Evaluation of software based redundancy algorithms for the EOS storage system at CERN

Andreas-Joachim Peters
CERN IT, Geneva, Switzerland
E-mail: andreas.joachim.peters@cern.ch

Elvin Alin Sindrilaru
CERN IT, Geneva, Switzerland
E-mail: elvin.sindrilaru@cern.ch

Philipp Zigann
CERN IT, Geneva, Switzerland
E-mail: philipp.zigann@cern.ch

Abstract. EOS is a new disk based storage system used in production at CERN since autumn 2011. It is implemented using the plug-in architecture of the XRootD software framework and allows remote file access via XRootD protocol or POSIX-like file access via FUSE mounting. EOS was designed to fulfill specific requirements of disk storage scalability and IO scheduling performance for LHC analysis use cases. This is achieved by following a strategy of decoupling disk and tape storage as individual storage systems. A key point of the EOS design is to provide high availability and redundancy of files via a software implementation which uses disk-only storage systems without hardware RAID arrays. All this is aimed at reducing the overall cost of the system and also simplifying the operational procedures.

This paper presents the advantages and disadvantages of redundancy by hardware (most classical storage installations) in comparison to redundancy by software. The latter is implemented in the EOS system and achieves its goal by spawning data and parity stripes via remote file access over nodes. The gain in redundancy and reliability comes with a trade-off in the following areas:

- Increased complexity of the network connectivity
- CPU intensive parity computations during file creation and recovery
- Performance loss through remote disk coupling

An evaluation and performance figures of several redundancy algorithms are presented for dual parity RAID and Reed-Solomon codecs. Moreover, the characteristics and applicability of these algorithms are discussed in the context of reliable data storage systems.

1. Introduction

1.1. The EOS Storage System at CERN

The EOS project [1] [2] started in April 2010 at CERN in the IT-DSS (Data and Storage Services) group. The main design target is to provide cost-effective and large disk-only storage space with configurable reliability for analysis use cases with low-latency data access for LHC experiments.
The current production system which is divided into three instances corresponding to the ALICE, ATLAS and CMS experiments is managing 9100 JBOD disks and 16 PB of raw disk space in April 2012. All files are currently stored with two replicas bisecting the logical space. Copies are created synchronously on different nodes to provide ‘redundancy on close’. Redundancy over nodes covers not only disk failures as in traditional RAID storage systems but also controller, node and partially network failures. Redundancy algorithms such as Reed-Solomon codes allow improvements in both space efficiency and reliability. They are characterized by redundancy parameters \((m, k)\), \(m\) indicating the number of data disks, \(k\) the parity disks. Applying these algorithms on disks distributed over several nodes enables scaling of bandwidth to data and redundancy at the same time.

### 1.2. RAID vs RAIN Approach

The two approaches explained in the following section differ mainly in the location of disks in the so-called failure groups used to provide redundancy. RAID\(^1\) disks are attached to the same node and disk controller. RAIN\(^2\) disks are attached to individual nodes and communication is done over the network.

#### 1.2.1. Redundant Array of Independent Disks

RAID technology is used to protect storage filesystems against failures of single or multiple disks depending on the RAID configuration. A widely used standard is RAID-6 allowing for double disk failures without data loss. A rebuild of a failed disk involves all disks within an array. However, the rebuild time increases with the

\(^{1}\) Redundant Array of Independent Disks

\(^{2}\) Redundant Array of Inexpensive Nodes
number of disks in the failure group. The EOS storage at CERN does not configure RAID arrays but JBODs\(^3\) on storage nodes.

1.2.2. Redundant Array of Inexpensive Nodes - RAIN technology is used to protect storage systems against failures of single or multiple disk and server nodes. Redundancy is applied at block or file level. Files get split into equal blocks which are distributed to the nodes and additional parity blocks are computed. All blocks making up a file are stored within the same failure group. Increasing the size of node failure groups allows scaling the rebuild performance independent of the redundancy parameters \((m, k)\). A further advantage is that block-level checksumming allows recovery of bit-level corruptions - not only full device failures. For these reasons EOS combines RAIN technology checksumming on block level. Figure 3 illustrates the block distribution of a file stored with a \((4, 2)\) RAIN schema.

2. Implementation

2.1. RAIN Algorithms

EOS provides so-called file layouts. These describe how files are structured on the storage system. The most basic type is a *plain* file. The current layout used in production is a *replica* layout doing synchronous replication on write. The client-server framework used in EOS is based on XRootD [4]. The XRootD server provides a plug-in mechanism used for the implementation of file and directory IO interfaces. The EOS file layout plug-in can be attached to the XRootD file IO plugin on the server side with one server acting as a gateway to the distributed file fragments. Otherwise, it can be attached on the client side using parallel IO to data and redundancy stripes. The first scenario does not require changes on the client side. A drawback is that the IO bandwidth is limited by the network of a single storage node.

We have implemented two RAIN layout drivers:

- RaidDP - a dual parity algorithm based layout driver - fixed \((m = 4, k = 2)\)
- ReedS - a reed-solomon code based layout driver - variable \((m, k)\)

The EOS RAIN layout drivers split files into 1 MB blocks and distribute them over \(m\) stripes on \(m\) disks/nodes. \(k\) parity stripes are computed and stored on additional disks/nodes. Each stripe is stored together with a block checksum file which contains checksums of all the blocks stored in

\(^3\) *Just a Bunch Of Disks* \(^7\)
that stripe. For performance reasons, the block checksum file is memory mapped on the storage server when the stripe is opened. Each stripe file begins with a header describing the stripe contents, block size etc. (see figure 3). The RaidDP algorithm has been implemented in EOS and is based on XOR operations as explained here [5]. The ReedS algorithms are provided by the ZFEC library [6]. No particular performance tuning of the algorithms has been undertaken for the following measurements.

2.2. Checksumming Algorithms
EOS supports five checksum algorithms which are usable for file and block checksumming:

- ADLER32 [7]
- CRC32 [7]
- MD5 [8]
- SHA1 [9]
- CRC32C [10]

ADLER32 and CRC32 are software algorithms provided by zlib [7]. MD5 and SHA1 are software algorithms provided by the OpenSSL library [11]. CRC32C is similar to CRC32 but uses a different polynomial. It is a hardware based algorithm using the SSE4.2 extensions of Intel CPUs [12]. The code for CRC32C has been published by MIT based on Intel’s Slicing by 8 project [13].

3. Performance Measurements
3.1. Performance Checksumming
Figure 5 shows a performance test on an 8-core Xeon 2.27 GHz with 12x4 GB DDR3/1GHz RAM with block sizes from 4 KB to 4 MB. The hardware CRC32C implementation performs close to 2 GB/s for all block sizes. The ADLER32 algorithm reaches identical performance for

![IO Scheme](image)
block sizes over 1 MB. This is an artifact of an additional feature in the EOS implementation which allows to compute ADLER32 file checksums for files which are not ordered by file offset. The implementation keeps a map with all ADLER32 checksums per block in memory which reduces the performance for small block sizes\(^4\). CRC32, MD5 and SHA1 checksum performance is independent of the block size but significantly slower with 720 MB/s, 440 MB/s and 295 MB/s.

The algorithm scalability by core is shown in figure 6. The behavior is linear allowing to reach the memory bandwidth limit of the DDR3 RAM with 16 GB/s for CRC32C checksumming. Given the previous measurements, it comes as a natural consequence that the ideal choice for block checksumming is the hardware based CRC32C algorithm. All new disk server models installed at CERN are shipped with an Intel CPU providing the required SSE4.2 extension.

3.2. RAIN Performance

The RAIN layout support has not yet been fully integrated into EOS. For performance measurements we used a stand-alone file copy tool eoscp. It allows to create and reassemble RAIN file fragments for upload and download using one of the two layouts: RaidDP or ReedS (see figure 7).

![eoscp](image)

Figure 7. eoscp

In order to make a better estimate of the algorithmic performance and implementation overhead, performance tests have been done against the XRootD server running on the same node as the eoscp command. There is a certain performance repression due to the fact that the client and server run on the same node.

Figure 8 (top) shows the file upload performance for RaidDP and ReedS with four data and two

\(^4\) Map lookup times increase with the number of blocks.
parity stripes. The single client performance for RaidDP is 380 MB/s, 212 MB/s for ReedS (4, 2). Both algorithms reach a maximum throughput of 840 MB/s for 6 concurrent clients.

Figure 8 (bottom) shows the single client performance of ReedS layouts varying the data and parity stripe numbers. The performance is inversely proportional to the number of (parity-) stripes in use. Nevertheless - for a (10, 6) configuration the algorithm would nearly saturate a 1 GBit network.

Figure 9 shows a final RaidDP and ReedS benchmark with (4, 2) configuration where the XRootD server was running remotely with 1 GBit connectivity to the client node. The read performance without recovery reaches 110 MB/s, write performance 78 MB/s and read recovery with two-stripe recovery at 106 MB/s for both algorithms. The penalty for writes is due to the not fully asynchronous implementation of the XRootD client in case of multiple destinations. A rewrite of the XRootD client is under way and will hopefully remove the observed performance loss.

4. Summary and Outlook
RAIN technology in combination with CRC32C checksumming is a very powerful technology to build resilient storage systems on. Single client/core performance exceeds 1 GBit and is sufficient for write performance requirements at CERN for LHC use cases. Besides the described extra resilience features, clients will be able to gain by the bandwidth-per-file and IOPS-per-file boost through node-wide striping - in particular for random IO patterns. The most simple and currently most performant RaidDP (4, 2) layout would bring a disk space gain of 33%. One of the shortcomings of these advanced redundancy algorithms is the increased IO and computational
effort for non-sequential writes. Fortunately, the main use cases in LHC perform non-sequential writes on local disks with streaming uploads to EOS - minimizing these issues.

In summary - the described performance measurements encourage us to fully implement and integrate RAIN in the EOS system. For the beginning, RAIN will be implemented inside XRootD servers in gateway mode, with the possibility of being moved on the client side in the future. As a parallel activity, all the operational tools for consistency checking and recovery will be developed and tested for production usage.

References

[1] Exabyte Scale Storage at CERN, Andreas J Peters and Lukasz Janyst 2011 J. Phys.: Conf. Ser. 331 052015
[2] EOS - CERN disk storage system, http://eos.cern.ch
[3] LHC - The Large Hadron Collider, http://lhc.web.cern.ch/lhc/
[4] XRootD, http://xrootd.org
[5] RAID-DP - Network Appliance, Chris Lueth, Network Appliance, Inc. TR-3298 [12/2006], http://tarasistem.com/docs/NetApp/raiddp.pdf
[6] zfec library, https://tahoe-lafs.org/trac/zfec
[7] zlib http://zlib.net
[8] The MD5 Message-Digest Algorithm, http://tools.ietf.org/html/rfc1321
[9] Secure Hash Standard, National Institute of Standards and Technology. FIPS 180-1, May 1993, http://csrc.nist.gov
[10] Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits, Castagnoli, G., Braeuer, S. & Herrman, M., IEEE Trans. on Communications, Vol. 41, No. 6, June 1993
[11] OpenSSL - Cryptography and SSL/TLS Toolkit, http://openssl.org
[12] Fast, Parallelized CRC Computation Using the Nehalem CRC32 Instruction, Vinodh Gopal, Jim Guilford, Martin Dixon, Wajdi Feghali, April 12, 2011, http://drdobbs.com/cpp/229401411
[13] Silcing-by-8, http://sourceforge.net/projects/slicing-by-8/