Benchmarking Performance of Erasure Codes for Linux Filesystem EXT4, XFS and BTRFS

Shreya Bokare[1(✉)] and Sanjay S. Pawar[2]

[1] Principal Technical Officer, Centre for Development of Advanced Computing, Mumbai , India
shreya@cdac.in, http://www.cdac.in
[2] Usha Mittal Institute of Technology, SNDT Women's University, Mumbai , India
sanjay.pawar@umit.sndt.ac.in, http://www.umit.ac.in

Abstract. Over the past few years, erasure coding has been widely used as an efficient fault tolerance mechanism in distributed storage systems. There are various implementations of erasure coding available in the research community. Jerasure is one of the widely used open-source library in erasure coding. In this paper, we compared various implementations of Jerasure library in encoding and decoding scenario. Our goal is to compare codes with different filesystems data to understand its impact on code performance. The number of failure scenarios is evaluated to understand performance characteristics of Jerasure code implementation.

Keywords: Erasure coding · Distributed storage · Filesystem–XFS · BTRFS · EXT4 · Jerasure 2.0

1 Introduction

Erasure coding for storage-intensive applications is gaining importance as distributed storage systems are growing in size and complexity. Erasure coding is an advanced version of RAID systems in the factors like fault tolerance and lower storage overhead and the ability to scale in a distributed environment. This makes erasure codes superior to RAID systems and the most suitable for storage intensive applications [1,2]. There are several implementation libraries of erasure coding, namely liberasurecode [3], Jerasure [4], Zfec [5], LongHair [6], Intel ISA-L [7], etc. Jerasure is a widely used erasure coding library in various open-source software-defined distributed storages like Ceph [8]. Jerasure is one of the stable libraries that supports a horizontal mode of erasure codes, written in C/C++ and implements several variants of Reed–Solomon and maximum distance separable (MDS) erasure codes (Vandermonde, Cauchy, Blaum–Roth, RAID-6, Liberation). Jerasure implementation uses matrix-based coding with Galois field arithmetic. GF-Complete library, which has procedures of Galois Field arithmetic, is used in Jerasure 2.0.

C. R. Panigrahi et al. (eds.), *Progress in Advanced Computing and Intelligent Engineering*, Advances in Intelligent Systems and Computing 1198, https://doi.org/10.1007/978-981-15-6584-7_32

In traditional enterprise architecture having structured and unstructured data types, filesystem plays an important role providing support for scalability, extendibility and optimization with respect to the storage technology. Impact of the traditional and modern filesystems needs to be evaluated with erasure coding used in distributed storage systems. Therefore, primary motivation behind the work is to evaluate the working of traditional and modern filesystems with respect to different erasure coding implementations, which can help in selecting the best suitable combination for optimal data storage and access.

In this paper, encoding and decoding experiments on various code variants of Jerasure library (version 2.0 released in 2014) are performed. The rest of the paper is organized as follows. Section 2 provides various code implementations in Jerasure. Section 3 mentions about the impact of filesystem on data the reading and writing. Experimental setup is explained Sect. 4. Section 5 provides details of benchmarks and results. The paper is concluded in Sect. 6.

2 Jerasure Coding Library

The Jerasure library with the first release in 2007 and the next release Jerasure 2.0 in 2014 is one of the oldest and most popular erasure coding libraries [4]. Jerasure implements minimum distance separable (MDS) codes, where erasure coding is organized in the following manner for given dataset D. The data D is divided into k equal-sized information blocks. The k information blocks are then coded into n blocks of equal size. The n blocks consist of k information blocks and $m = (n - k)$ parity blocks. These n blocks are written into n distributed storage nodes. The k storage nodes holding data information are called data nodes, and those having parity information are called as parity nodes. In the case of failure of any m nodes, the lost data can be reconstructed using the remaining k nodes. Another important parameter in the erasure coding is a strip unit w, which is the word size. All the code views each device as having w bits worth of data. The $w \in \{8, 16, \text{ and } 32\}$ can be considered as collection of w bits to be a byte, short word or word, respectively. The matrix-based coding used in Jerasure is based on distribution (generator) matrix whose elements are calculated using Galois field arithmetic GF (2^w) for some value of w. Figure 1 shows the distribution matrix and calculation of data and parity blocks.

2.1 Reed–Solomon Code

Reed–Solomon codes are the widely used codes and have the longest history [9]. The strip unit, i.e., w-bit word, must be large enough to satisfy $n <= 2^w + 1$ so that words may be manipulated efficiently, and w is usually considered to fall on machine word boundaries: $w \in \{8, 16, 32, \text{ and } 64\}$. Most implementations having the systems less than 256 disks choose $w = 8$ that performs the best. Reed–Solomon codes treat each word as a number between 0 and $2^w - 1$ and operate on these numbers with Galois field arithmetic GF(2^w), which performs addition, multiplication and division on these words to have well-behaved system [10].

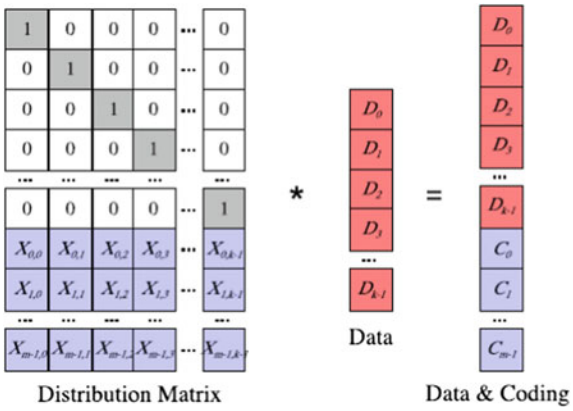

Fig. 1. Calculation of data and parity using generator matrix

There are two types of implementations of Reed–Solomon in Jerasure library, the Vandermonde matrix-based Reed–Solomon coding and Reed–Solomon coding optimized for RAID-6. In Vandermonde matrix-based Reed–Solomon coding, the last m rows of the distribution matrix in GF(2^w) are based on an extended Vandermonde matrix. Vandermonde matrix is an $m \times$k matrix that has first row and columns of the matrix, guaranteed to be all ones. In the RS-RAID-6 code, the number of parity nodes cannot be more than 2, therefore able to handle two disk failures at the most.

2.2 Cauchy Reed–Solomon (CRS) Codes

CRS codes are modified from RS codes [9]. They create the generator matrix using Cauchy matrices instead of Vandermonde matrices. They also eliminate the complicated multiplications of RS codes by converting them to XOR operations. The Cauchy matrix M can be calculated using

$$M[i,j] = \frac{1}{i \oplus (m+j)}$$

Instead of operating on single word, CRS coding operates on entire strips. In Jerasure library, there are two implementations of CRS codes, Cauchy original and Cauchy good. Cauchy good is the improved version of Cauchy original, where the improved matrix is generated by first dividing each column by its element in row 0, then similarly improving the rest of the rows.

2.3 Minimal Density Codes

Minimal density RAID-6 codes are the MDS codes that are based on binary matrices which satisfy a minimum bound on the number of nonzero entries. In minimal density codes, the bit matrix elements do not correspond to elements

in GF(2^w); however, the bit matrix itself satisfies MDS property. In this code, the bit matrix is encoded with $(k-1)+\frac{k-1}{2w}$ XOR operations per coding word. These codes perform better when w is large; thus, minimal density RAID-6 codes perform faster than Reed–Solomon and Cauchy Reed–Solomon codes for the same parameters values [11]. There are three examples supported in Jerasure Liberation coding, Liber8tion coding, and Blaum–Roth.

- Blaum–Roth codes when $w+1$ is prime [12].
- Liberation codes when w is prime [13].
- The Liber8tion code when $w=8$ [14].

3 Filesystem Characteristics

A filesystem is an important component in determining the overall performance of an application on operating system and external devices like storage disk. It serves as the interface for interacting with external storage devices. Accessing storage devices may introduce significant latency when making the data available to an application, thus affecting system performance. Filesystem implementation by incorporating feature enhancement helps in reducing this latency. The Linux kernel supports a number of block-based filesystems like XFS, BTRFS, EXT2/3/4, etc. This flexibility of choice makes it important to evaluate which filesystem delivers the best throughput performance for specific application goals. The goal of experiments is to compare the performance of different encoding and decoding on filesystems storage under similar workloads. In this paper, we considered three filesystems XFS, BTRFS and EXT4 and calculated encoding and decoding performance of implementations in Jerasure library.

- XFS: SGI released XFS for open-source community 1999. The community has integrated XFS into the kernel of the Linux OS, available for Linux distributions. XFS supports large files, and it can handle filesystems of up to 18 exabytes, with a maximum file size of 9 exabytes. XFS is a journaling filesystem and provides guaranteed consistency of the filesystem and speeds up the recovery in the various failure events such as power failures or system crashes.
- BTRFS: BTRFS is a modern filesystem that began with development in 2007. It was integrated in Linux kernel and distributed in the Linux 2.6.29 release. BTRFS is GPL-licensed but currently considered unstable and is experimental. Thus, even though Linux distribution provides BTRFS in distribution as an option, it is not yet present as the default filesystem. BTRFS is considered to offer better scalability and reliability. It is a copy-on-write filesystem that can address various weaknesses in the existing Linux filesystems. Primary focus points of BTRFS are to include fault tolerance, easy repair and easy administration filesystem. BTRFS can support up to sixteen times of the data of Ext4 as mentioned in documentations.
- EXT4: The EXT4 filesystem is an extension of EXT2 and EXT3, designed primarily to improve using journal checksums. In EXT4, data allocation was

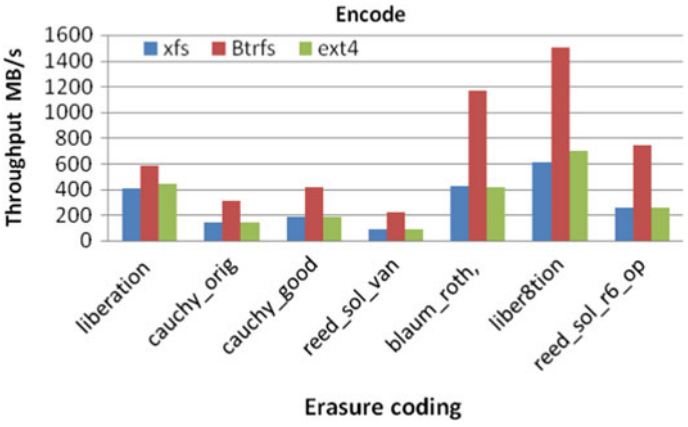

Fig. 2. Encoding performance

modified from fixed block allocation to extent allocation. In EXT4, an extent is described by its starting and ending place on the hard drive. Thus, it possible to describe a very long and contiguous files in a single inode pointer entry. This reduces the number of pointers required to describe the location of all the larger data files. To reduce fragmentation, other allocation strategies have been implemented in EXT4. Fragmentation is reduced by scattering newly created files across the disk so that they are not bundled up in one location, as was happening in earlier filesystems. Additional strategies are used such as to pre-allocate extra disk space during new file creation or expansion of the existing file. This helps to ensure that fragments are not created during file expansion. New files are not allocated immediately after the existing files; this also prevents fragmentation of the existing files. There are many more features added in EXT4 such as delayed allocation for strategizing the contiguous allocation of the files.

As per the performance evaluation performed in white paper [15], for all the above mentioned filesystem, BTRFS performs better in case of small files (sequential as well as random). For sequential read operation XFS, BTRFS and EXT4 have similar throughput; however in case of performance of random read operation, BTRFS performs better than EXT4 and XFS [16].

4 Evaluation Setup and Testing Methodology

In this paper, the encoding and decoding performance of Reed– Solomon, Cauchy Reed–Solomon and Minimal Density codes for the data from different filesystems, namely XFS, BTRFS and EXT4, is evaluated. The experiments are performed on a Dell workstation with an Intel Core i5 CPU running at 1.6 GHz with 2 GB of RAM, a L1 cache of 32 KB, a L2 cache of 256 KB and a L3 cache of 3 MB. The operating system is Debian GNU Linux revision 4.15.0-9 generic-i686.

A set of hypotheses that encompass the most important performance aspects of erasure codes in storage systems, such as encoding and decoding for unavailability of number of data blocks, has been devised. Our focus on benchmarking efforts is toward exploring and validating the following hypotheses:

- H1 The system decoding performance decreases with the unavailability of encoded data (disk failures).
- H2 For the similar parameter values, minimal density codes have better performance due to its reduced complexity.
- H3 The storage overhead of erasure coding is irrespective of coding implementations in Jerasure library and is proportional to the disk redundancy.

Evaluation of encoding and decoding patterns with respect to filesystem is also one of the objectives of experiments. To this end, we employed a sequential testing strategy in all the three filesystems in order to establish the theoretically expected performance for various code implementations of Jerasure. Encode and decode functions are executed 100 times each, and average throughput (MB/s) is calculated for all codes in Jerasure library. The decoding experiments are performed for number of tolerated disk failure for each code in Jerasure. All the experiments are performed by taking care to flush filesystem buffers and drop caches before the running each test.

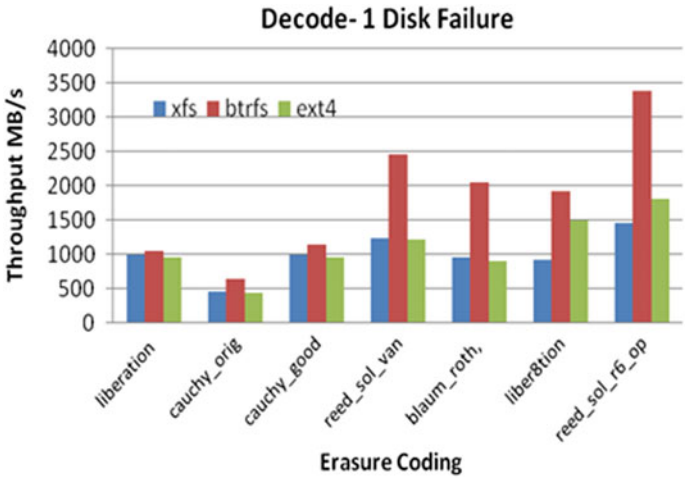

Fig. 3. Decoding performance—one disk failure

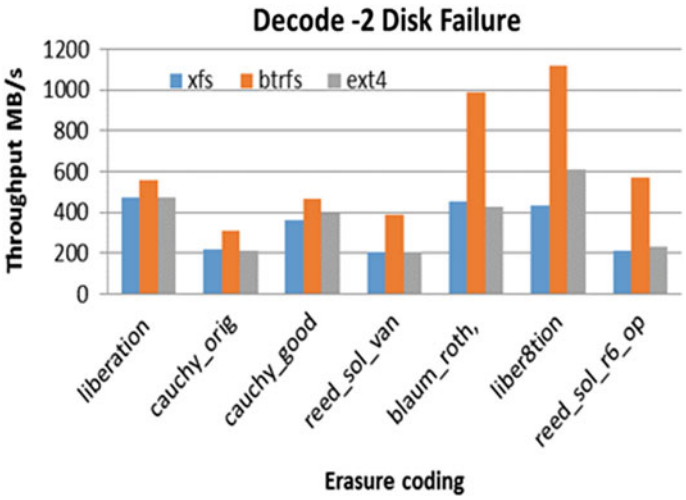

Fig. 4. Decoding performance—two disk failures

5 Microbenchmarks

The encoding and decoding of large movie file of 396.8MB are performed for each code implementation of Jerasure library for a video file on three filesystem formats XFS, BTRFS and EXT4. In this section, coding algorithm performance characteristics e.g. encoding, decoding and storage overhead with respect to the coding parameters and selected filesystem type, are evaluated.

5.1 Coding Algorithms Performance Characteristics

Parameters k and m for encoding and decoding are considered as follows. RS-Vandermonde, Liberation, Cauchy, Blaum–Roth: $k = 6$, $m = 4$, $n = 10$.

Figure 2 presents the encoding performance for various code implementations, for three filesystem, namely XFS, BTRFS and EXT4. It is observed that encoding in case of BTRFS performs better than other two filesystems for all code implementations. If code implementations are compared, Blaum–Roth coding in BTRFS gives better encoding throughput (around 1170.93 MB/s) than other coding implementations. In case of RAID-6 implementations, Liber8tion in BTRFS performs better than RS-RAID6 (around 1501.02 MB/s).

Figure 3, 4, 5 and 6 show the decoding performance of all the codes of one, two, three and four disk failures, respectively. It is observed that as the number of disks fails representing unavailability of coded blocks, the decoding throughput to recreate the data information decreases. Comparing filesystem performance for decoding, the results are as expected, i.e., BTRFS decoding is high performance or all three decoding scenarios. In case of coding library comparison,

blaum roth in BTRFS for one, two and four disk failure scenarios performs better. In case of RAID-6 implementations, Liber8tion code performs better than RS-RAID codes.

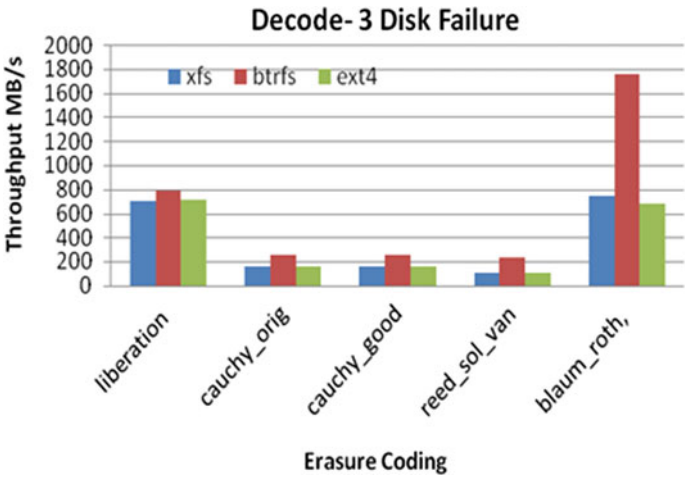

Fig. 5. Decoding performance—three disk failures

5.2 Storage Overhead

For various experiments done in this paper, a video file of 396.8 MB is considered. With all the Jerasure coding implementations, the encoded n files each of size 66.2 MB are created. In case of RAID-6 implementations the total encoded size n is 8($k = 6$, $m = 2$) and for other implementations n is 10 ($k = 6$, $m = 4$). Storage overhead is the difference between the actual size of the file and the encoded size of the file and represented as follows.

$$\text{storage overhead} = 100 \times \frac{n \times \text{encoded file size-actual file size}}{\text{actual file size}}$$

6 Conclusion

In this paper, the performance characteristics of Jerasure library code implementations for different filesystem data are evaluated. The experiments are based on three different widely used Linux file formats XFS, BTRFS and EXT4. The concluding remarks for hypothesis backed by empirical results with various experiments carried are as follows.

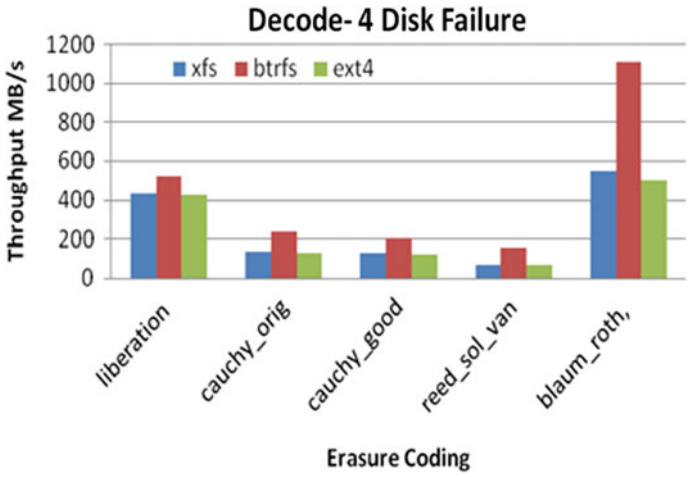

Fig. 6. Decoding performance—four disk failures

- BTRFS filesystem performs better than other two filesystems for encoding as well as decoding [17]. As mentioned in various research articles, BTRFS has better performance due to reduced complexity of bit matrix design based on scalability and deterministic data access. This is now backed by empirical proof.
- The Blaum–Roth coding in BTRFS gives better encoding throughput (around 1170.93 MB/s) than other coding implementation
- Minimal density codes compared with RS and Cauchy RS codes. The demonstrable results are as follows
  - In case of RAID-6 implementations, Liber8tion in BTRFS performs better than RS-RAID6 for data encoding with throughput around 1501.02 MB/s.
  - It is observed that as the number of disks fails representing unavailability of coded blocks, the decoding throughput decreases.
  - The coding library comparison, blaum_roth, in BTRFS for one, two and four disk failures performs better than other coding implementation. For three disk failure scenarios, liberation code in BTRFS performs better.
  - In case of RAID-6 implementations, Liber8tion code performs better than RS-RAID codes.
  - For three disk failure scenarios, liberation code in BTRFS performs better.
- The storage overhead for 396.8 MB file is around 33.46% for RAID-6 codes and around 66.83% for other code implementations.

Acknowledgment. The research leading to the results is an outcome of the R&D work undertaken project under the Visvesvaraya Ph.D. Scheme of Digital India Corporation, Ministry of Electronics and Information Technology, Government of India.

References

1. O'Reilly J, Raid versus erasure coding. http://www.networkcomputing.com/storage/raid-vs-erasure-coding/a/d-id/1297229. Online; Accessed 09 Oct 2015
2. Valb, Erasing misconceptions around raid & erasure codes. http://community.netapp.com/t5/Company/ErasingMisconceptions-Around-RAID-amp-Erasure-Codes/ba-p/83689. Online; Accessed 09 Oct 2015
3. https://bitbucket.org/tsg-/liberasurecode
4. http://jerasure.org/jerasure/jerasure
5. https://software.intel.com/en-us/storage/ISA-L
6. https://github.com/catid/longhair
7. 2008 WILCOX-O'HEARN, Z. Zfec 1.4.0. Open source code distribution capitalized. J Name Stand Abbrev http://pypi.python.org/pypi/zfec (in press)
8. Singh K, Learning Ceph- A practical guide to designing, implementing, and managing your software-defined, massively scalable Ceph storage system. Packet publishing
9. Plank JS, Xu L (July 2006) Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications. In: NCA-06: 5th IEEE international symposium on network computing applications, Cambridge, MA
10. Macwilliams FJ, Sloane NJA (1977) The theory of error-correcting codes, part I. North-Holland Publishing Company, Amsterdam
11. Plank JS, Luo J, Schuman CD, A performance evaluation and examination of open-source erasure coding libraries for storage. In: FAST '09 Proceedings of the 7th conference on File and storage technologies technologies
12. Blaum M, Roth RM (1999) On lowest density MDS codes. IEEE Trans Inf Theory 45(1):46–59
13. Plank JS (2008) The RAID-6 Liberation codes. In: FAST-2008: 6th Usenix conference on file and storage technologies San Jose, February 2008, pp 97–110
14. Plank JS (2006) A new minimum density RAID-6 code with a word size of eight. In: NCA-08: 7th IEEE international symposium on network computing applications, Cambridge, MA, July 2008
15. Nanal N, white paper on File system throughput performance on RedHawk^{TM}Linux
16. Burihabwa D, Felber P, Mercier H, Schiavoni V (2016) A performance evaluation of erasure coding libraries for cloud-based data stores (Practical Experience Report). In: IFIP international federation for information processing 2016, pp 160–173
17. Vujicic D, Markovic D, Djordjevic B, Ranić S (2016) Benchmarking Performance of EXT4, XFS and BTRFS as Guest File Systems Under Linux Environment, IcETRAN 2016, At Zlatibor, Serbia