XrdEc: the whole story

Michal Simon

Introduction

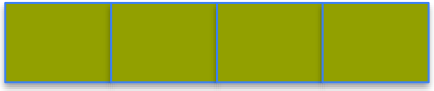
- XrdEc a high performance scalable EC-based file storage module motivated by the ALICE O2 use case.

- Originally developed for EOS and afterwards extended to work with any type of XRootD backend storage

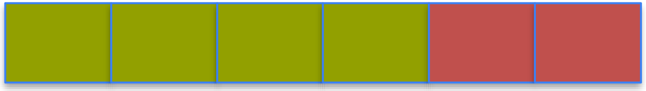
Writing

- Client buffers the data until it has a full block

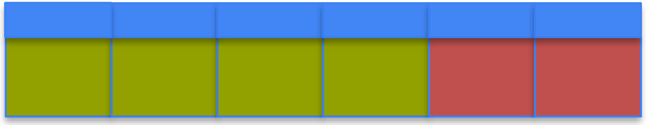
- The block is divided into chunks



- The chunks are erasure coded (Intel ISAL Reed-Solomon)

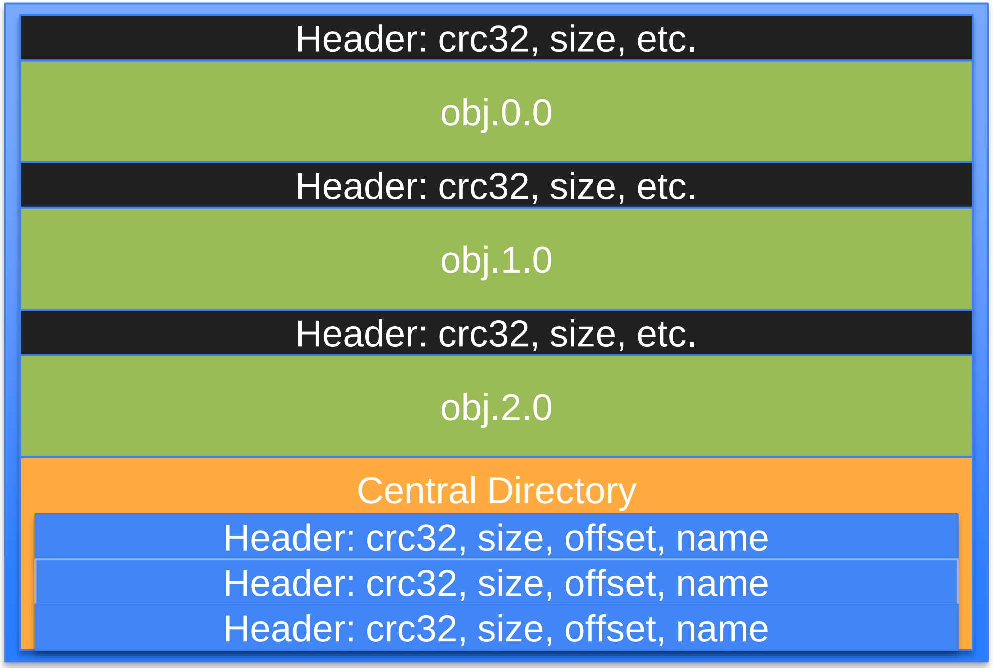


- All chunks (data/parity) are checksumed (h/w assisted CRC32C)



Writing

- Each stripe is stored in a ZIP archive, each chunk is a separate file within the archive



Header: crc32, size, etc.

obj.0.0

Header: crc32, size, etc.

obj.1.0

Header: crc32, size, etc.

obj.2.0

Central Directory

Header: crc32, size, offset, name

Header: crc32, size, offset, name

Header: crc32, size, offset, name

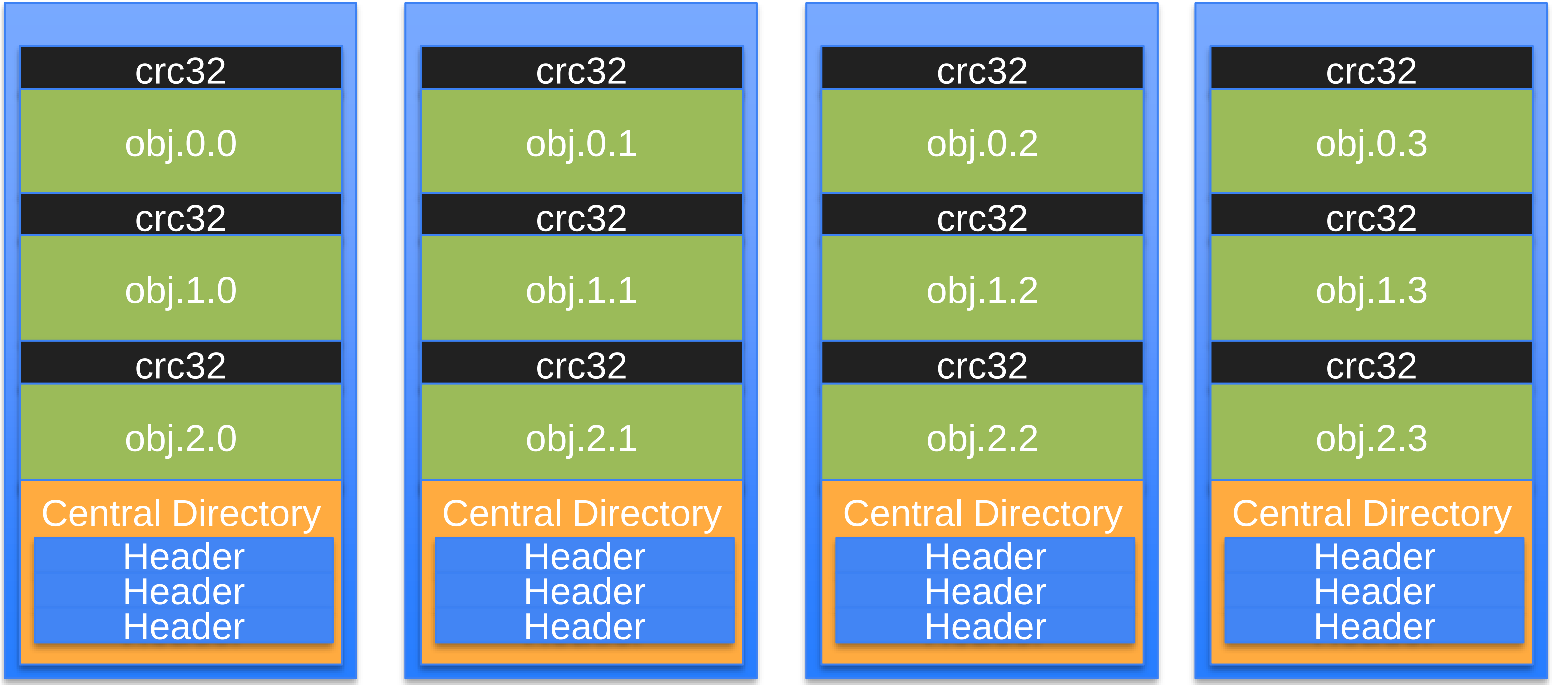
Block: 0, Stripe: 0

Block: 1, Stripe: 0

Block: 2, Stripe: 0

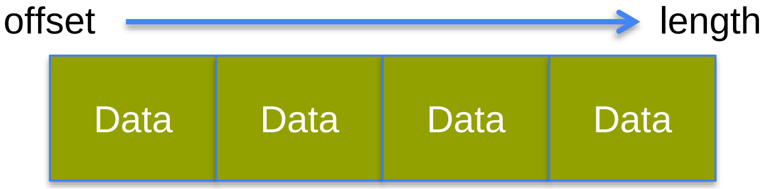
Writing

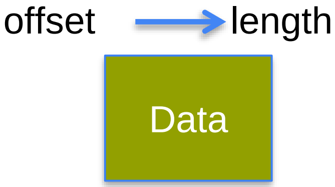
- Each stripe is stored in a ZIP archive, each chunk is a separate file within the archive



Reading

- There is **no need to reconstruct a block** for every read
  - Unless the client needs to do error correction
  - While streaming the data user can benefit from full performance boost due to striping
- In order to verify the checksum the client at minimum needs to read a whole chunk
  - **Reads are translated into respective chunks**
  - **Chunks are cached** until user is accessing data within same block





Loading the EC plug-in

- On demand by the server
  - The server can send a **special redirect response that will trigger loading the plug-in** for given file
  - The response contains: **number of data and parity chunks, block size, placement group** (data servers hosting the stripes), additional cgi to be send to data servers

- Standard client plug-in **config file**
  - EC layout, checksum type, etc.
  - One think we cannot preconfigure is the placement group, in this case we have to obtain it during runtime using **deep locate**

Operation support

- **Open modes:** Write + New or Read

- **Natively** in the EC module
  - Open, Close, **Read**, **VectorRead**, Write

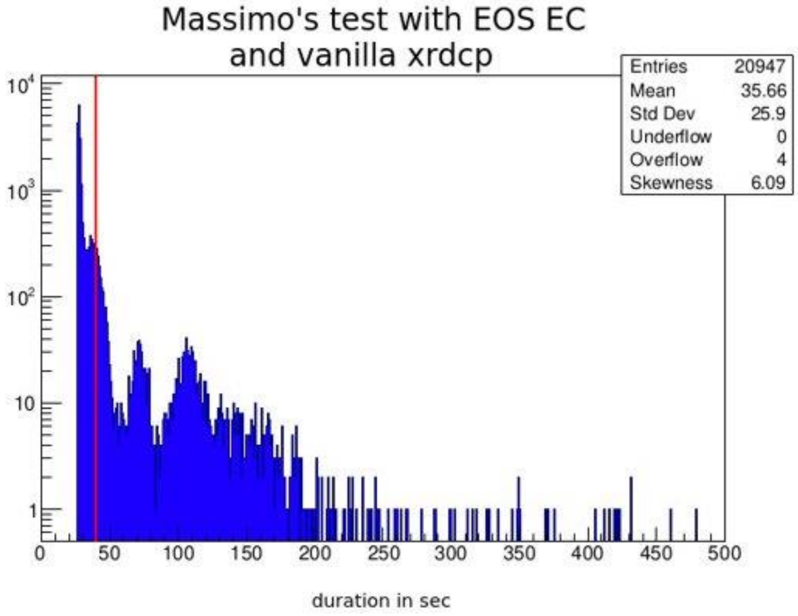
- In addition in the plug-in
  - **PgRead**, **PgWrite**, Stat

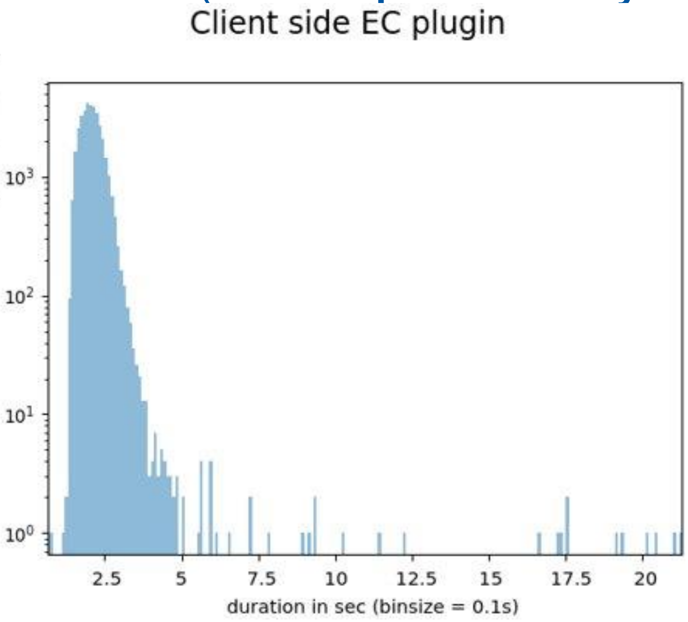
Use Case: Alice O2

- 500 EPNs (Event Processing Node), each hosting 4 GPUs, each GPU generating a Time Frame every 40 seconds
  - **2000 data sources** in total
  - Aggregate throughput of **100GB/s**
- A Time Frame (TF) corresponds to a single 2GB file in EOS
  - **TF has to be copied to EOS in less than 40 seconds**
- Data sources transfer data directly to EOS (CERN CC) in (kind of) round robin fashion at 20 ms intervals
  - **every 20 ms a new file will be created and 2GB of data transferred**

Use Case: Alice O2

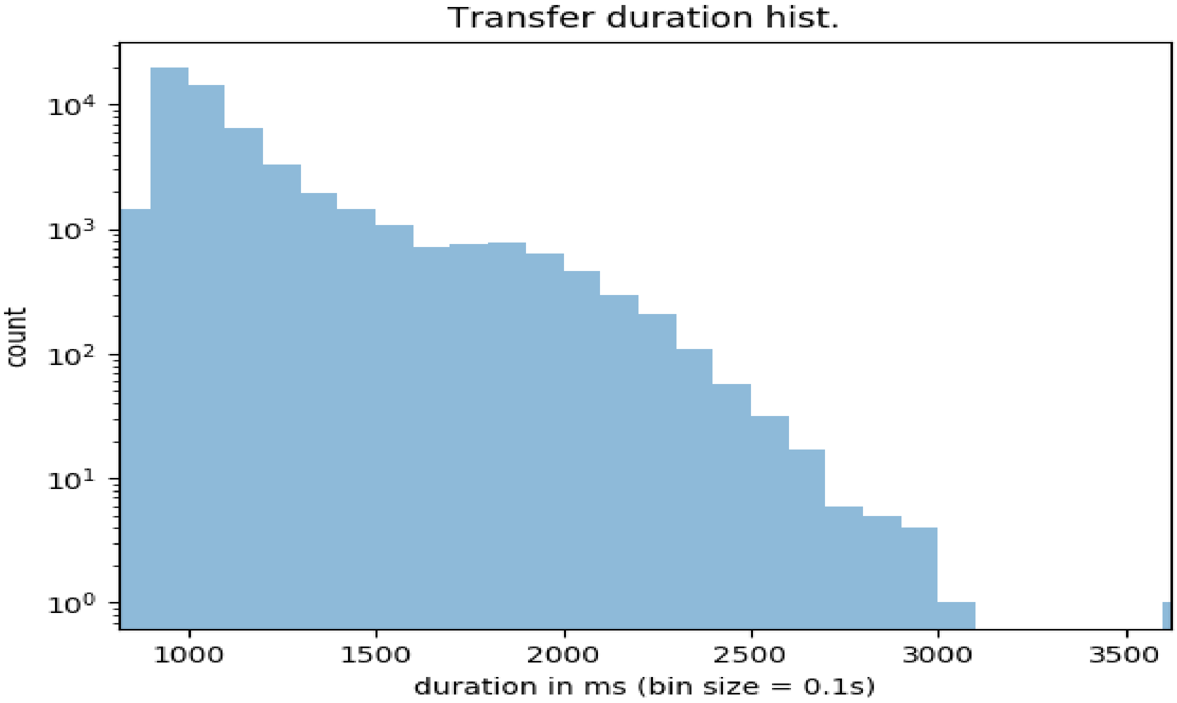
- Massimo’s test: clients run on the batch farm (**20% of the target load**), data recorded on EOSALICEO2 cluster (10 servers, **~20% of production system**)
- Client side EC plugin test: EPN simulator (4 AliceO2 servers) generating **~10% of the target load**, data recorded on 6 EOSALICEO2 servers (**~10% of production system**)





Use Case: Alice O2

~30% of the target production load, ~10% of the cluster capacity



10+2 layout,
30GB/s of aggregate throughput (600 streams),
1 hour run, 6 data servers

Avg duration: 1127msec
Avg transfer rate: 1.84GB/s
Transfer rate stdev: 0.317
Transfer duration stdev: 272

Integrating XrdCl+EC with the xrootd storage

1. Mode 1. Use xrootd storage directly as an EC store
    - Xroot protocol and xrootd client (with EC support) only

This mode is good for local administration

2. Mode 2. Use XRootD Proxy as gateway to backend storage
    - Enable EC in the proxy’s xrootd client component.
    - EC is invisible to the users
        - They use existing xrdcp/xrdfs, gfal, curl
    - Support all WLCG security, protocols, TPC, etc.
    - The backend xrootd storage is plain and simple

This mode is better for user access

- The rest of the slides are about this mode

Interface to users

Nothing changed: users will still work with root(s) or http(s) URL:

- https://atlas.cern.ch:1094/atlas/rucio/user/jdoe/my.data or
- root://atlas.cern.ch:1094**//**atlas/rucio/user/jdoe/my.data
- Think of “atlas/rucio/user/jdoe” as bucket, folder, whatever you like.
  - Your access permission may be based on top level buckets/folders.

Three sets of tools for GET/PUT/DEL/LIST/RENAME

- **xrdcp/xrdfs**: work mostly with root(s) URLs
- **gfal2**: works with both root(s) URL and http(s) URLs
- **curl**: works with http(s) URLs

Supports DTN functionality

- **Authentication**, **VOMS**, **access tokens**, **TPC**, etc.

Performance test environment

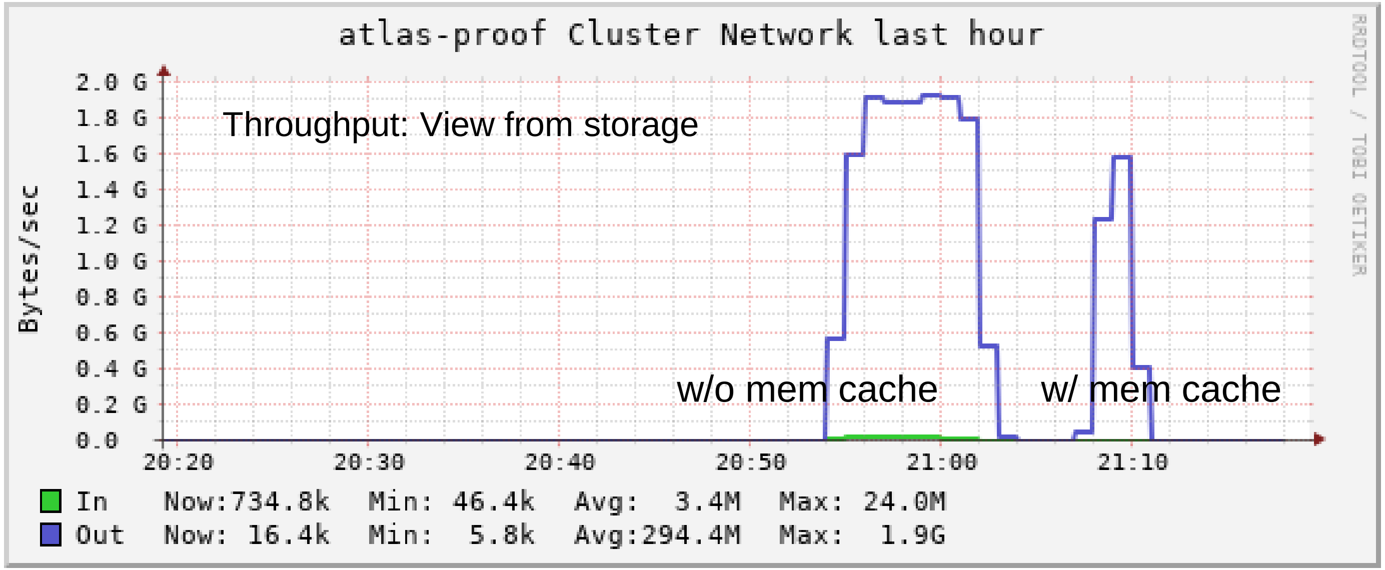
Backend: XRootD storage:

- 19 nodes of retired Dell R510s, each:
  - 24GB RAM, 1Gpbs NIC, 12x 3TB HDD (some have 11)
  - Each HDD is presented to the OS as its own SCSI device (via LSI RAID controller)
  - CentOS 7, XRootD 5.3.4 (later auto-updated to 5.4.0), xrootd “sss” security
- 312 pre-placed test files (ATLAS data files) ranging from 30MB to 1.1GB, all with known adler32 checksum

Frontend: XRootD EC proxy

- 64 core, 128GB, 100Gbps NIC
- CentOS 7, unreleased XRootD (2021-12-17+patch)
- EC configuration: **8+2**, chunk size 1MB (so a block has 8+2 MB)

Aggregate read performance by many clients



Throughput: View from storage

w/o mem cache

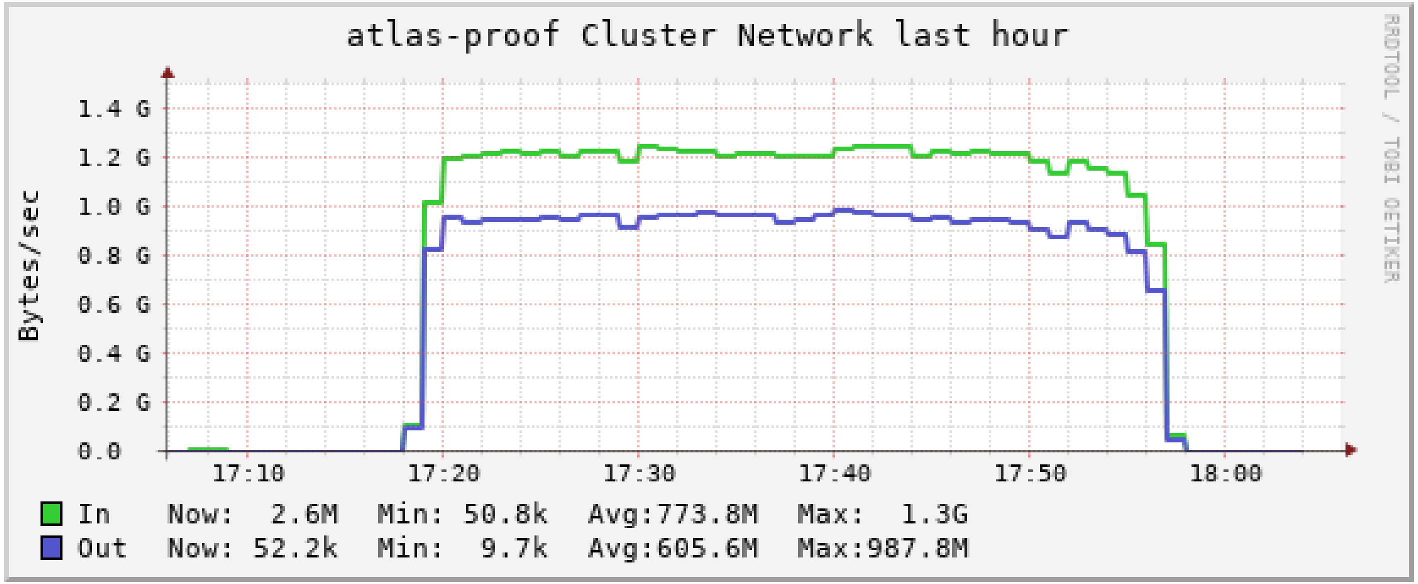
w/ mem cache

Network upper limit

- 19 Gbit/s or
- 2.375GB/s

- Read the pre-placed 312 data files, repeat 5 times
- Spread the read to 150 concurrent clients
- Memory cache clearly helped, it both
    - cache (reduce read from storage)
    - enable large block read (align with EC blocks)

Aggregated Read/Write performance



Backend storage view

- In: write
- Out: read

Memory cache: off

- By 200 concurrent clients
- Randomly pick 20 files from the 312 sample files
- Read and write back at the same time
  - Note: FS prioritizes write over read

Summary

- XrdEc is a very performant implementation, we run almost at h/w speed (Intel ISAL, h/w assisted CRC32C)
- The tests at AliceO2 and SLAC yield very good results
- We started work on ops tools this summer (using student workforce), which resulted in a xrdrepair tool