1 Namespace Architecture Evolution

4.1.1 QuarkDB

The namespace architecture until 2017 was based on a stateful metadata service. The namespace was in-memory and loaded on service startup. Changes were written to a changelog file used for master/slave replication. The in-memory solution reached its limit in 2017 on the largest instance at CERN **EOSUSER** with more than half a billion of files and memory requirements over one TB. The startup time of the service was more than one hour and had led to problematic service outages whenever a restart was either necessary or forced by a software bug. The architecture has evolved into an almost stateless metadata service with an active/-passive architecture in combination with service sharding to scale out metadata performance. This transition is shown in figure 1.

The enabling technology was the introduction of QuarkdDB (see [9]).
QuarkDB [10] is a persistent KV store offering part of the REDIS [11] protocol. Persistency is provided by RocksDB [13], high-availability is provided by a cluster of servers using RAFT [12] as consensus algorithm. QuarkDB is highly available, high performance, scalable and provides low-latency access to billions of small objects. The QuarkDB API provides the following data types/mechanisms, which are used by the MGM service:

- KV pairs

[3]Data acquisition system
[4]Metadata Management Server
[5]File Storage Server

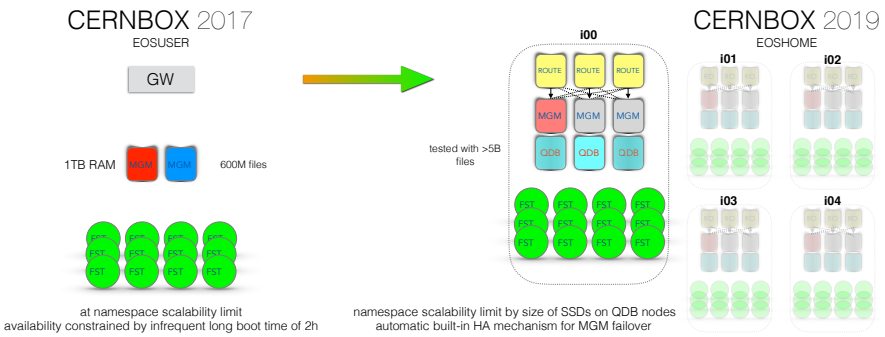


Figure 1. Namespace Architecture Evolution

- sets
- hashes
- pub-sub
- leases

Figure 2 illustrates a typical deployment setup with three database services. The MGM service uses still a similar implementation of in-memory metadata - his time only as a cache. Performance comparisons show minimal differences between in-memory and QuarkDB based namespaces. The major improvement is that the service startup time has decreased in some cases from 1h to 10s.

4.1.2 Namespace High Availability

A new high-availability model is based on the lease functionality offered by QuarkDB. In a typical deployment three MGM services try to obtain/renew an exclusive lease for the active MGM state every few seconds. If a lease is obtained the configuration is automatically loaded and this MGM services becomes active. All MGMs without a lease stay passive and redirect requests to the active MGM service. Service fail-over can be forced using the command line interface or happens automatically when the active MGM does not renew his lease. This is illustrated in figure 3.

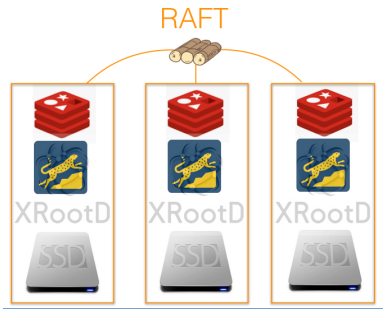


Figure 2. QuarkDB three-node deployment

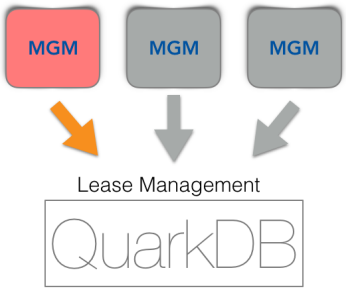


Figure 3. Lease mechanism using QuarkDB

4.2 Filesystem Consistency

For EOS v4.6 the filesystem consistency check functionality (FSCK) has been re-engineered. With the transition from in-memory to QuarkDB namespace the implementation was not usable anymore in large instances. This was problematic because some instances had accumulated inconsistencies over 9 years, which in turn can lead to data loss.

The FSCK functionality is provided by three elements:

- backward consistency check
- forward consistency check
- repair engine

The backward consistency check compares filesystem contents to the namespace information. A data scanner runs on every FST node and collects inconsistencies in size, checksum and file layout. The CERN default setting is to scan all data one per week - scanning means here to read complete files using direct IO to compute and validate their checksums. This check collects also orphaned files: these are files which exist on a disk, but there is a location pointer from the namespace to access these.
The forward consistency check compares namespace information to filesystem contents. The MGM scanner identifies missing replicas on filesystem.
The last component is the repair engine which collects and report all errors from backward and forward scan and triggers automatic repair actions.

4.3 Filesystem Access

The FUSE daemon providing filesystem access has seen four implementation generations in EOS.

- xrootdfs - native XRootD FUSE mount
- eosd v1 - native EOS FUSE mount
- eosd v2 - native EOS FUSE mount using kernel caching with TTLs
- eosxd - native EOS FUSE mount client with active cache-invalidation using callbacks

One of the main problem areas in filesystem access in the past was the expected POSIX compatibility of the mount client. For **eosd** this has been very limited and certain features like hard links, byte-range locks and cross-client metadata synchronisation were not existing. The latest generation client **eosxd** provides now most POSIX features:

- POSIX and BSD **file locking**
- **hard links** within directories only
- **RichACL** [14] client support
- local file **caching** and **write journaling**
- **bulk deletions** and *rm -rf*
- **strong security** like kerberos, X509 and OIDC/token support [15]
- subtree based user, group and project **quota**

The latest development project in this area is the support of copy on write **snapshot support** to offer the possibility of consistent backups.

4.4 Tape Integration

The storage group at CERN is developing a replacement of the CASTOR [16] software called CERN tape archive CTA [17]. This service entered production in early 2020 and allows the coupling of EOS to CTA.

EOS provides a disk cache while CTA implements the management and transfer of tape data. A file on tape is visible in EOS as an offline replica. EOS has been extended with a synchronous notification interface to inform CTA about changes in the disk cache.

In a CTA enabled instance data is write-once read-many (WORM). CTA uses a new EOS mechanism to inject a replica from tape into the disk cache. An extension of the XRootD protocol allows checking migration to tape and trigger recalls from tape. This allows to drop the need for a storage resource manager (SRM), which had been introduced many years ago to provide this interface but highlighted many problems in the SRM standard.

The operation model of CTA is to provide a big disk-only EOS instance combined with a smaller CTA enabled EOS instance. Transfers between disk and tape storage are orchestrated by the file transfer service (FTS [18]). Regular users have only direct access to the disk-only EOS instance, while data migration and recall is only provided to production services (see figure 4

4.5 Protocol Support

4.5.1 GRPC

We added GRPC support with token and X509 authentication as a new namespace API. The idea behind GRPC is to provide to high-level services a well standardized protocol with

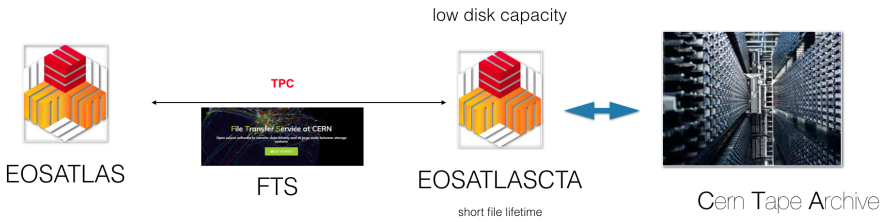


Figure 4. EOS/CTA deployment model with disk-only and tape-enabled EOS instances

support for almost any language. The primary target at CERN is the CERNBox service. Clients can be mapped from their *DN* or from an individually created token to a *uid*/*gid* pair. Another use case of GRPC is the service migration from CASTOR to CTA. All the existing meta data has to be migrated from an Oracle database into the EOS namespace and GRPC offers a very efficient way to do bulk insertions.
GRPC protocol is not implemented using the XRootD protocol bridge interface, which is used for XrdHttp (see 4.5.2). The library provided C++ embedded GRPC server listens on a separate port with a separate thread pool. GRPC functions access the XRootD OFS filesystem object to access the same XRootD server API, which provide the storage server implementation for XRootD and HTTP(S) protocols.
The currently supported operations in GRPC are:

- mkdir/rmdir
- touch
- rm/unlink
- ls/find
- rename
- symlink
- setxattr
- chown/chmod/acl
- list-version/purge-version

API calls like *find* use the GRPC streaming interface allowing clients to process results as they are produced by the MGM service. In particular this allows reducing response time for large queries and to minimize memory requirements in the MGM service.
GRPC by default uses PROTOBUF and is suitable for metadata access. GRPC using flatbuffers [21] could replace WebDav/HTTP(S)/XRootD protocol as data transport protocol in the future.

4.5.2 HTTP(S)

Currently HTTP support in EOS is based on libmicrohttpd [19]. HTTPS can only be provided using a NGINX proxy in front of EOS native HTTP access. XrdHttp however is a native implementation of HTTP and HTTPS access for the XRootD framework and we have now bridged the libmicrohttpd based implementation to XrdHttp. Both interfaces can be used in parallel which will allow us in the future to deprecate libmicrohttpd. XrdHttp also provides

now macaroons [20], token and HTTPS third-party copy support with optional delegation, which is envisaged by the WLCG[6] community to replace gridFTP protocol in the near future.

4.5.3 S3

AARNet uses a MiniO [22] plug-in to provide S3 compatible access to EOS instances. EOS itself has an internal S3 implementation. The coverage of the implemented S3 API is relatively small, in particular bucket handling and ACL functions are missing. The internal S3 interface might be deprecated in the next major release version.

4.6 Tokens

Besides bearer token support which is prepared in the context of WLCG and the XRootD project, we have added a proprietary token format. This token format allows one to grant bearers of a token scoped access to EOS instances. An EOS token is a serialized protobuf structure, which is zlib compressed and base64 URL-encoded and signed. This allows to use identical token formats for HTTP(S) and XRootD protocol. A token carries the following entities:

- a namespace scope: file, directory or tree
- an ACL entry replacing locally stored ACLs using identical EOS ACL syntax
- an optional role e.g. the owner to use when creating a file
- an optional set of origin restrictions for clients and their authentication
- a generation value allowing immediate token revocation without changing encryption keys
- an expiration time

Figure 5 shows the JSON representation of a token, figure 6 shows how token are used and created using the EOS command line interface.


```
{
 "token": {
  "permission": "rwx",
  "expires": "1571319146",
  "owner": "",
  "group": "",
  "generation": "1",
  "path": "/eos/dev/token",
  "allowtree": false,
  "vtoken": "",
  "voucher": "baecb618-f0e4-11e9-85d9-fa163eb6b6cf",
  "requester": "[Thu Oct 17 15:47:59 2019] uid:0[root] gid:0[root] tident:root.13809:107@localhost
name:daemon dn: prot:sss host:localhost domain:localdomain geo:cern sudo:1",
  "origins": []
 },
 "signature": "daUeOZafRUt6VfQZ+g3FMbR/ZA5WvARELqFwdQxbyFU=",
 "serialized":
"CgJyeBDq2qHtBTIJL2Vvcy9kZXYvSiRiYWVjYjYxOC1mMGU0LTExZTktODVkOS1mYTE2M2ViNmI2Y2ZSnAFbVGh1IE9jdCAxNyAxNTo0Nzo1
OSAyMDE5XSB1aWQ6MFtyb290XSBnaWQ6MFtyb290XSB0aWRlbnQ6cm9vdC4xMzgwOToxMDdAbG9jYWxob3N0IG5hbWU6ZGFlbW9uIGRuOiBwc
m90OnNzcyBob3N0OmxvY2FsaG9zdCBkb21haW46bG9jYWxkb21haW4gZ2VvOmFqcCBzdWRvOjE=",
 "seed": 1399098912
}
```


Figure 5. JSON representation of an EOS token

Tokens can be used to provide delegation, sharing and single file access token functionality (similar to signed S3 URLs). Details to the token format can be found under [23].

[6]Worldwide LHC Computing Grid

```
# as a filename
xrdcp root://myeos//zteos64:MDAwMDAwNzR4nONS4WIuKq8Q-Dlz-ltWI3H91Pxi_cSsAv2S_OzUPP2SeAgtpMAY7f1e31Ts-od-
rgcLZ_a2_bhwcZO9cracy /tmp/

# via CGI
xrdcp "root://myeos//eos/myfile?authz=zteos64:MDAwMDAwNzR4nONS4WIuKq8Q-Dlz-
ltWI3H91Pxi_cSsAv2S_OzUPP2SeAgtpMAY7f1e31Ts-od+rgcLZ_a2_bhwcZO9cracy" /tmp/

eos token --path /eos/myfile --expires $LATER
zteos64:MDAwMDAwNzR4nONS4WIuKq8Q-Dlz-ltWI3H91Pxi~cSsAv2S~OzUPP2SeAgtpMAY7f1e31Ts-od-
rgcLZ~a2~bhwcZO9cracyhm1b3c6jpRIEWWOws71Ox6xAABeTC8I
```

Figure 6. Usage and creation of an EOS token

5 Community Events

The CERN EOS team is organizing a yearly workshop beginning of each year to foster exchange between users and providers of EOS services. Information and all presentations of the fourth edition 2020 can be found under [24]

6 Summary

EOS development has left the area of rapid feature extensions. Focus today is on consolidation of a new architecture, improvement of reliability & consistency and optimization of internal storage services to profit from QuarkDB by code refactoring. Support for the HTTP ecosystem is important based on the WLCG community direction towards HTTPS based file transfers and access. GRPC is strategic as metadata API, WebDAV as data access API. Erasure Coding (EC) is another major cost reduction factor in EOS deployments. Details about EC are discussed in a separate publication of this conference [25].
The architectural change started in 2018 has proven to bring great improvements in usability, stability and efficiency of EOS services. It is the necessary preparation for the scalability and performance challenges in future LHC runs and requirements of many other science communities.

References

[1] Exabyte Scale Storage at CERN, Andreas J Peters and Lukasz Janyst 2011 J. Phys.: Conf. Ser. 331 052015
[2] CERNBox - the cloud storage solution from CERN, https://cernbox.web.cern.ch
[3] Australias Acadamic and Research Network, https://www.aarnet.edu.au
[4] Joint Research Centre JRC, https://ec.europa.eu/jrc/
[5] eXtreme Data Cloud, https://www.extreme-datacloud.eu
[6] Protocol Buffers, https://github.com/protocolbuffers
[7] XRootd HTTP Interface, https://github.com/xrootd/xrootd/tree/master/src/XrdHttp
[8] gRPC - a high-performance RPC framework, https://grpc.io
[9] Scaling the EOS namespace – new developments, and performance optimizations, EPJ Web of Conferences 214, 04019 (2019) CHEP 2018, https://doi.org/10.1051/epjconf/201921404019
[10] QuarkDB - a highly available datastore, https://github.com/gbitzes/QuarkDB
[11] Redis - in-memory data structure store, https://redis.io
[12] In Search of an Understandable Consensus Algorithm, D. Ongaro and J. Ousterbout, Stanford University, https://raft.github.io/raft.pdf

[13] RocksDB - an embeddable, persisten key-value store, https://github.com/facebook/rocksdb

[14] Implementing an advanced access control model on Linux, Aneesh Kumar, Andreas Gruenbacher, Greg banks, https://www.kernel.org/doc/ols/2010/ols2010-pages-19-32.pdf

[15] OpenID Connect, https://openid.net/connect

[16] CERN Advanced STORage manager, https://castor.web.cern.ch

[17] CERN Tape Archive, https://cta.web.cern.ch

[18] File Transfer Service, https://fts.web.cern.ch

[19] GNU Libmicrohttpd, https://www.gnu.org/software/libmicrohttpd

[20] Macaroons: Cookies with Contextual Caveats for Decentralized Authorisation in the Cloud, Arnar Birgisson and Joe Gibbs Politz and Úlfar Erlingsson and Ankur Taly and Michael Vrable and Mark Lentczner, 2014, Network and Distributed System Security Symposium

[21] Flatbuffer, https://google.github.io/flatbuffers/

[22] MINIO - High Performance, Kubernetes-Friendly Object Storage, https://min.io

[23] Using Eos Tokens for Authorisation, http://eos-docs.web.cern.ch/eos-docs/using/tokens.html

[24] EOS Workshop 2020, https://indico.cern.ch/event/862873

[25] Erasure Coding for production in the EOS Open Storage System, this proceedings, Computing in High Energy Physics, CHEP 2020