Distributed File System based on Erasure Coding for I/O Intensive Applications

Dimitri Pertin, Sylvain David, Pierre Evenou, Benoît Parrein, Nicolas Normand

Distributed File System based on Erasure Coding for I/O-Intensive Applications

Dimitri Pertin[1,2], Sylvain David[2], Pierre Évenou[2], Benoît Parrein[1] and Nicolas Normand[1]
[1]*LUNAM Université, Université de Nantes, IRCCyN UMR CNRS 6597, Nantes, France*
[2]*Fizians SAS, Nantes, France*
{*dimitri.pertin,benoit.parrein,nicolas.normand*}*@univ-nantes.fr,* {*sylvain.david, pierre.evenou*}*@rozofs.com*

Keywords: Distributed File System, RozoFS, Erasure Coding, Mojette Transform, IOzone, Video Editing.

Abstract: Distributed storage systems take advantage of the network, storage and computational resources to provide a scalable infrastructure. But in such large system, failures are frequent and expected. Data replication is the common technique to provide fault-tolerance but suffers from its important storage consumption. Erasure coding is an alternative that offers the same data protection but reduces significantly the storage consumption. As it entails additional workload, current storage providers limit its use for longterm storage. We present the Mojette Transform (MT), an erasure code whose computations rely on fast XOR operations. The MT is part of RozoFS, a distributed file system that provides a global namespace relying on a cluster of storage nodes. This work is part of our ongoing effort to prove that erasure coding is not necessarily a bottleneck for intense I/O applications. In order to validate our approach, we consider a case study involving a storage cluster of RozoFS that supports video editing as an I/O intensive application.

1 INTRODUCTION

Distributed storage systems have been used to provide data reliability. Failures in such large systems are considered as the norm, and can come either from hardware or software considerations. They can result in dramatic data loss and/or the crash of the service. The traditional way to deal with data protection is to replicate the data. Once n copies of data are distributed across multiple network nodes, the system is able to face $n-1$ failures. If a node has a breakdown, its data is not accessible anymore, but other copies are still available on other nodes. On the other hand, this technique is expensive, particularly for huge amounts of data. Replication is the default data protection feature included in distributed storage systems.

Erasure coding is an alternative that provides the same data protection but reduces significantly the storage consumption. Optimal codes, called Maximal Distance Separable (MDS), aim at encoding k data blocks into n parity blocks, with $n \geq k$. The encoded blocks hold enough redundancy to recover the former data from any subset of k parity blocks during the decoding process. Compared to replication whose storage overhead is n, erasure coding's overhead is defined by n/k. However, encoding and decoding require further computations compared to the simple replication technique. Efforts are done today to design efficient codes. The most famous ones are the Reed-Solomon (RS) codes. They rely on Galois field operations. In this arithmetic, addition is fast as it corresponds to exclusive-OR (XOR), but multiplication has many implementations which are much more computational expensive. The storage systems that provide erasure coding by RS suffer from the slowing down of its computations. Many implementations of RS codes exist. Examples of implementation libraries are OpenFEC[1] and Jerasure (Plank, 2007).

We propose the use of the Mojette Transform (MT) (Normand, Kingston, and Évenou, 2006) as an erasure code for distributed storage. The MT relies only on additions and its implementation uses the fast XOR operations. The MT is part of RozoFS[2], an open-source software providing a distributed file system. In RozoFS, when a client wants to store a file, the data is cut into small chunks of 8 KB. Chunks are composed of k blocks of fixed size which depends on the desired protection. These blocks are encoded into n parity blocks, which are discrete projections in the Mojette formalism. These projections are then distributed to storage nodes and the system is able to face

[1]http://openfec.org/
[2]code available at http://www.rozofs.org

$n-k$ failures. Decoding is done client-side after retrieving k blocks out of the n from storage nodes.

The current storage systems provide erasure coding to benefit from the storage capacity saving for longterm storage. When intensive data I/Os are needed, replication is put forward because of the additional workload of erasure codes. Our work differs from this position by exploring the use of an erasure code for intensive applications. We experiment the concept by applying RozoFS to video editing.

The rest of the paper is structured as follows: Section 2 briefly describes some distributed storage systems and their data protection policy. In Section 3, we present the MT and its erasure code characteristics, while Section 4 describes how RozoFS use it to provide a distributed file system. The video editing experiment is presented in section 5 and demonstrates that erasure coding is not a bottleneck for I/O intensive applications. Section 6 concludes the paper with the possible future work.

2 RELATED WORK

Erasure Coding receives significant attention from both the business and the research communities. Scality, Caringo, Isilon or Cleversafe are such examples of companies providing private storage solutions based on erasure codes. Data protection for these systems are partially described in technical white papers.

Scality and Caringo provide replication and erasure coding as a way to protect data. Erasure coding is only used for the longterm storage of massive amounts of data. They both recommend replication for intensive applications. Isilon puts forward the use of erasure coding through a Reed Solomon implementation in OneFS. Replication is only used when erasure coding is not applicable (e.g. too few many nodes). Cleversafe provides exclusively data protection through erasure coding. It relies on the Luby's implementation (Blömer et al., 1995) of the Reed Solomon algorithm.

Besides these solutions, famous and stable free alternatives exist. One of the most famous free solution is Apache HDFS. This Distributed File System (DFS) aims at archiving and data-intensive computing (i.e. not really an I/O centric DFS). It relies on the MapReduce framework that divides and distributes tasks to storage nodes. Data protection is done by replication, and triplication is the default protection policy. Carnegie Mellon University has developed a version based on erasure codes, called HDFS-RAID (Fan et al., 2009). It is based on Reed-Solomon codes for generic parity computations. HDFS-RAID has been applied in practice (e.g. Facebook Warehouse). However, only a small part of the warehouse's data has been translated by erasure coding. Warm data (i.e. frequently accessed data) is still replicated. Once the data have not been used for a while, the raid-node daemon encodes it and erases the corresponding replicated blocks to make room. GlusterFS[3] and Ceph (Weil et al., 2006) are examples of popular I/O centric DFS. Replication is currently their standard for data protection, but erasure coding is a hot topic on the roadmap.

Erasure coding is already included in private solutions but it is not to be efficient enough for intensive applications. It is exclusively used today for longterm storage and systems benefit from its low data consumption compared to replication. Open-source storage solutions are still actively looking for high speed erasure code implementations as an alternative to replication.

Our contribution, RozoFS, is an open-source software jointly developed by Fizians SAS and the University of Nantes. It provides a POSIX DFS whose file operations are exclusively done by erasure coding. This code is based on the MT whose computations rely entirely on the very fast XOR operations. Thus, we expect RozoFS to be efficient enough for intensive applications.

3 MOJETTE TRANSFORM

The Mojette Transform (MT) is a mathematical tool based on discrete geometry. Long developed at the University of Nantes, its first definition as an erasure code remains in (Normand, J. Guédon, et al., 1996). A first design for distributed storage relying on the MT was proposed in (J. P. Guédon, Parrein, and Normand, 2001).

3.1 Algorithm

Fundamentally, an erasure code should compute efficiently a desired amount of redundant information from a set of data. This operation is done by a linear operation that should be efficiently inverted. The inverse operation reconstructs the former data from a subset of the computed information.

The Mojette encoding is done as follows. A set of data fills a discrete grid whose elements are identified by (i, j), where i and j are integers. The linear transform consists in summing these elements following different discrete directions. Each direction is defined

[3] http://www.gluster.org/

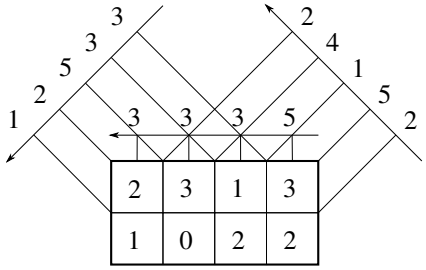

Figure 1: The Mojette Transform, applied to two lines of 4 integers, computes here the following set of 3 projections: $\{(p_i,q_i)=(-1,1),(0,1),(1,1)\}$.

by a couple of integers (p,q). The elements aligned in a direction are summed to form a bin. The set of bins defined by the same direction is called a projection. Hence, it is possible to use the Mojette Transform to compute a desired number of projections from a set of lines of data. If k and n are respectively the number of lines of the grid, and the number of projections, and if $n > k$, the set of projections holds redundant information. Figure 1 shows the computation of 3 projections from a set of 2 lines of data.

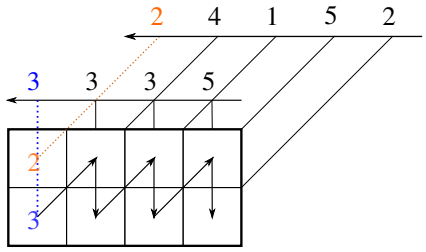

Figure 2: Inverse Mojette Transform as described in (Normand, Kingston, and Évenou, 2006).

The decoding is the inverse operation. It consists in filling the empty grid with the only projection data. Data reconstruction is possible if the Katz's criterion (Normand, Kingston, and Évenou, 2006) holds for the rectangular-shaped grid. This criterion was extended for any convex shape in (Normand, J. Guédon, et al., 1996). Reconstruction is straightforward for the elements in the corners of the grid. Indeed these elements match entirely a projection bin. (Normand, Kingston, and Évenou, 2006) showed that if a reconstruction path can be found to fill a side of the grid, then it can be applied several time to easily reconstruct the grid. Figure 2 represents such reconstruction considering the example of Figure 1. According to the Katz criterion if the following condition is sufficient to validate the reconstruction:

$$\sum_{i=0}^{r} q_i \geq k \qquad (1)$$

where r is the number of projections that remains, q_i depends on this projection set, and k is the number of lines of the grid. For instance, in Figure 2, $\sum_{i=1}^{2} q_i = 2$ which equals k, so reconstruction is possible. By modifying the number of lines k and the number of projections n, it is possible to set the desired fault-tolerance threshold.

3.2 Implementation

In storage application we cut the input stream into k fixed size blocks. Each line of the grid in the Mojette representation is such a data block. The shape of the grid must be rectangular then. The set of projections is set so that $q_i = 1$. If we suppose that l, the size of blocks (i.e. grid columns) is higher than k, the number of blocks (i.e. grid lines), then the Katz's criterion guarantees that reconstruction is possible from any subset of k projections.

The strength of the Mojette Transform is that encoding and decoding rely entirely on additions. Compared to classical codes like Reed Solomon, there is no need to compute expensive multiplications or divisions in Galois fields. The elements of the grid should fit computer words to improve computation performance. Then, addition is done by efficient exclusive-OR (XOR). Most recent Intel-based processors can perform very fast computations with elements of 128 bits as it fits the dedicated SSE (Streaming SIMD Extensions) registers.

In traditional solutions like Reed Solomon codes, the size of the parity blocks is fixed. The Mojette Transform relaxes this constraint. The size B of each projection varies slightly with the angle of projection, and is given by the following formula:

$$B(k,l,p,q) = (k-1) \mid q \mid +(l-1) \mid p \mid +1 \qquad (2)$$

where k is the number of the grid lines and l the size of blocks. The MT is then considered as $(1+\varepsilon)$ MDS (Parrein, Normand, and J. P. Guédon, 2001), where ε is the quotient of the number of bins required for decoding by the number of element in the grid. The set of projections is taken such as $q = 1$ and p varies around 0 in order to minimize the value of ε.

4 ROZOFS

RozoFS is an open source software solution providing a scalable distributed file system. Once mounted, the file system yields a unique name space (i.e. an hierarchy of directories and files), that relies on clusters of commodity storage nodes. RozoFS manages scalability by adding storage nodes to expand the global resources of the system. For data reliability, it relies on the Mojette Transform as an erasure code.

4.1 Information Dispersal

We consider a network of commodity computers (or nodes) communicating by one or several links. The more the links, the more the paths for packets, and the more reliable and high-performance the system is. Reliability comes from the capacity to communicate even if a link is down. Again, failure probabilities are non negligible and should be considered as the norm. Multiple links induce high-performance because packets can be sent in parallel.

RozoFS considers flows of information of fixed size. A small data chunk of 8 KB fills a Mojette grid as seen in Figure 1. The protection settings, called layouts, define the value of n and k (respectively the number of projections and the number of lines in the Mojette grid).

Table 1: The protection settings provided by RozoFS.

Layout	k	n	storage nodes	fault-tolerance
0	2	3	4	1
1	4	6	8	2
2	8	12	16	4

Currently, three layouts are designed in RozoFS. The table 1 displays the relative information for these configurations. Each layout corresponds to a storage consumption of $n/k = 1.5$ times the size of the input. For instance, we consider the write operation of 1 GB of data in an exported volume of RozoFS, set with the layout 0. It results that 3 projection files of around 500 MB (i.e. plus ε) are distributed across physical disks.

4.2 Stackable Architecture

RozoFS relies on three components:

- *exportd*: the daemon that manages metadata;
- *storaged*: the daemon in charge of the projections;
- *rozofsmount*: used by the clients to mount RozoFS locally.

For the sake of modularity, these components are stackable. For instance, it is either possible to collocate them on a single node, storing projections on different disks, to obtain protection over disk failure (similar to some RAID configurations). In a large network topology, dedicated nodes are much more appreciated for the sake of scalability.

RozoFS Client. Each client can use RozoFS as a standard POSIX file system thanks to the *rozofsmount* process. It relies on FUSE[4] to operates in the user-mode. Clients handle two kinds of operations: (i) the metadata operations (*lookup*, *getattr*, etc), interfacing the export server; (ii) the file I/O operations (read/write the projections), interfacing the storage nodes through the *storcli* subprocesses. The encoding and decoding workload are managed by the clients. The network design should take care of reliability. Services and links must be replicated to provide availability facing failures.

[4]http://www.fuse.sourceforge.net

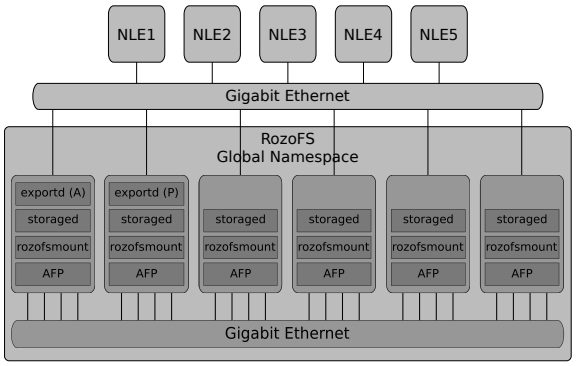

Figure 3: The RozoFS cluster is composed of 6 nodes that hold different services. They provide the DFS to 5 clients running a Non-Linear Editing (NLE) application.

Metadata Server. This node manages the *exportd* process that services the metadata operations. Its configuration file defines the exported file systems. It keeps statistics of storage nodes to provide a data distribution policy based on the storage capacity. The metada server supplies clients with two lists: (i) the list of storage nodes servicing the mounted volume; (ii) the list of storage nodes associate with a regular file (for read/write operations). This service should be replicated to guarantee high availability.

Storage Nodes. These entities hold the *storaged* daemon. Two services are provided by storage nodes: (i) the management services to exportd; (ii) the projection I/O services to the clients, called the *storio* processes. Each storio process can listen to a specific network interface. A storcli groups all storio of a storage node in a load-balancing group to improve availability.

5 EVALUATION

In this section, we explore the capacity of RozoFS to scale as the global workload increases.

Experiment Setup. For our evaluation, we employ a RozoFS cluster of 6 similar servers. Each machine contains an Intel Xeon CPU @ 2.40 GHz, 12 GB of RAM, and 10 disks (7200 rpm SAS) with 2 TB each. These drives are managed by a RAID-5

controller, providing better I/O performances and local reliability. Each node holds 8×1 Gbit Ethernet ports. The exported volume is set to layout 0 providing fault-tolerance against a single failure. The services are distributed as follows:

- the 6 nodes serve as storage servers;
- one among them serves as active metadata server;
- another is a passive/replicated metadata server;
- the 6 nodes mount RozoFS and re-export the file system through the AFP protocol for client access;

Figure 3 displays the platform used here. For high speed communications and reliability, 4 Ethernet ports are reserved for storio processes. RozoFS manages itself the packet scheduling for load-balancing and availability. Because the metadata server is a potential point of failure, it is necessary to set a high-availability strategy. Here, we use DRBD[5] to synchronise the active metadata server with the passive one. Pacemaker[6] is used as a cluster management software that manages the failover mechanisms in case of failure.

The Study Case. We use the previous RozoFS cluster as a storage solution for video editing. Non-Linear Editing (NLE) applications entail intensive workloads for computers. Multiple source editing is particularly file I/O intensive since multiple multimedia contents are accessed simultaneously. For this experimentation, we use the famous NLE software Apple Final Cut Pro[7]. Remote clients, running on Apple Mac OS, attach the RozoFS file system to their own machine using the AFP protocol. Once mounted, the file system provides an easy access to the source media stored on RozoFS. In our case, the system stores video encoded with the lossy compressed format Apple ProRes 422. It requires at least a rate of 200 Mbit/s (i.e. 25 MB/s) to avoid frame drops. We consider that editing involves 5 input video files simultaneously and outputs a single one. The software is both designed for sequential operations (e.g. read the video stream) and direct access to a part of the video.

5.1 IOzone Benchmark

IOzone[8] is a filesystem benchmark that can output file I/O performance according to a set of stressing tests. In particular, the software can perform sequential and random read/write tests, which are in accordance with our study case. We explore the capacity of RozoFS to adapt, facing a growing number of clients accessing the file system simultaneously.

[5]Distributed Replicated Block Device (www.drbd.org)
[6]http://clusterlabs.org/
[7]http://www.apple.com/final-cut-pro/
[8]http://www.iozone.org

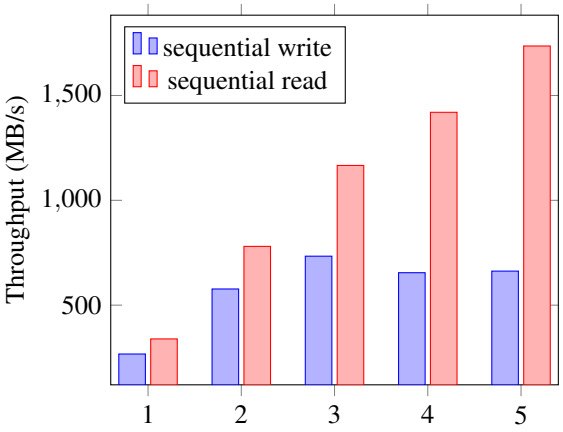

Figure 4: Accumulated throughput (in MB/s) recorded by the threads, depending on the number of clients working in parallel, for sequential access operations.

IOzone takes the following parameters. The file size is set to 1 GB which is larger than the CPU cache, and smaller than the RAM capacity. To fit the multi-source editing, each client involved in the test manages 5 threads. A thread reads or writes the 1 GB file according to the desired access strategy. For instance, in a writing test, each node induces the writing of 5 GB in RozoFS (i.e. which represents 7.5 GB in the physical disks).

We measure the accumulated throughput recorded by the threads as the number of clients grows. Each thread should access data with a rate of at least 25 MB/s to validate the study case. Figure 4 and 5 respectively display the average results from 5 test iterations for sequential and random access.

Benchmarking Sequential I/Os. The write operations are asynchronous and multiple write requests are done simultaneously. Figure 4 shows that as the workload grows, the performance for sequential write scales up to 3 nodes. When more clients are added, the nodes are overwhelmed by the requests and cache misses slow down the performances as data need to be fetch on the disks. The sequential read operations significantly benefit from fast access as data is prefetched in the cache. The performance scales up as the number of clients increases. For instance, when 5 clients are involved, the 25 threads record an accumulated throughput of 1800 MB/s. Each client receives $1800/5 = 360$ MB/s and each thread receives $360/5 = 72$ MB/s. In any case in this benchmark, we validate the study case as each thread receive more than 25 MB/s. Mechanical disks are particularly sensitive to the type of access. For sequential access, the

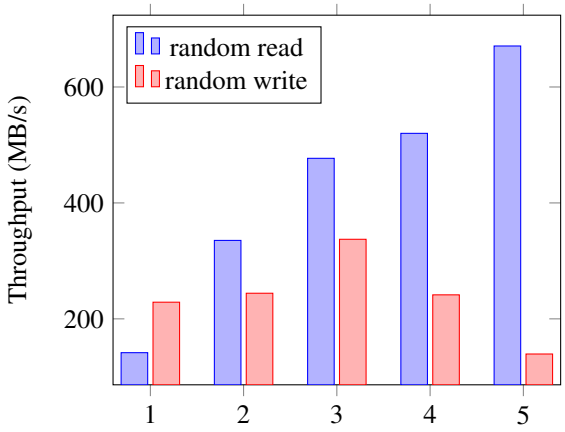

Figure 5: Accumulated throughput (in MB/s) recorded by the threads, depending on the number of clients working in parallel, for random access operations.

disk's arm move the read-and-write head to reach the correct track on the physical disk. Once there, the following accesses are straightforward.

Benchmarking Random I/Os. Random operations are slowed down by intermediate seeks between operations, increasing the latency of disk's head. Figure 5 shows that the throughput suffers from these random accesses. During random read operations, the performance scales as it benefits from small cache effects and each thread receive at least 25 MB/s. However, the random write test is clearly the worst case and shows the limit of hardware as the performances collapse. For 3 clients, each thread receives $350/3/5 = 23$ MB/s. For more clients, performances get worse. We should note that 5 writing threads per client does not fit our study case. For video editing, each editors outputs a single file. It would correspond to a single write thread in our case.

6 CONCLUSIONS

In this work, we have presented the Mojette Transform (MT) as an efficient alternative to classical erasure codes since it relies on fast computations, and a great candidate to handle intensive applications. We have set the MT as the I/O centric process in RozoFS with good expectations for practice efficiency. Finally, we have designed an evaluation based on a RozoFS cluster. The platform is able to handle multiple parallel access from intensive I/O applications (i.e. multiple source video editing). The evaluation has revealed that erasure coding is not a bottleneck for intensive applications and should not be limited to longterm storage.

More specific measures should be done to reveal the real cost of the MT in the computational time. Our evaluation could be extended to more intensive applications like virtualization, which access data over block devices. There is clearly a need for comparisons with other existing solutions. The MT must be compared to erasure code libraries such as OpenFEC and Jerasure. Further erasure code aspects beyond computational considerations should be explored, such as the node repair problem.

ACKNOWLEDGEMENTS

This work has been supported by the French Agence Nationale de la Recherche (ANR) through the project FEC4Cloud (ANR-12-EMMA-0031-01).

References

Blömer, J. et al. (1995). *An XOR-based erasure-resilient coding scheme*. Tech. rep. TR-95-048. International Computer Science Institute.

Fan, B. et al. (2009). "DiskReduce: RAID for data-intensive scalable computing". In: *Proc. PDSW 2009*. Portland, Oregon: ACM. DOI: 10.1145/1713072.1713075.

Guédon, J. P., B. Parrein, and N. Normand (2001). "Internet Distributed Image Information System". In: *Integr. Comput.-Aided Eng.* 8.3, pp. 205–214. ISSN: 1069-2509.

Normand, N., J. Guédon, et al. (1996). "Controlled redundancy for image coding and high-speed transmission". In: *Proc. VCIP 1996*. Vol. 2727. Orlando, FL. DOI: 10.1117/12.233180.

Normand, N., A. Kingston, and P. Évenou (2006). "A geometry driven reconstruction algorithm for the Mojette transform". In: *DGCI*. Vol. 4245. LNCS. Springer Berlin Heidelberg. DOI: 10.1007/11907350_11.

Parrein, B., N. Normand, and J. P. Guédon (2001). "Multiple description coding using exact discrete Radon transform". In: *Proceedings of the Data Compression Conference*. DCC '01. Washington, DC, USA: IEEE Computer Society. DOI: 10.1.1.19.5708.

Plank, J. S. (2007). *Jerasure: A library in C/C++ facilitating erasure coding for storage applications*. Tech. rep. CS-07-603. University of Tennessee.

Weil, S. A. et al. (2006). "Ceph: A scalable, high-performance distributed file system". In: *Proc. OSDI 2006*. Seattle, Washington: USENIX Association, pp. 307–320.