



Michal Simon

High throughput erasure coding with XRootD client

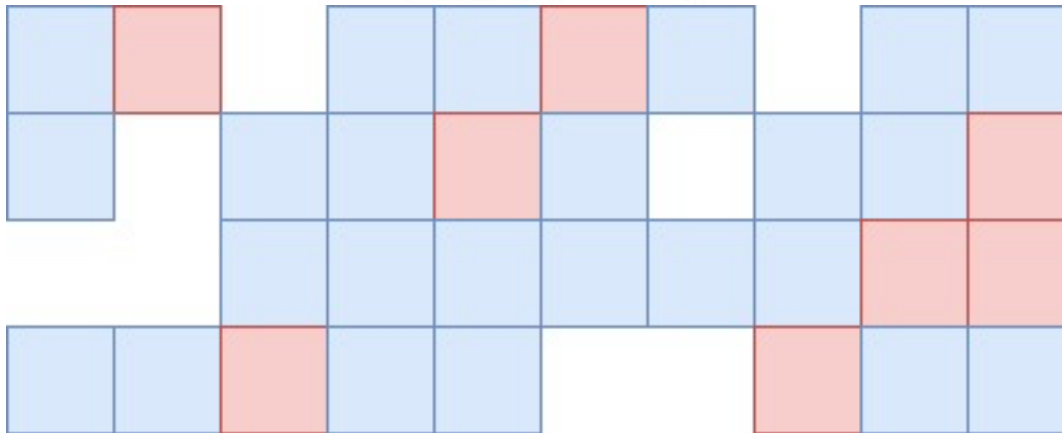


Outline

- Design
- Write tests on AliceO2 cluster
- Further developments

XrdEc design

- For better performance uses **only asynchronous APIs**
- Intel **ISAL** (Intel Storage Acceleration Library) – **state of the art implementation of Reed Solomon codes**
- Hardware assisted **CRC32C**
- Distribute chunks uniformly including spare locations (e.g. 6 data + 2 parity + 2 spare)

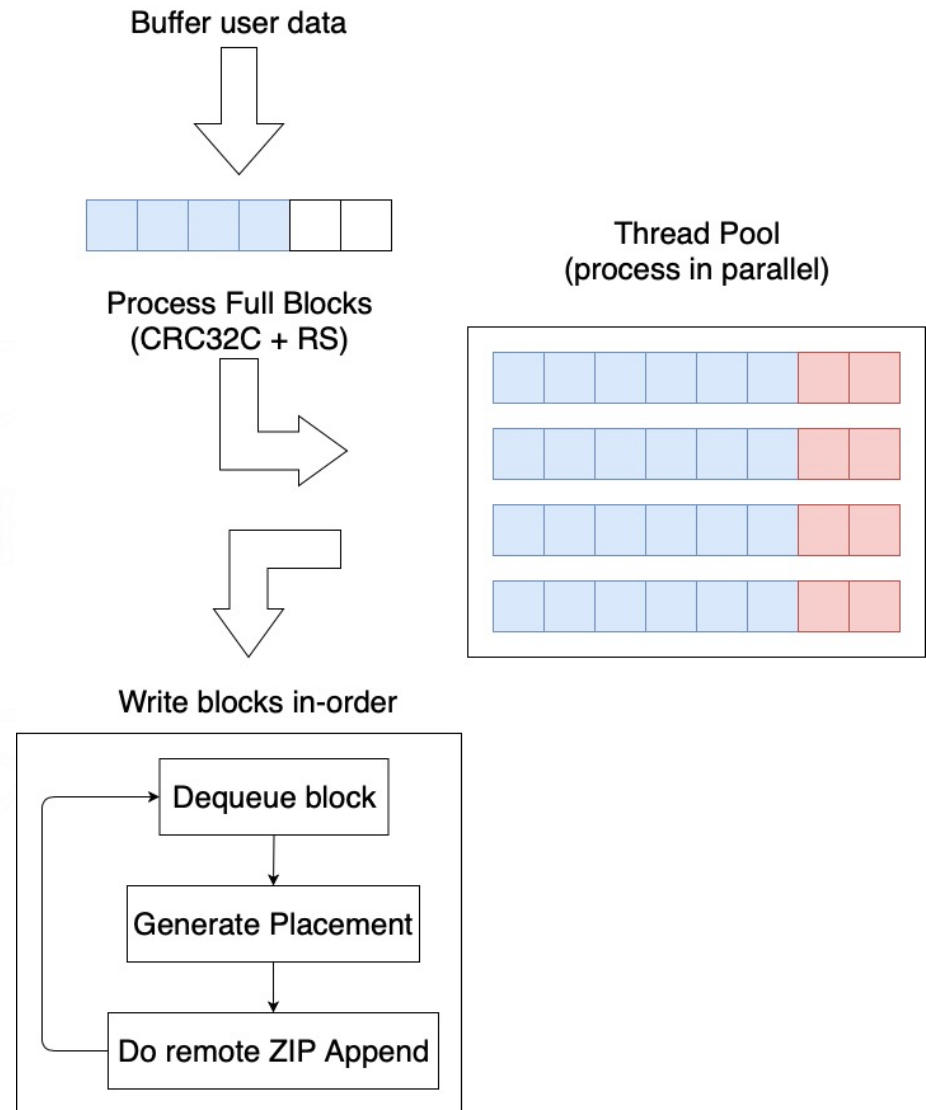


XrdEc design

- At every location **two files are created**: *objid.data.zip* and *objid.metadata.zip*
 - *objid.data.zip*: contains the raw data; **each data chunk stored in a separate file; vanilla ZIP archive except for the checksum** (we use CRC32C instead of CRC32)
 - *objid.metadata.zip*: contains the metadata for each *objid.data.zip* file, **replicated over all locations** (at least #parity + 1)

XrdEc design

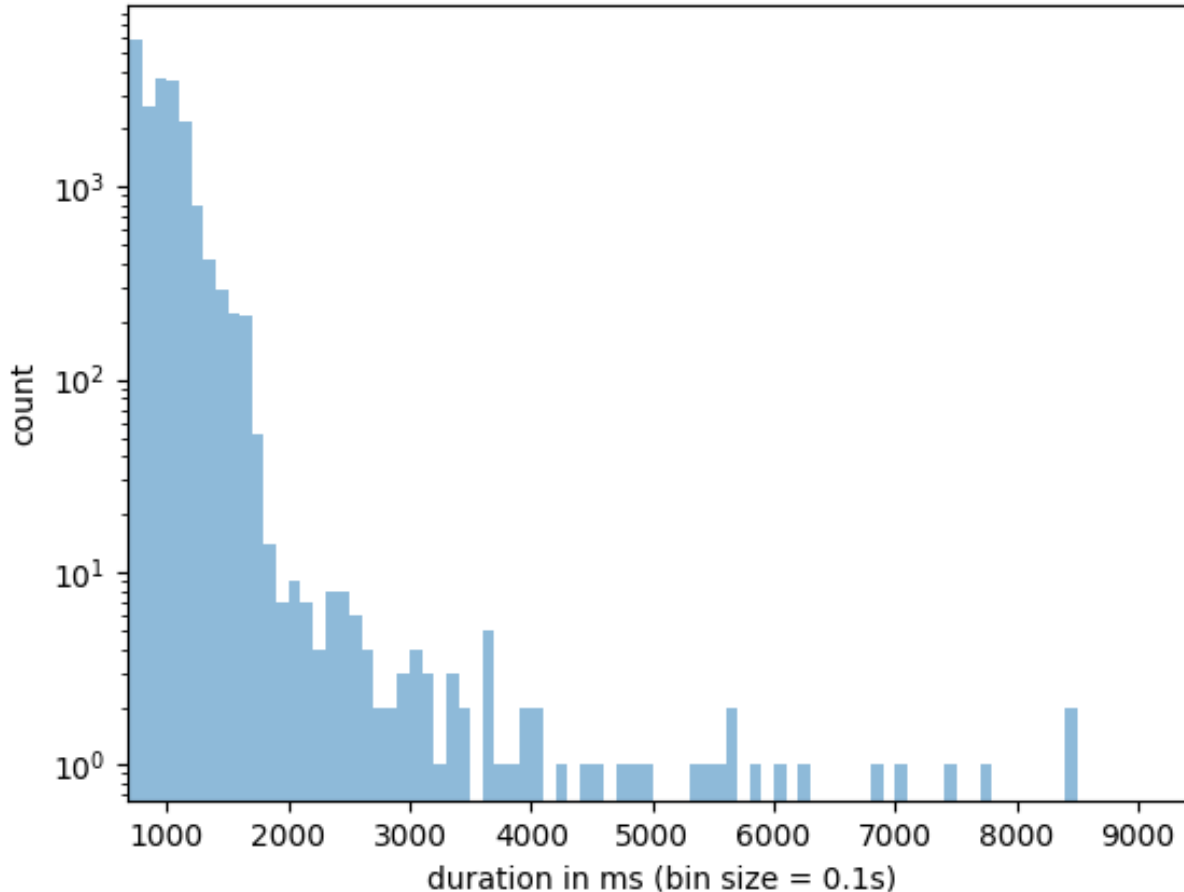
- User writes are buffered into full blocks
- The blocks are **erasure coded and checksummed in a thread-pool** (in parallel)
- Ready blocks are **written in-order** into the destination servers



AliceO2 write tests

~10% of the target production system, ~10% of the cluster capacity

Transfer duration hist.



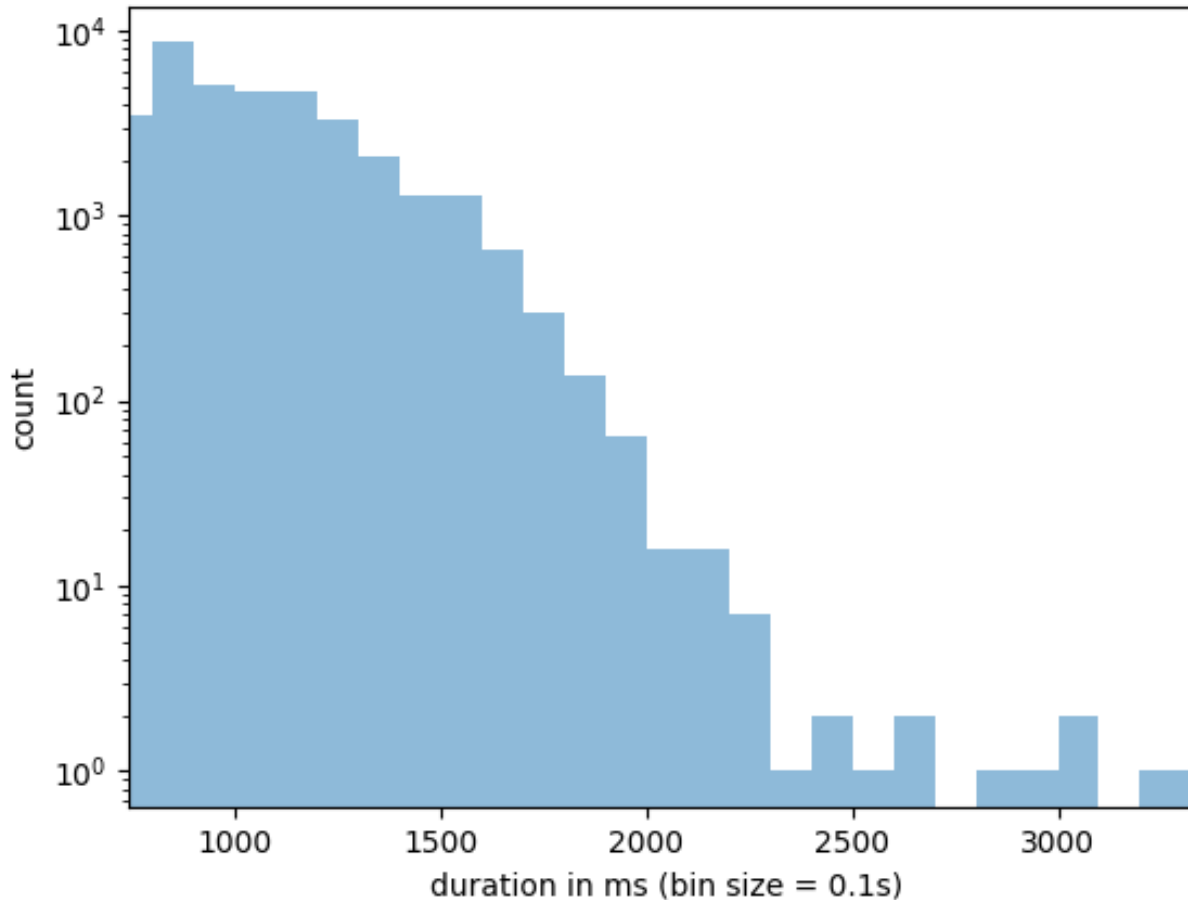
10GB/s of aggregate throughput (200 streams),
1 hour run, 6 data servers

Avg duration: 974 msec
Avg transfer rate: 2.15GB/s
Transfer rate stdev: 0.418
Transfer duration stdev: 290

AliceO2 write tests

~20% of the target production system, ~10% of the cluster capacity

Transfer duration hist.



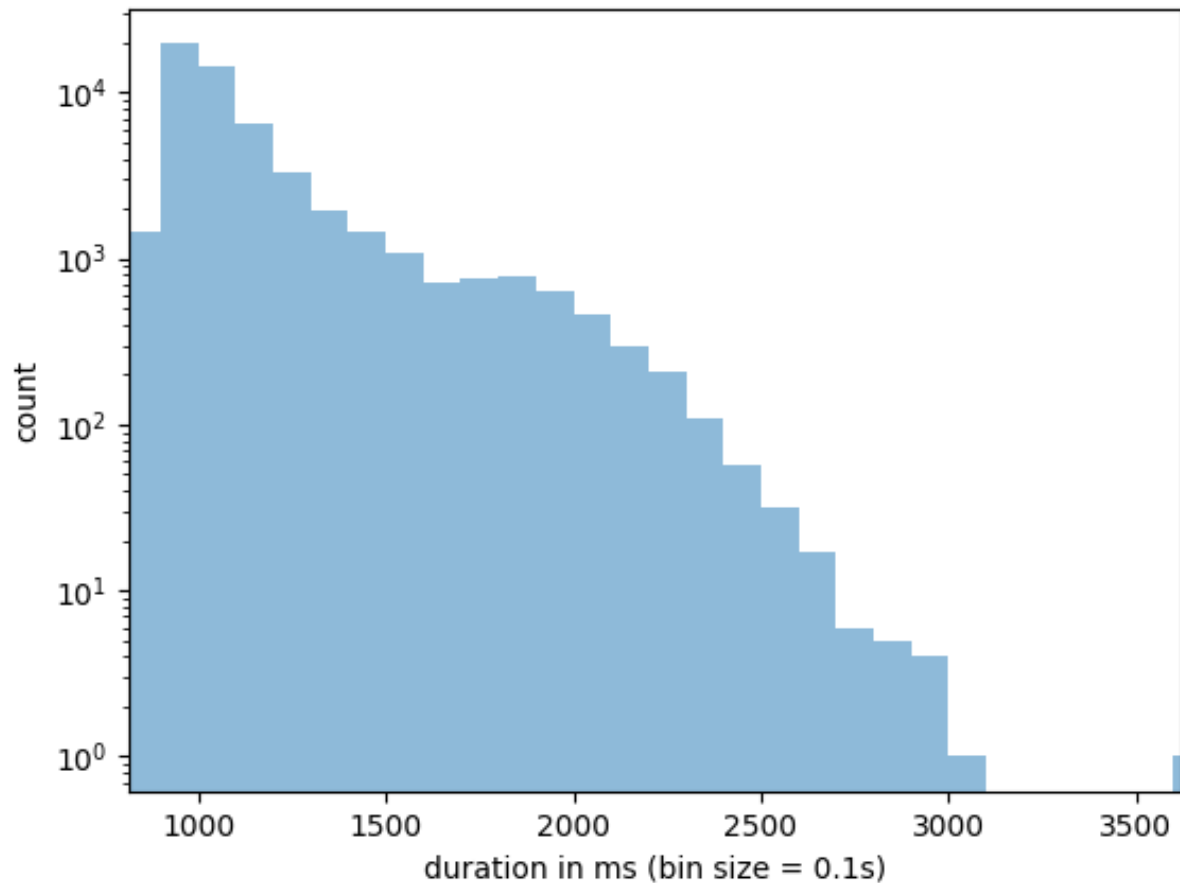
20GB/s of aggregate throughput (400 streams), 1 hour run, 6 data servers

Avg duration: 1063 msec
Avg transfer rate: 1.97GB/s
Transfer rate stdev: 0.400
Transfer duration stdev: 244

AliceO2 write tests

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

Transfer duration hist.



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

Plans

- **Test read performance**
- **Provide XRootD client plug-in**
- Adopt PgWrite (once implemented) for transportation
 - Minimal overhead (reuse the checksums when appending new blocks)
- If user buffer is block aligned optimize out copying data to internal buffer

Plans

The update problem

- Even if a **single byte has been changed we need to update several files** on several remote devices
 - The file containing the raw data and all the parities
- The update has to be done in a **atomic** way
 - Either it is **successful and all stripes have been updated** or it **failed and non of the stripes have been updated**
- Difficult to implement: consider user doing Ctrl+C during the update

Plans

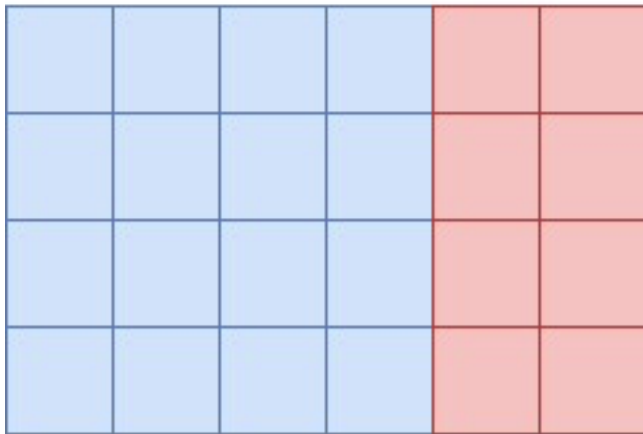
Update: range cloning to the rescue

- What can we do?
 - Get **new object ID** from MGM
 - In memory **update the block and recalculate the parities**
 - Create a new sparse file with the updated block
 - **Clone the remaining part of the file**
 - Newer file systems (XFS, Cenots 8) support sharing physical storage between multiple files (`ioctl_ficlone`)
 - Commit new version

Plans

Update: range cloning to the rescue

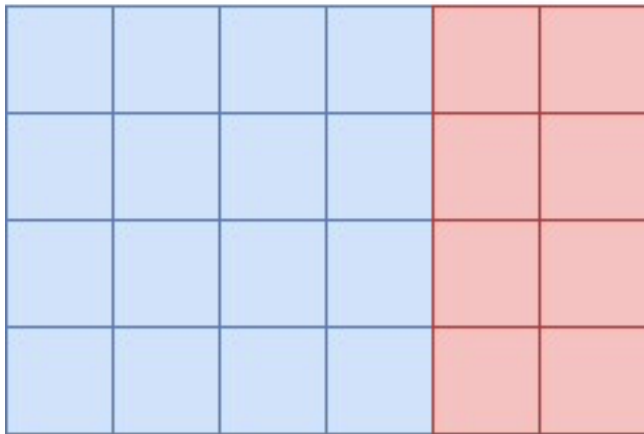
Original file



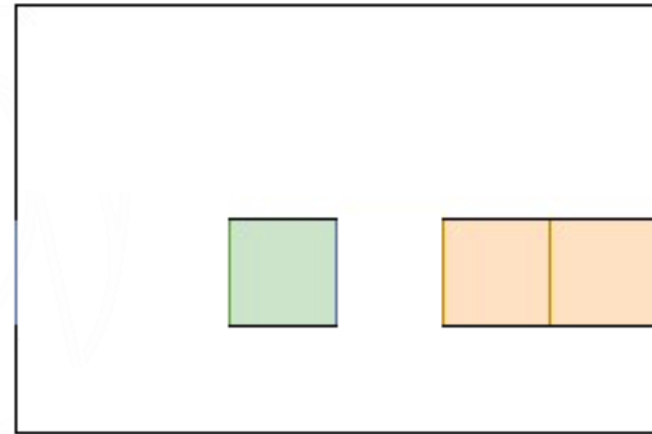
Plans

Update: range cloning to the rescue

Original file



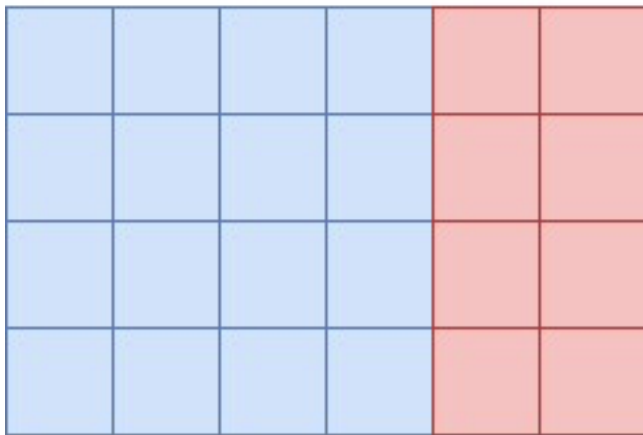
One updated block



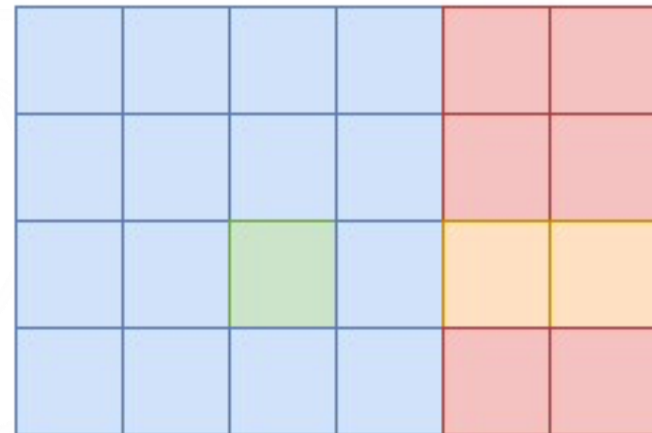
Plans

Update: range cloning to the rescue

Original file



Updated file



The blue blocks are shared between the two versions of the file

Questions?

