Computing and Software for Big Science (2021) 5:25
https://doi.org/10.1007/s41781-021-00071-1

ORIGINAL ARTICLE

Evaluating CephFS Performance vs. Cost on High-Density Commodity Disk Servers

Andreas J. Peters[1] · **Daniel C. van der Ster**[1]

Received: 1 July 2021 / Accepted: 27 September 2021

Abstract
CephFS is a network filesystem built upon the Reliable Autonomic Distributed Object Store (RADOS). At CERN we have demonstrated its reliability and elasticity while operating several 100-to-1000TB clusters which provide NFS-like storage to infrastructure applications and services. At the same time, our lab developed EOS to offer high performance 100PB-scale storage for the LHC at extremely low costs while also supporting the complete set of security and functional APIs required by the particle-physics user community. This work seeks to evaluate the performance of CephFS on this cost-optimized hardware when it is combined with EOS to support the missing functionalities. To this end, we have setup a proof-of-concept Ceph Octopus cluster on high-density JBOD servers (840 TB each) with 100Gig-E networking. The system uses EOS to provide an overlayed namespace and protocol gateways for HTTP(S) and XROOTD, and uses CephFS as an erasure-coded object storage backend. The solution also enables operators to aggregate several CephFS instances and adds features, such as third-party-copy, SciTokens, and high-level user and quota management. Using simple benchmarks we measure the cost/performance tradeoffs of different erasure-coding layouts, as well as the network overheads of these coding schemes. We demonstrate some relevant limitations of the CephFS metadata server and offer improved tunings which can be generally applicable. To conclude, we reflect on the advantages and drawbacks related to this architecture, such as RADOS-level free space requirements and double-network penalties, and offer ideas for improvements in the future.

Keywords Scientific computing · Distributed file systems · Object storage

Introduction

In the coming years, higher luminosity data taking at the Large Hadron Collider will place increased demands on the storage throughput, capacity, and durability of the storage at CERN [1]. Recent innovations in open source storage systems demonstrate a compelling level of features and maturity [2], raising the question of if and how these components might play a role in future physics storage systems. Off-the-shelf software is missing important high-level features and there is limited evidence of the efficiency on the cost-optimized hardware critical to LHC physics programmes; however, a complete solution might be constructed by layering HEP-specific gateways on top of the open source offerings [3]. In this paper we describe and evaluate a novel combination of one such open source clustered storage system, CephFS [4], with EOS [5], the high performance and low cost storage solution designed at CERN for LHC data taking.

CephFS and Its Application at CERN

CephFS is a modern clustered filesystem which acts as an NFS-replacement in typical computing scenarios for a single data centre, including home directories, HPC scratch areas, or shared storage for other distributed applications. The software implements a scale-out architecture for data and metadata IOPS: data and metadata are persisted in the distributed object store RADOS [7] and the metadata is mediated by a small number of replaceable MDS servers. Capacity and performance can be increased dynamically without downtime: raw capacity and IOPS by adding servers to the RADOS backend, and metadata scales by re-assigning filesystem subtrees to new MDS servers.

✉ Daniel C. van der Ster
daniel.vanderster@cern.ch

Andreas J. Peters
andreas.joachim.peters@cern.ch

[1] CERN, Geneva, Switzerland

Published online: 09 November 2021

RADOS provides a durable object store using either replication—typically 3 copies—or erasure coding with arbitrary layouts, for example with four data stripes and two parity stripes (EC4,2). RADOS uses CRUSH [8] to place objects across failure-domains: in this way, systems may be designed to tolerate failures at the disk, host, rack, power, or switch level depending on local constraints.

CephFS aims to provide consistency guarantees equal to those of a local filesystem. To accomplish this the MDSs delegate to clients a range of IO capabilities which grant different POSIX operations to be carried out synchronously or asynchronously depending on the real time need for parallel access to directories and files. For example, a file opened by one writer with no other clients may be written to quickly with client-side buffering and persisted only periodically, whereas a file with concurrent writers/readers must be persisted synchronously and clients are not permitted to cache their reads.

CERN has operated several CephFS clusters in production since 2017, and as of 2021 we use CephFS in three settings:

- *HPC Scratch* an all-flash cluster built using Ceph OSDs co-located on SLURM compute nodes [9], using a local idle node as the MDS; 3x replication with usable capacity of around 110 TiB;
- *OpenStack Manila* [10] a mixed HDD/SSD cluster offering general purpose shared storage for IT and scientific applications; 3x replication with usable capacity of around 1 PiB;
- Enterprise Groupware: an all-flash cluster co-located on OpenStack hypervisors which are dedicated to a new groupware solution for the CERN community; EC2,2 with usable capacity of around 100 TiB.

In these settings, CephFS has demonstrated its robustness and performance over several years of operations. These and other Ceph clusters at CERN have survived several external outages and lived through three hardware procurement cycles: throughout this we have noted very few incidents related to data availability, loss, or corruption.

Despite these strengths, CephFS is currently limited at CERN to the previously listed use-cases due to several missing features which are essential for the high-energy physics community:

- authentication mechanisms and user/group management: SciTokens [11], X.509, Kerberos, quota and access control via eGroups [12];
- storage protocols and features: HTTPS, XRootD [13], and third-party copy;

Moreover, CephFS has not yet been extensively tested at CERN for high throughput LHC data-taking, for example with write rates exceeding 20 GiB/s.

Introduction to EOS

EOS is a large scale storage system developed at CERN currently providing 350 PB of capacity to both physics experiments and 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. EOS is implemented as plug-ins to the XRootD framework. Files are stored using either replication or erasure-coding and organized in a hierarchical namespace using QuarkDB [6] as a persistency backend. The frontend MGM service provides cached access to the namespace and other metadata. Storage nodes run one or several FST services to provide access to data stored on a locally mounted filesystem (FileIO[posix]) or remote storage (XrdIo[root protocol], DavixIo[Webdav/S3]). As for any Linux filesystem, files are organised as inodes. The MGM services translates logical path names to inodes and FST servers store all data by inode name. The namespace on the local or remote FST filesystems is organized using a simple inode hash prefix directory and the hexadecimal inode name to build a physical path for a given inode number. The only features require of the FST local filesystem are the basic POSIX semantics and extended attributes.

It is, therefore, straightforward to replace a local FST filesystem with CephFS. In this case, data accessed via an FST makes use of a remote CephFS filesystem. In such a deployment model, redundancy and data high-availability is delegated to the CephFS layer and EOS is configured to store files with single replica layouts.

System Architecture

Ceph Backend Storage

A proof-of-concept was constructed out of eight disk servers each with the following specifications:

- Dual Intel Xeon Silver 4216 CPUs and 192 GiB RAM;
- Mellanox ConnectX-5 network interface supporting 100Gb/s Ethernet;
- 60x 14 TB enterprise SATA HDDs connected via a single SAS3616 host bus adapter;
- 1×1 TB SSD.

These high-density disk servers are not yet used in production at CERN. Presently, EOS uses servers with four 24-disk enclosures connected to frontends with 192 GiB of memory.

Because of the large amount of memory required by each Ceph OSD, these 96-disk EOS systems would require clustering of disks or extra memory. On the contrary, the high-density servers evaluated in our PoC are ideal, since they provide 3 GiB of memory per OSD.

On this hardware we installed Ceph using Octopus version 15.2.8. Each server's disks were prepared to run 61 Ceph OSDs: the HDDs and SSDs were used to host the CephFS data and metadata pools, respectively. A single virtual machine non-local to the disk servers acts as the MON, MGR, and MDS for the cluster. CephFS was configured with top level directories each backed by a different RADOS pool with erasure coding and CRUSH configured as follows:

- /ec42: Reed–Solomon coding with $k = 4$, $m = 2$; each host has at most one object chunk; 4096 placement groups, 51.2 per OSD;
- /ec82: Reed–Solomon coding with $k = 8$, $m = 2$; each host has at most two object chunks; 2048 placement groups, 42.6 per OSD;
- /ec162: Reed–Solomon coding with $k = 16$, $m = 2$; each host has at most three object chunks; 1024 placement groups, 38.4 per OSD;

We, additionally, evaluated the effect of object size on performance using CephFS's file layout extended attributes. The RADOS placement groups were balanced to a max deviation of one per OSD.

EOS Frontend Server

We used eight additional identical machines as EOS FST nodes mounting CephFS using the kernel client included with CentOS 8.2. For each FST we created one separate data directory in the CephFS mount directory and configured these as eight EOS filesystems. The setup is shown in Fig. 1, while Figs. 2 and 3 show the EOS space and filesystem configuration.

Testing and Results

We performed two sets of benchmarks to evaluate the performance of the CephFS backend and EOS frontend services:

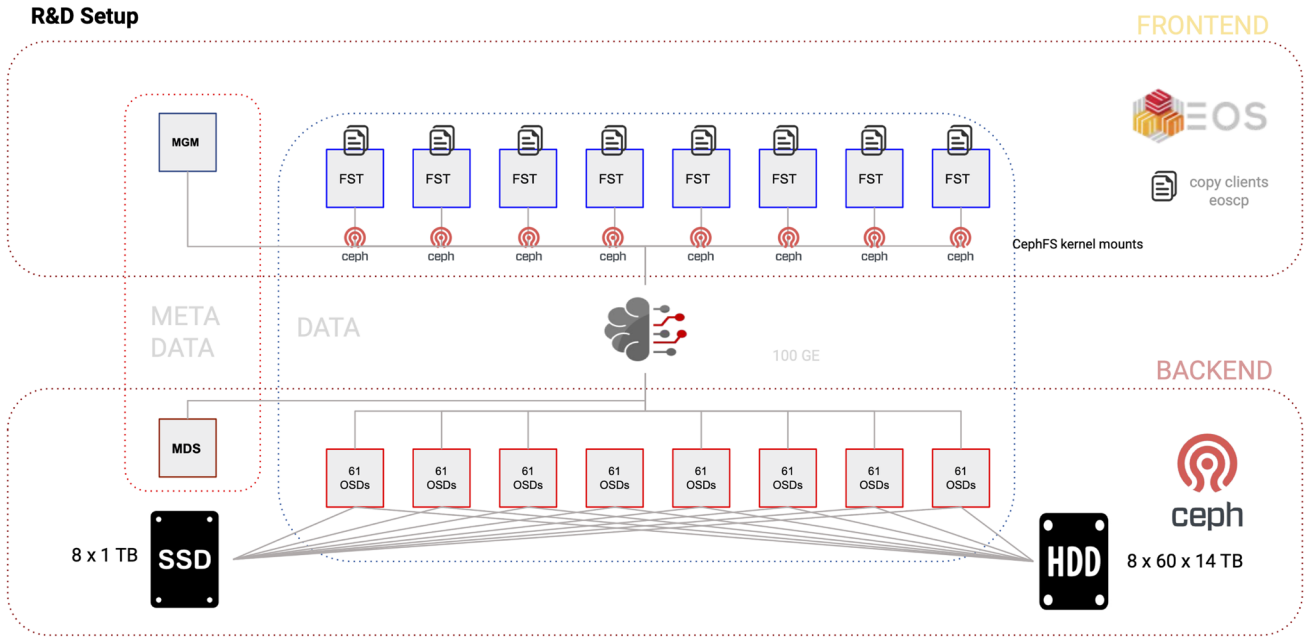

Fig. 1 Test setup with eight backend disk server (blocks 1–8) to run Ceph OSDs and eight frontends (blocks 9–16) running EOS FSTs and CephFS kernel mounts. MGM and MDS handle metadata for EOS and CephFS, respectively. CephFS metadata is stored on the SSDs, while the data objects are stored on the HDDs

type	name	groupsize	groupmod	N(fs)	N(fs[rw])	sum(used bytes)	sum(capacity)
spaceview	ceph	8	8	8	8	2.96 PB	6.72 PB

Fig. 2 Configuration for a `ceph` space provided by eight CephFS mounts

host	port	id	uuid	path	schedgrp	headroom	boot	config	drain	active	scanintv	health
fst1	1095	1	...81d6d90be9d4	/cephfs/01	ceph.0	0.00	booted	rw	nodrain	online	604800	OK
fst2	1095	2	...6f2214a3f93d	/cephfs/02	ceph.0	0.00	booted	rw	nodrain	online	604800	OK
fst3	1095	3	...9a81d82a4e1b	/cephfs/03	ceph.0	0.00	booted	rw	nodrain	online	604800	OK
fst4	1095	4	...0ceb1b68f5ee	/cephfs/04	ceph.0	0.00	booted	rw	nodrain	online	604800	OK
fst5	1095	5	...178b4eefc5f9	/cephfs/05	ceph.0	0.00	booted	rw	nodrain	online	604800	OK
fst6	1095	6	...7f9bbe969b28	/cephfs/06	ceph.0	0.00	booted	rw	nodrain	online	604800	OK
fst7	1095	7	...0456ae4c328c	/cephfs/07	ceph.0	0.00	booted	rw	nodrain	online	604800	OK
fst8	1095	8	...ed0932b88cd1	/cephfs/08	ceph.0	0.00	booted	rw	nodrain	online	604800	OK

Fig. 3 Filesystem configuration for eight CephFS mounts inside the EOS ceph space

– *backend* using *dd* commands directly on client kernel mounts, to study the streaming performance of the CephFS backend;
– *frontend* using XRootD protocol *eoscp* copy clients via EOS FSTs, to study their impact on overall performance.

Benchmarking Setup

Each benchmark uses ten parallel streams per `ceph` mount (80 in total) to create/write or read files of 2 GB size each. Benchmarks were generally executed for several hours to observe stable running conditions; to test for performance degradation we, additionally, tested when the backing CephFS was filled up to 95%. In the frontend benchmarks the concurrent number of streams can fluctuate by design. The average number of streams was configured to be again ten per client mount. We also tuned the RADOS object size parameters to improve write performance for each erasure coding layout.

We have benchmarked the raw controller and network speeds. Both IO paths reach the design spec 12 GiB/s under optimal load conditions. We, additionally, measured the limitation of a single CephFS kernel client; it reaches a maximum of 6 GiB/s for reading and writing. Figure 4 shows that write throughput scales linearly with the number of clients until the maximum cluster write performance is reached with 6 out of 8 client nodes. Figure 5 shows equally that the read throughput is not client-limited when sufficiently many are working concurrently: read scaling begins linearly and then shows a damped curve most likely due to platter seek times increasing with the number of concurrent streams.

Results

Figure 6 shows the dependency of the relative write performance depending on the volume usage of the hard disks: 100% performance is equivalent to 31 GiB/s. The degradation is consistent with an observed increase in IO wait on the OSD nodes.

Table 1, visualized in Figs. 7 and 8, summarizes the write performance for various erasure coding layouts and object sizes measured with space usage under 10%. Table 2, visualized in Figs. 9 and 10, summarizes the read performance for various erasure coding layouts, object sizes and read blocksizes measured with space usage under 10%. Both tables show the average time to upload or download a 2 GiB file

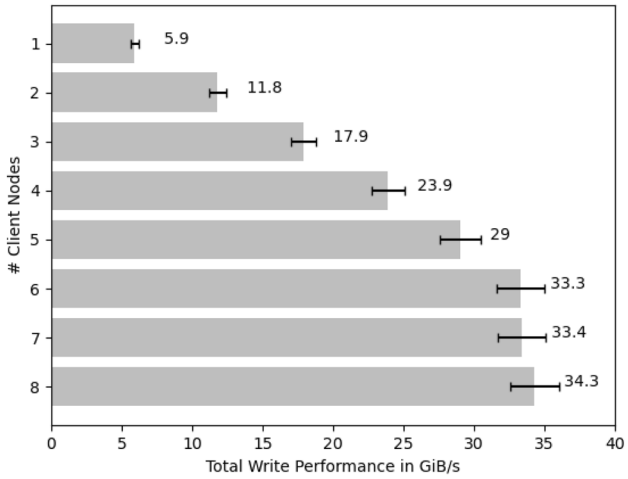

Fig. 4 Performance scaling for writing with number of client nodes using EC16,2;64M. Write throughput scales with the number of clients until 6 out of 8 are running concurrently

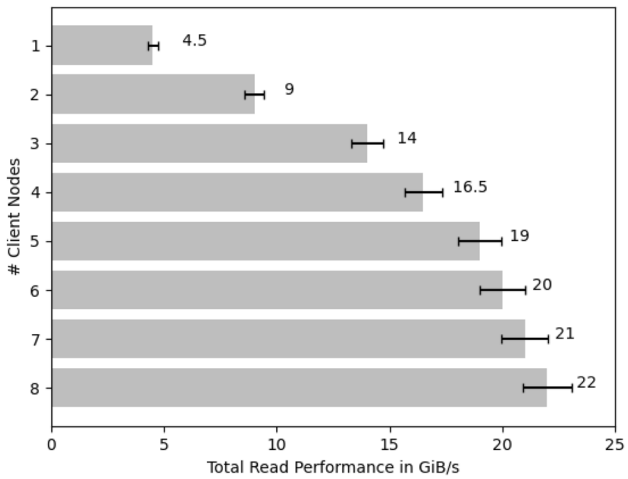

Fig. 5 Performance scaling for reading with number of client nodes using EC16,2;64M. Read performance begins scaling linearly but is damped above 3 concurrent streams

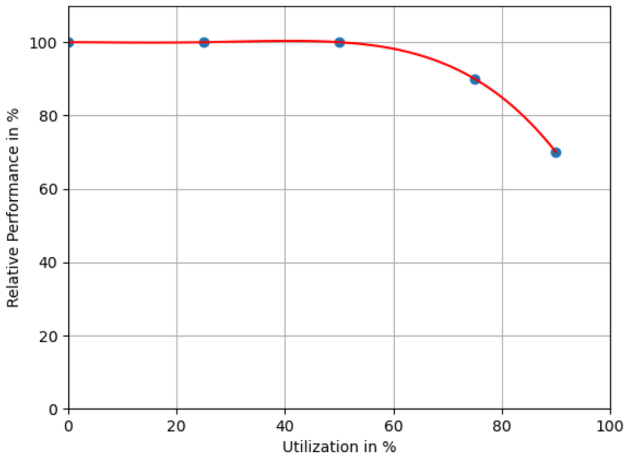

Fig. 6 Correlation of write performance with CephFS total usage. Peak performance is achieved when the backend CephFS is between 0 and 50% full, but above 75% usage the performance is decreased

with 80 parallel dd commands running on 8 client nodes. Each benchmark uses 8000 files and 16 TiB of data volume. In addition, the standard deviation, the average stream rate, the 99th percentile and the maximum value for the IO time is shown.

It is intuitive that high-performance streaming favours large blocksizes. EC16,2 provides the highest write throughput, because it has the smallest parity payload compared to EC8,2 or EC4,2 configurations; however these large blocksizes demonstrate long tails due to the increased variance of the object distribution. The blocksize impact is more pronounced while reading. The default read-ahead setting of the kernel mounts are 8 MiB; blocksizes larger than this help to improve the read throughput. The object size impact also manifests when reading, since more disk seeks are expected per GiB served.

Table 3, visualized in Fig. 11, shows the impact accessing the CephFS via EOS as a frontend service. The overall performance does not change but the usage of XRootD protocol increases tail effects due to unfair stream scheduling

Table 1 Write performance for various erasure coding layouts (ECk,m) and object sizes (;*s*M). IO times and rates are shown per 2 GiB file stream with 80 concurrent IO streams

–	Avg [s]	Sigma [s]	Rate [MiB/s]	99th percentile [s]	Max [s]
EC4,2;4M	6.26	1.30	319	8.95	11.07
EC4,2;16M	6.4	1.42	312	9.16	13.81
EC8,2;16M	4.95	0.89	403	7.1	10.08
EC16,2;4M	5.76	0.95	347	7.57	10.56
EC16,2;16M	4.96	1.03	402	8.10	10.31
EC16,2;64M	4.7	2.68	426	17.32	41.62
EC16,2;128M	4.73	2.07	422	13.36	29.95

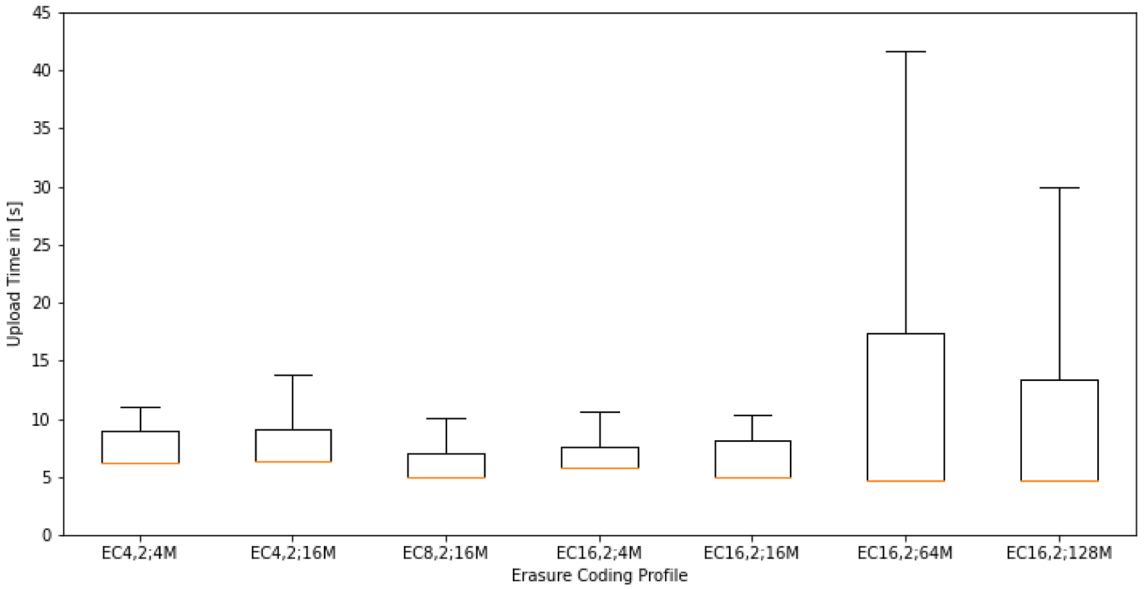

Fig. 7 Write performance tails: the red line shows the average upload time, the box limit shows the 99 percentile and the error limit the maximum upload time observed for a given erasure coding layout. Based on data from Table 1

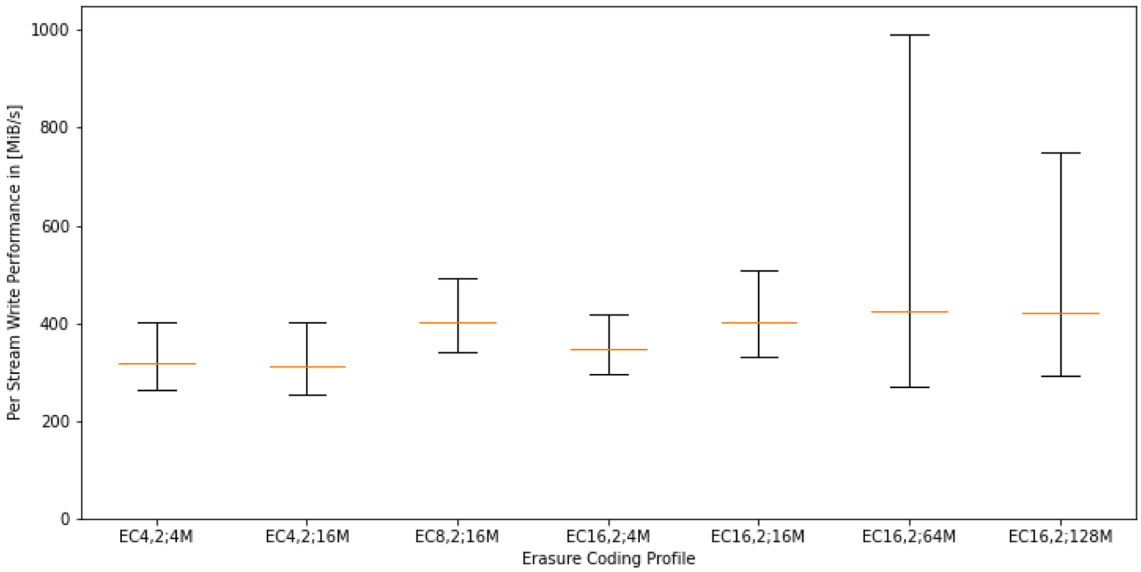

Fig. 8 Average write stream speed with standard deviation for various erasure coding layouts, based on data from Table 1

Table 2 Read performance measurements for various erasure coding layouts (ECk,m), object sizes (;*s*M) and `dd` blocksize (,*b*M)

–	Avg [s]	Sigma [s]	Rate [MiB/s]	99th percentile [s]	Max [s]
EC4,2;4M,1M	14.70	1.24	136	17.37	19.96
EC4,2;16M,8M	11.25	0.54	177	12.61	13.87
EC8,2;16M,1M	13.02	0.64	153	14.61	15.96
EC8,2;16M,128M	5.23	0.61	382	6.81	15.19
EC16,2;4M,1M	25.23	3.85	79	36.74	48.63
EC16,2;4M,128M	13.63	4.05	146	28.61	54.68
EC16,2;16M,1M	11.59	0.84	172	13.7	15.76
EC16,2;16M,128M	4.89	0.91	408	7.61	13.53
EC16,2;64M,1M	9.53	0.78	209	11.46	19.53
EC16,2;64M,128M	5.23	0.31	381	6.07	7.42
EC16,2;128M,1M	9.44	1.17	211	13.02	20.60
EC16,2;128M,128M	5.26	0.38	380	6.27	7.56

IO times and rates are shown per 2 GiB file stream with 80 parallel streams. The default kernel readahead setting of 8 MiB is used

when writing. These tail effects could be eliminated by throttling each stream to a nominal 325/350 MiB/s client side. The read performance actually benefits from the frontend, because the blocksize used in EOS transfers is larger than the baseline comparison of the native CephFS backend.

Tuning Ceph

During our performance evaluations we came across a few areas, where the default Ceph configurations and warnings were not ideal:

Client throttling bytes in transit By default a librados client limits the number of in-flight writes to 100MiB. We observed that this throttle was reached often, capping the achievable write performance. Setting `objecter_inflight_op_bytes` to `10485760000` removed this artificial limitation.

MDS caps recall tuning The EOS *fsck* process is used to check the consistency of the EOS namespace with the backend CephFS storage. This process puts continuous pressure on the MDS to stat all files as quickly as possible, which can lead to a scenario, where clients acquire caps more quickly than the MDS will ask them to be recalled, causing an out-of-memory error on the MDS. Improved default caps recall settings, effectively increasing the caps grant/recall rate by more than 5x were suggested

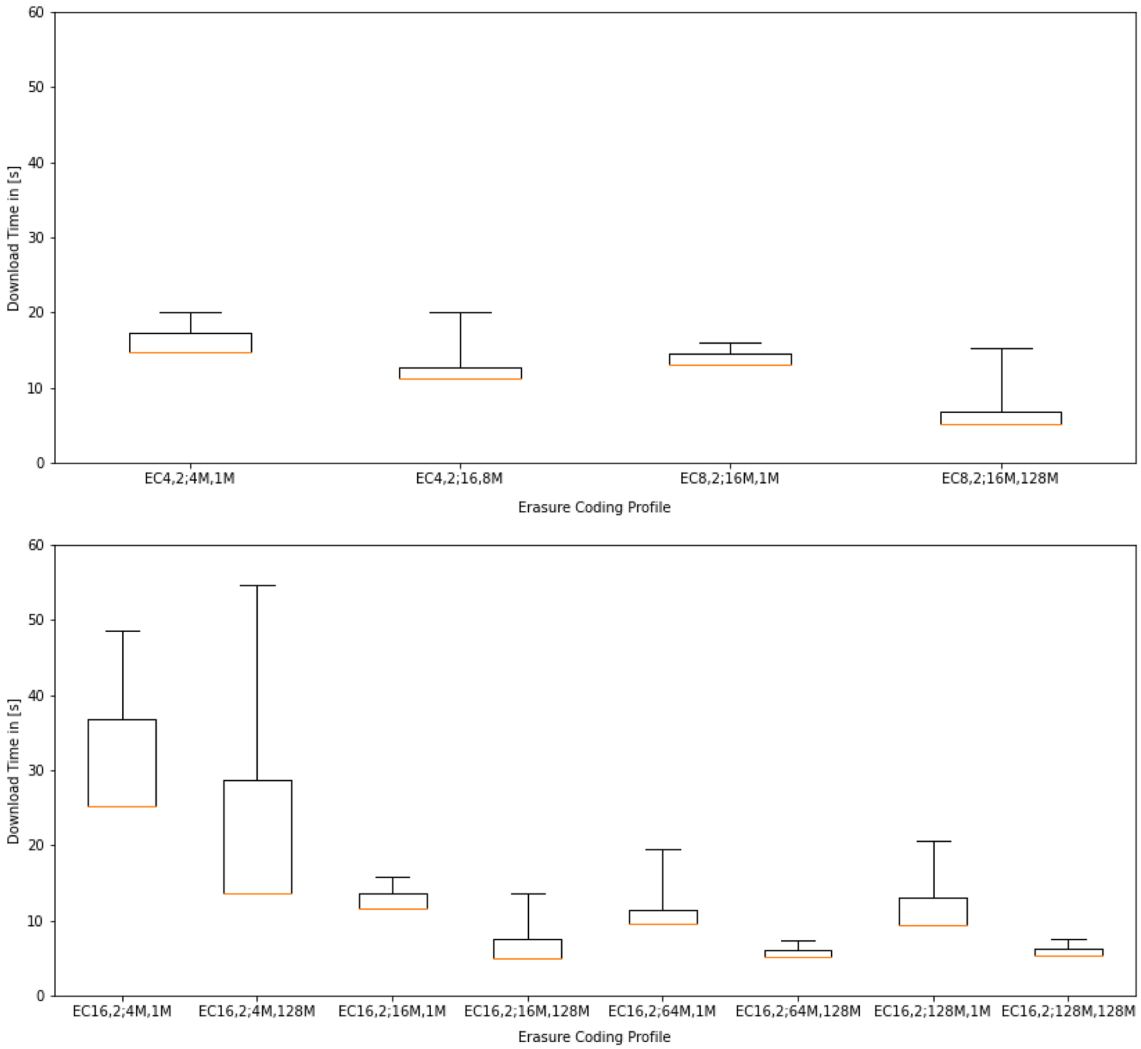

Fig. 9 Read performance tails: the red line shows the average download time, the box limit shows the 99 percentile and the error limit the maximum download time observed for a given erasure coding layout. Based on data from Table 2

and accepted by the upstream community [14]. In addition, EOS *fsck* can now be throttled to scan the namespace within a configurable interval.

Single high-latency OSD ruins everything: At one point in our testing the cluster-wide write performance dropped from a nominal 25 GiB/s to under 5 GiB/s. After troubleshooting it was found to be the result of a single HDD with a poor physical SATA connection, causing small IO requests to take longer than 2*s* on average. Once this disk was removed from the cluster, the expected performance returned. This type of issue had not been previously seen in production at CERN. Because Ceph does not currently detect and warn about this type of issue, we are currently working on an external probe which warns when anomalous OSD latency is detected, to be contributed upstream if it proves useful.

Conclusions and Discussion

The evaluated setup based on a high-density disk servers provides excellent performance with various erasure coding schemes and allows up to 4 GiB/s read or write data payload per node for streaming access. A substantial performance degradation with increased CephFS usage has to be taken into account when planning a service, and a safe maximum usable space threshold without risking operational hazards during hardware failures requires more practical experience. We have tested filling the backing CephFS up to 95%, using `upmap` data balancing to achieve a uniform disk utilization.

Erasure code write performance performed at nearly the network connectivity limit; neither the CPUs nor the

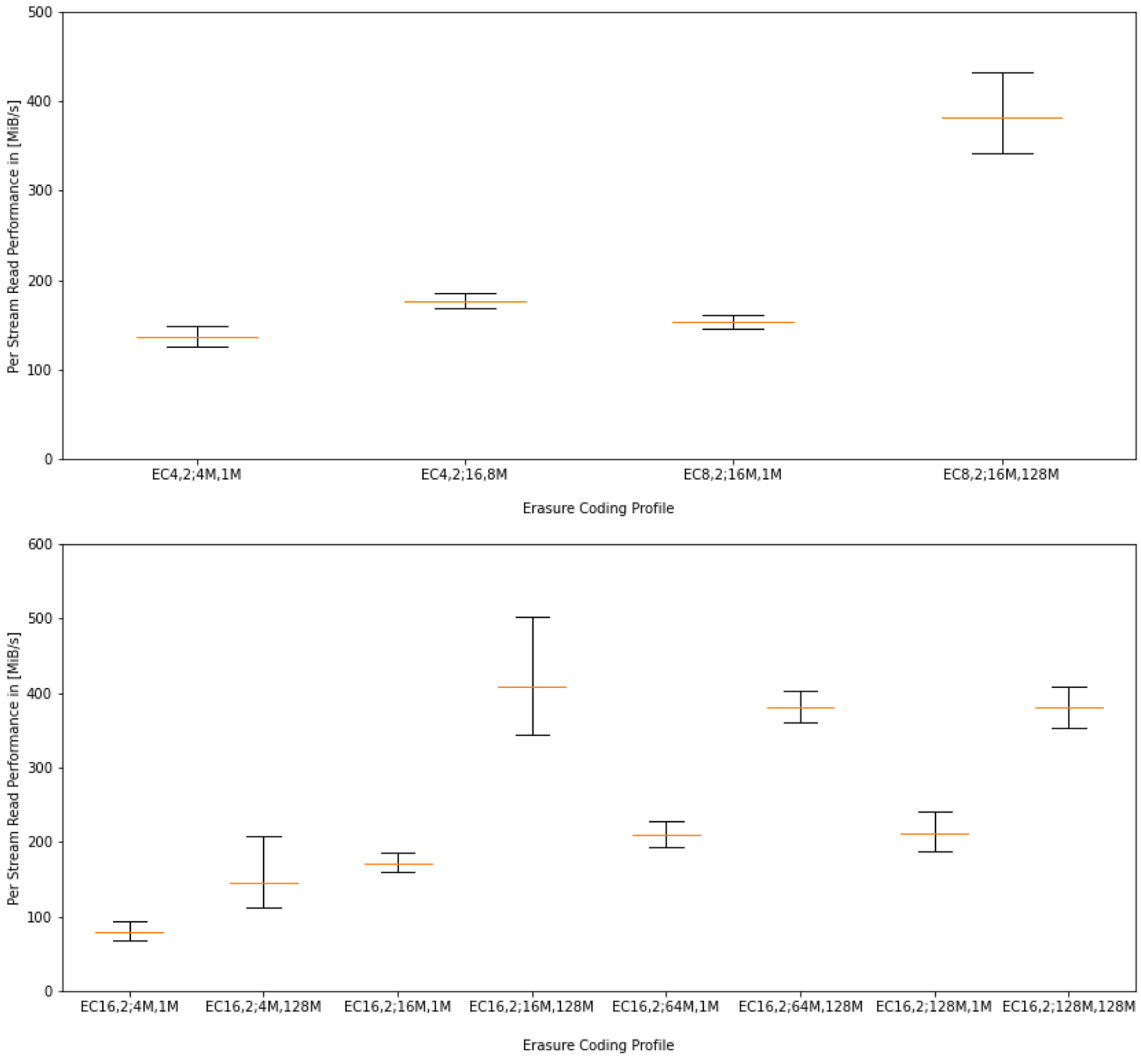

Fig. 10 Average read stream speed with standard deviation for various erasure coding layouts based on data from Table 2

Table 3 Comparison of native CephFS backend performance and access via the EOS frontend service, for various erasure coding layouts (ECk,m), object sizes (;*s*M) and IO blocksize (,*b*M)

–	Avg [s]	Sigma [s]	Rate [MiB/s]	99 perc. [s]	Max [s]
wr CephFS EC4,2;4M,1M	6.26	1.30	319	8.95	11.07
wr EOS EC4,2;4M	6.13	4.96	326	26.67	47.10
wr EOS[a] EC4,2;16M,8M	6.61	0.65	302	10.43	15.03
wr EOS[b] EC4,2;16M,8M	6.33	1.23	315	13.11	20.34
rd CephFS EC4,2;4M,1M	14.70	1.24	136	17.37	19.96
rd CephFS EC4,2;16M,8M	11.25	0.54	177	12.61	13.87
rd EOS EC4,2;4M,8M	10.45	1.25	191	13.26	14.5

With EOS[a] and EOS[b] we throttle the clients to 325 MiB/s and 350 MiB/s, respectively

disks were saturated at these peak throughputs. Bottlenecks for reading are more difficult to disentangle; they are most likely dominated by disk platter seek latencies. The CephFS erasure coding IO model roughly doubles the traffic for reading and writing. During write tests with large blocksizes the network input on a single node reaches an impressive 9 GiB/s, while the outgoing traffic is 5 GiB/s and the disk output is 5 GiB/s. To make use of the total available disk IO bandwidth of each server (10 GiB/s), one would have to double the network connectivity. In

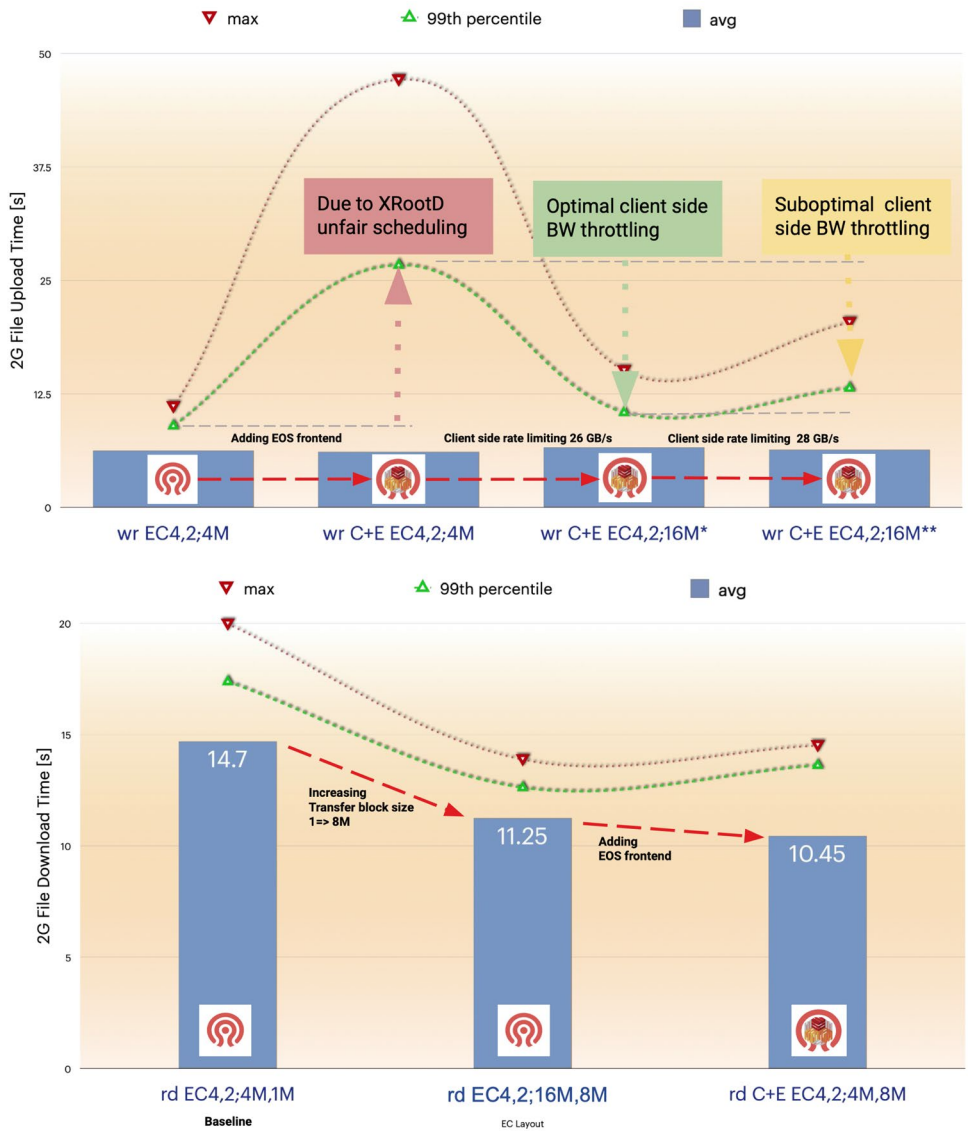

Fig. 11 Visualization of impact of adding EOS frontend to CephFS backend based on data from Table 3

addition, to the reported results we have also investigated concurrent read and write use cases. CephFS prioritizes the available bandwidth to writers, leaving readers with the remaining bandwidth; writer-preferred IO scheduling is indeed the ideal behaviour for most use cases.

The EOS frontend has only a marginal impact on the overall IO performance. The increased tails when writing should be investigated with respect to the unbalanced stream scheduling implementation inside the XRootD server.

In theory it would be possible to co-locate the EOS FSTs and Ceph OSDs on the same servers, however this would require mounting CephFS on the OSD nodes: such a mount is known to lead to a kernel deadlock if memory pressure occurs. The described hyperconverged storage/gateway model would require some extra testing under high-load situations.

The hybrid CephFS+EOS setup is a simple way to combine high performance parallel IO features of CephFS with the high-level functionalities provided by EOS. This includes strong security, efficient WAN access using XRootD and HTTP(S) protocol, extended quota and permission management, third party transfers, token authorization, checksum support, an optional tape backend and more.

When designing a hybrid service with 100Gig-E technology, particular attention has to be given to the network in the backend OSDs and given IO limits per frontend node. We managed to write at most 4.5 GiB/s and read 6 GiB/s with a single gateway FST with one CephFS kernel mount. In LHC storage usage at CERN we observe a typical 10:1 ratio of

read vs. write. Therefore, it could be an interesting option to add a redirect to local functionality to EOS in cases, where CephFS can be mounted with open access for reading on client nodes. The local redirect could be conditional on per-directory permission settings. The CephFS mount itself could also be triggered inside an XRootD plug-in based on a redirect response containing the cephx authentication key for the required CephFS read-only mount. When file access is sparse the gateway FST model provides an additional caching layer to convert client-side sparse access into streaming backend traffic. This requires appropriate tuning of the CephFS mount read-ahead settings.

The proposed service model allows to cluster several independent CephFS setups with independent failure domains and different quality of service behind a single administrative domain. It enables operators to carry out transparent hardware migrations between CephFS systems from an old to new backend using EOS management tools and third-party transfers without interrupting production usage.

We have validated this setup for IO streaming operations. Usability for sparse physics analysis use cases would be a next step for validation. In addition, an expected fragmentation penalty after aging through several filling/deletion cycles has not yet been evaluated. We demonstrated that CephFS can be used for high-throughput streaming IO without requiring dedicating SSDs for the BlueStore metadata `block.db`. The main requirement to operate large capacity server is to provide at least 3 GiB of memory per OSD (HDD).

In summary CephFS + EOS is a viable solution to combine the object storage concepts of Ceph and high-level service functionalities of EOS in a very simple way. One needs to balance the reasons of the additional complexity and service cost against benefits of such an approach.

Funding Open access funding provided by CERN (European Organization for Nuclear Research).

References

1. Albrecht J et al (2019) A roadmap for HEP software and computing R&D for the 2020s. Comput Softw Big Sci 3(1):1–49
2. Carlson M et al (2014) Software defined storage. Storage Networking Industry Assoc, San Francisco. https://www.snia.org/education/whitepapers
3. Dewhurst A et al (2017) The deployment of a large scale object store at the RAL Tier-1. J Phys Conf Ser 898(6):062051
4. Weil SA et al (2006) Ceph: a scalable, high-performance distributed file system. In: Proceedings of the 7th symposium on operating systems design and implementation, pp 307–320
5. Peters AJ, Sindrilaru EA, Adde G (2015) EOS as the present and future solution for data storage at CERN. J Phys Conf Ser 664(4):042042
6. QuarkDB—a highly available datastore. https://github.com/gbitzes/QuarkDB. Accessed 1 May 2021
7. Weil SA et al (2007) Rados: a scalable, reliable storage service for petabyte-scale storage clusters. In: Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07, pp 35–44
8. Weil SA et al (2006) CRUSH: controlled, scalable, decentralized placement of replicated data. In: SC'06: Proceedings of the 2006 ACM/IEEE conference on supercomputing, pp 31–31
9. Yoo Andy B, Jette Morris A, Grondona Mark (2003) Slurm: simple linux utility for resource management. In: Workshop on job scheduling strategies for parallel processing, pp 44–60
10. Sefraoui O, Aissaoui M, Eleuldj M (2012) OpenStack: toward an open-source solution for cloud computing. Int J Comput Appl 55(3):38–42
11. Withers A et al (2018) SciTokens: capability-based secure access to remote scientific data. In: Proceedings of the practice and experience on advanced research computing, pp 1–8
12. Aguado CA et al (2020) CERN's identity and access management: a journey to open source. In: EPJ Web of Conferences, vol. 245, p 03012
13. Dorigo A et al (2005) XROOTD-A highly scalable architecture for data access. WSEAS Trans Comput 1(4.3):348–353
14. Github Ceph Pull Request: mds: update defaults for recall configs. https://github.com/ceph/ceph/pull/38574. Accessed 1 May 2021