2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)

# Dynamic Erasure Coding Policy Allocation (DECPA) in Hadoop 3.0

Aatish Chiniah
Department of Digital Technologies,
University of Mauritius.
Reduit, Mauritius
a.chiniah@uom.ac.mu

Avinash Mungur
Department of ICT
University of Mauritius
Reduit, Mauritius.
a.mungur@uom.ac.mu

***Abstract*—Erasure Code (EC) is being tipped to be the next best alternative to Replication for providing redundancy for Cloud Storage. Already major players like Microsoft and Facebook are having initial implementations using Erasure Code. Hadoop 0.20 was the first version that supported Erasure Code (aka HDFS-RAID), but EC was not included in later versions, only to resurface in Hadoop 3.0 (HDFS-EC). Hadoop 3.0.0 supports three default Erasure Code polices which are RS(3,2), RS(6,3) and RS(10,4). To have greater flexibility, in this work we opt for the implementation of new Erasure Code policies [RS(4,3), RS(5,3), RS(7,3), RS(7,4), RS(8,4), RS(9,4)] and the development of a Dynamic Erasure Coding Policy Allocation, based on minimum overhead produced, in order to maximize storage capacity. Three types of dynamic allocation have been proposed and implemented. A performance evaluation of the new polices was conducted in order to find the optional one for a NAS/SAN architecture and the effectiveness of the three implemented Dynamic Allocation of EC policy is provided.**

***Keywords: Erasure Code, Replication, Cloud Storage, Network Attached Storage, Hadoop 3.0.***

## I. INTRODUCTION

Hadoop [1][2] (aka Apache Hadoop) is an open-source software designed to operate on networked computer (cluster) to manipulate massive amount of data. The core of Hadoop is firstly its file system known as Hadoop File System (HDFS) and secondly, its programming framework known as Map-Reduce programming model. Hadoop was first released in 2011 and its latest stable release is Hadoop 3.0.0, launched in December 2017. During that period of time, a lot of enhancements have seen applied to Hadoop. Examples of enhancement are: Yarn, Oozie, Hive, Pig-Latin. Hadoop has since been widely adopted, by major cloud providers such as Yahoo and Facebook, just to mention a few.

Hadoop's defacto Redundancy scheme is the expensive Replication – the default 3x replication scheme in HDFS has 200% overhead in storage space and other resources such as CPU consumption and network bandwidth. Despite that the replicas are used only in cases of failures. However they need to be available as and when needed, therefore need to be live.

An alternative solution is to use Erasure Coding (EC) in lieu of replication, as EC provides the same level of fault-tolerance with much less storage space. In typical Erasure Coding (EC) setups, the storage overhead is no more than 50%.

Erasure Coding can be divided into 3 sub-processes: Splitting, Encoding and Placing. Any file is spilt into *n* blocks, and then encoded to produce *m* parity blocks. All the blocks are placed in different datanode, so as to minimize block lost in cases of HDD failures. A typical (10,4) implementation is shown in Fig 1.

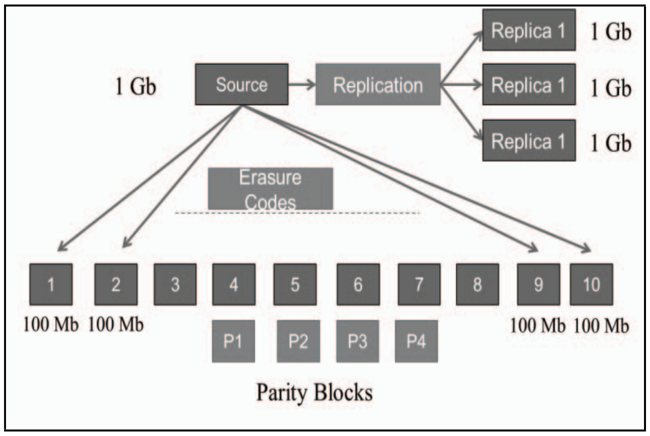

Fig.1 – Erasure Code VS Replication

The effectiveness of Erasure Code compared to Replication is summarized in the table below:

**Table 1. Efficiency of Erasure Codes [3]**

| | Data Durability | Storage Efficiency |
|---|---|---|
| Single Replica | 0 | 100% |
| 3-way Replication | 2 | 33% |
| XOR with 6 data cells | 1 | 86% |
| RS(6,3) | 3 | 67% |
| RS(10,4) | 4 | 71% |

From Table 1, we identify that there is a trade-off between data durability (availability) and storage efficiency. To minimize this trade-off, we propose DECPA to dynamically apply most appropriate EC policy depending on size of data to maximize storage efficiency.

This paper is structured as follows: section II provides an overview of the related works. Section III gives an insight on Erasure Code as deployed in Hadoop 3.0. Section IV illustrates the algorithm DECPA and its three derivatives. Section V explain the Experimental Setup used in this work. Section VI present performance results of DECPA as well as evaluation of its effectiveness and lastly Section VII is the conclusion.

## II. RELATED WORKS

Minimising storage space is one of the main characteristics of erasure codes. Though producing a maximum efficiency of 71% compared replication, erasure coding still produces some overhead. Different approaches have been worked out so far to minimise storage overhead. In Zebra [4], it determines the erasure code parameters by using geometric programming to minimise storage and reconstruction overhead. In [5], a new erasure code technique is proposed called DLRC (Dynamic Locally Reconstruction Code) which uses four parameters. One of them being storage overhead. In FlexBM [6], a dual dual-scheme block management approach is used, applying either erasure code or replication to data according to its temperature. For hot data, replication is used, and for cold, erasure code is applied. This approaches have to some extent achieved to further storage saving taking into account that files are larger than blocks. With the emergence of IoT, a lot of small files are being generated and stored in the cloud [7]. To optimally store them, object storage has been proposed in [8]. In the same work, it is mentioned that 0.9% of total storage capacity is occupied by objects smaller than 4 MB, but still stored as 64 MB blocks. Thus our motivation of creating DECPA that will dynamic allocate erasure code policies to be applied on objects.

Dynamic allocation of blocks for replicated system are addressed in different works, namely in [9], [10] and [11]. As for Erasure Coding Systems, major cloud providers such as Azure, Amazon, Google and so on, have already embarked on projects working with Erasure codes. However, given the newness of the latest version of Hadoop, very few publication are available in this domain. One of the few available, is the performance evaluation of Hadoop 3.0 [12], where benchmarking of the different components of Hadoop is performed and presented. However this work does not evaluate EC part of Hadoop 3.0. We also benchmark the performance of EC on our cluster and present it in the results section.

## III. ERASURE CODE ON HADOOP 3.0

Apart from supporting Erasure Coding, Hadoop 3.0 has adopted a series of changes as follows: usage of Java 8, support for opportunistic containers, shell script rewrite, support for more than 2 namenodes, and support for file system connector and intra-datanode balancer [13].

Incorporating Erasure Code again in Hadoop make sense as data can be read/write in pipeline from different datanodes simultaneously. Working in pipeline, the blocks can be striped and distributed fairly amongst the different datanodes following the rack-aware [14] principle.

The first major change in Hadoop 3.0 is the architecture for Erasure Code. To reduce memory consumption of Namenode keeping track of the locations of each block, a new block naming protocol is used. HDFS-RAID was using a block ID which is allocated sequentially depending on the time of creation of the block. HDFS-EC also uses Block ID, which consists of a flag (0 = contiguous/1 = striped), a middle section, which indicates the logical block and a last part which represents the index of the storage block in the logical block. In doing so, a namenode is able to manage a logical block (Data + Parity Blocks) at the level of block group rather than individuals blocks.

The second major change is the application of Erasure Code Policy. In HDFS-RAID, a whole system had to be encoded in only one EC Policy. The EC Policy is set in the configuration of Hadoop, and is applied as soon as Hadoop is started. In HDFS-EC of Hadoop 3.0, EC Policies are applied on files and directories. This allows different directories to have different EC Policies. Subsequently all files and subfolders within a folder on which EC policy has been applied, will also inherit that EC policy.

Another change is in the recovery process. Previously, recovery of loss blocks was done by the Namenode itself, however, in Hadoop 3.0, the Namenode is still responsible for block health monitoring using periodic heartbeats. Whenever the Namenode detects a failed block, it chooses an appropriate datanode to perform the recovery. The choice of the datanode is primarily based on locality of other related data blocks to minimize bandwidth requirements. Each datanode has an ECWorker [15] process running, which is responsible for recovery once contacted by Namenode.

## IV. DECPA – DYNAMIC ERASURE CODING POLICY ALLOCATION

The default block size of Hadoop 3.0 has increased from 64 to 128 MB. Assuming that RS(3,2) EC Policy is used, any file with file size less than 384 (128*3) MB will fit within a stripe of 3 data blocks. However if the file size is greater than 384, then 2 stripes of RS(3,2) will be required. Alternatively, the policy RS(6,3) can be used, which can store up 768 MB in a single stripe. And finally if the file/folder size is greater than 768 MB, then RS(10,4) can also be used. So as to reap the benefits of the different EC

Policies, new EC policies {RS(4,3), RS(5,3), RS(7,3), RS(7,4), RS(8,4), RS(9,4)} were implemented, and then DECPA was designed. DECPA will allocate EC policies so as to minimize the number of stripes produced, hence minimize storage usage. The efficiency (amount of extra storage vs number of failures tolerable) of the EC Policies is given in Fig 2.

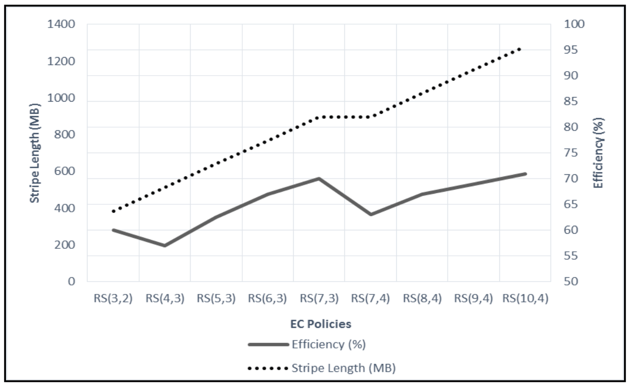

Fig 2 – Efficiency of EC Policies

From the graph, it can be seen that the maximum efficiency is reaped from RS(7,3) with an average size stripe (7*128 MB plus 3*128MB). However if a file or a number of files grouped as one with size 600MB for example, will result in the creation of a number of padded blocks (blank ones) to create a complete stripe and that is the rationale for coming up with DECPA.

## *A. Algorithm of DECPA*

By default, the Block Size in Hadoop 3.0 is 128 MB and each erasure code policy will have *m* data blocks and *n* parity blocks. We consequently derive an algorithm that would be used for determining the optimal policy to be applied for any given folder in terms of minimising storage overhead.

Given that RS(10,4) is the policy that can hold the most data, we present the first of DECPA-F whereby it considers folders with less than 10 data blocks * 128 MB (size of one block), that is a total 1280 MB.

**Algorithm of DECPA-F (First)**

```
if (file_size <  2.5*blocksize ){
        Use Replication (3 replica) - default
} else {
        if (2.5* blocksize < file_size < 3* blocksize){
            Use RS(3,2)
        } else {
            If(3* blocksize < file_size < 6* blocksize){
                Use RS(6,4)
            } else {
                if (6* blocksize < file_size < 10*128){
                        Use RS(10,4)
                }
            }
        }
}
```

Fig 3 – DECPA-F

The algorithm DECPA-F, is used if an object is of the size less than 1280 MB, that is will fit in 10 blocks or less. For file sizes less than 320 MB, simple 3-way Replication is recommended as it is the default storage redundancy method in Hadoop 3.0.0. For other file sizes, EC policies are applied as per DECPA-F. In Figure 3, only the 3 default EC policies are shown. However, since we have also implemented RS(4,3), RS(5,3), RS(7,4), RS(8,4)and RS(9,4), they are also incorporated in the DECPA-F algorithm.

## *B. Algorithm of DECPA-M (Multiple)*

```
if (file_size > 10* blocksize) {
        //find RS that provides smallest storage overhead
        overhead = 0;
        For (n=5, n++, n<=14){
            exclude n = 6;
            n_overhead = modulus(n* blocksize, file_size)
            if(n_overhead < overhead){
                overhead=n_overhead;
                no_of_block=n;
            }
        }
        Apply RS(n,m);
}
```

Fig 4 – DECPA-M

The algorithm DECPA-M, is used if an object is of the size more than 1280 MB, that is, it will require more than one stripe. In DECPA-M, the amount of overhead that will be produced, is used as criterion, to apply the required EC policy. The optimal EC policy is chosen based on minimal overhead generated.

## *C. Algorithm of DECPA-MF (Multiple RS(10,4) and 1 DECPA-F)*

```
if (file_size > 10*128) {
        //use RS(10,4) for x stripes n=10
        stripes = modulus(n*128, file_size)
        for(stripes){
                Apply RS(10,4)
        }
        remaining = file_size – stripes*(n*128)
        Apply DECPA-F using remaining;
}
```

Fig 5 – DECPA-MF

The algorithm DECPA-MF, is used if an object is of the size more than 1280 MB, that is it will require more than one stripe. In DECPA-MF, the max amount of RS(10,4) stripes are created, and the remaining is encoded using the DECPA-F algorithm.

## V. Experimental Setup

This section provides the details of the setup used for the performance testing as well as evaluation of implemented algorithm.

The cluster consists of 14 nodes with 1 node acting as the master (namenode) and the remaining 13 nodes acting as the workers (datanodes). They communicate with each other using passwordless-ssh. The Master node is a powerful PC (4x3.2 GHz Intel Processors with 8GB of RAM and 1 TB HDD) hosting the NameNode/RaidNode and 11 HP PCs acting as Clients hosting DataNodes (each with a 3.2 GHz Intel Processor with 4 GB RAM and 500 GB HDD). The average bandwidth of this cluster is 12MB/s.

## VI. Results

For the purpose of experimentation, a file size of 1.07 GB is used so as to generate enough blocks and stripes. Firsty the performances of writing the 1.07 Gb file to cluster under different policies are presented. Secondly the repair performances to simulate one and two node/block failures are presented . And finally the effectiveness of DECPA is evaluated by comparing it to the default Hadoop redundancy implementation.

### A. *Write Performances*

As mentioned for testing purposes, a file of the size of 1.07 GB was used to evaluate the performance. Figure 6 shows the time taken to write each stripe for the different EC policies used. Since RS(7,3) is the most balanced policy in terms of blocks and parity created, it provides the minimum time for writing the same amount of data to the cluster.

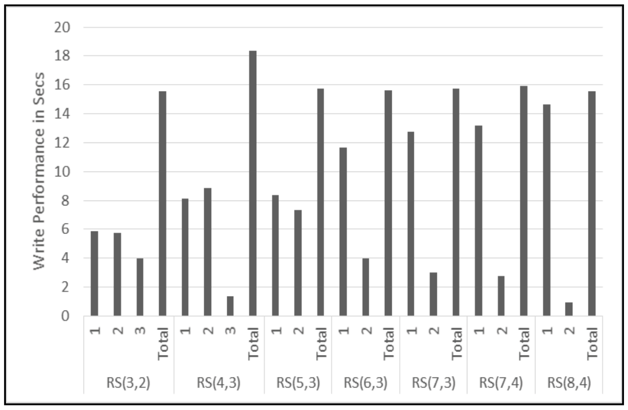

Fig 6 – Write Performances

### B. *Repair Performance*

The repair performance, as shown in Figure 7, demonstrates an exponential trend with longer stripes requiring more time. This is due to the number to blocks transmission required for the repair of 1 or more failures. Given the limited number of nodes (12), we could only perform testing for 2 repairs using the RS(7,4) EC Policy.

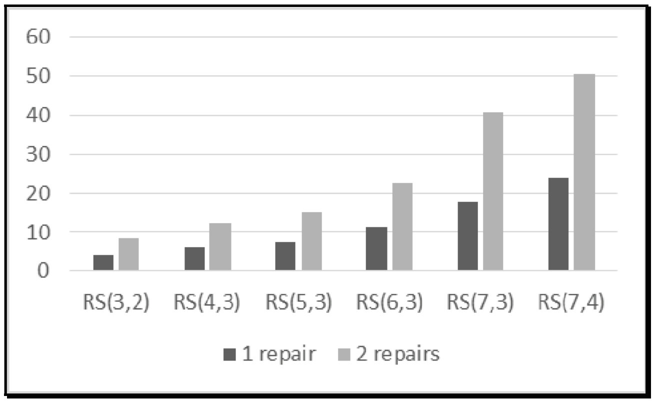

Fig 7 – Repair Performances

### C. *Efficiency of DECPA (F, M and MF)*

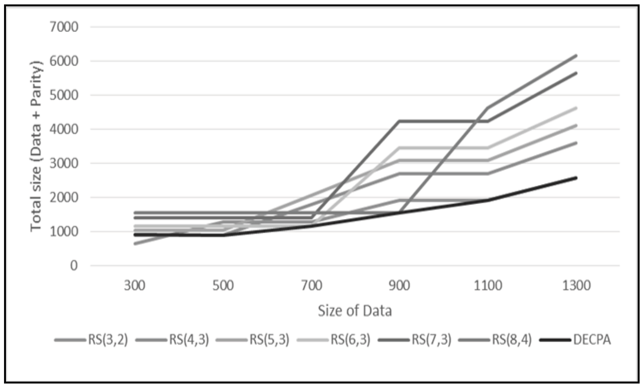

Fig 8 - DECPA-F

As mentioned earlier, DECPA-F is used where the size of data can fit in a single stripe, that is less or equal to 1280 MB. From Figure 8, it can be clearly observed that DECPA-F will generate the least amount of the overhead for any EC policies used. And when the size of the data is 1280 MB, the storage saving can be as much as 4000MB.

DECPA-M is used when multiple stripes are indeed to encode the data, that is, more than 10 blocks will be generated. It can be observed from figure 9, that DECPA-M produces the least overheads irrespective of the EC policy used or the size of data.

DECPA-MF adopts a different approach, that of using RS(10,4) stripes to cover maximum data to be encoded, and the remainder is then encoded using DECPA-F, to find the appropriate EC policy producing least overheads. Though, from figure 10, DECPA-MF achieves lesser overheads compared to DECPA-M with greater data sizes, it requires more metadata and processing. In order to implement DECPA-MF, stripes encoded with RS(10,4) are stored and EC policy is applied to it, and the rest is stored and encoded using RS(*m,n*) based on DECPA-F, in a separate folder. It is implemented in this way because EC policies in Hadoop 3.0 can only be applied to folders and not individual files.

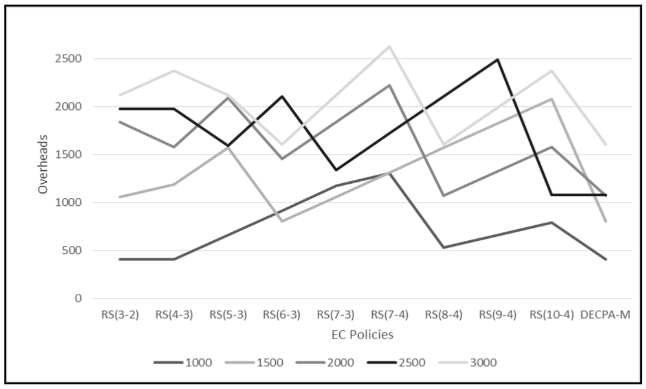

Fig 9 - DECPA-M

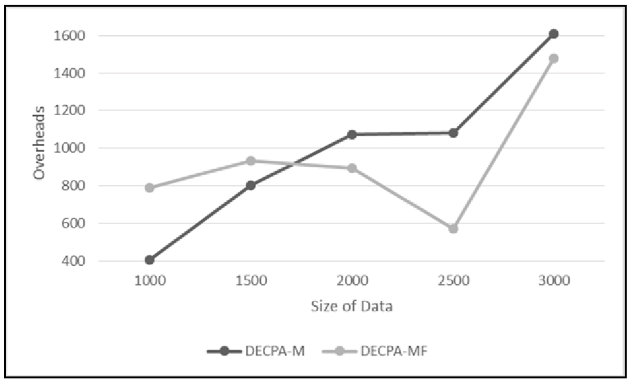

Fig 10 - Comparison of DECPA-M and DECPA-MF

## CONCLUSION

In this work, we designed and implemented DECPA, Dynamic Erasure Coding Policy Allocation, based on minimum overhead produced, in order to maximize storage capacity. Three types of allocation has been implemented, (i) DECPA-F, for file sizes less than 10 blocks, (ii) DECPA-M, for file sizes greater than 10 blocks using 1 EC policy and finally (iii) DECPA-MF, for file sizes greater than 10 blocks, using maximum RS(10,4) and RS(m.n) based on DECPA-F. We demonstrated the effectiveness of the algorithm developed. Though the savings might seem minimal (few GBs), however when the algorithm would be deployed globally, it would have a greater impact.

## ACKNOWLEDGMENT

We thank Associate Professor Anwitaman Datta, NTU, Singapore, for his scientific insight on the subject matter and constant support.

## REFERENCES

[1] Hadoop.apache.org. (2018). Apache Hadoop 3.0.0 – HDFS Erasure Coding. [online] Available at: https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html [Accessed 24 Sep. 2018].

[2] Zhe, Z., Wang, A., Zheng, K., Maheswara, U. and Vinaya, k. (2018). Introduction to HDFS Erasure Coding in Apache Hadoop - Cloudera Engineering Blog. [online] Cloudera Engineering Blog. Available at: http://blog.cloudera.com/blog/2015/09/introduction-to-hdfs-erasure-coding-in-apache-hadoop/ [Accessed 24 Sep. 2018].

[3] Chiniah, A., Dhora, J.A.D. and Sandooram, C.J., 2017, May. Erasure-Coded Network Backup System (ECNBS). In International Conference on Information, Communication and Computing Technology (pp. 35-43). Springer, Singapore.

[4] Li, J. and Li, B., 2016, June. Zebra, 2016: Demand-aware erasure coding for distributed storage systems. In 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS) (pp. 1-10).

[5] Meng, Y., Zhang, L., Xu, D., Guan, Z. and Ren, L., 2019, February. A Dynamic Erasure Code Based on Block Code. In Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks (pp. 379-383). Junction Publishing.

[6] Lee, C.W., Hsieh, K.Y., Hsieh, S.Y. and Hsiao, H.C., 2014. A dynamic data placement strategy for hadoop in heterogeneous environments. Big Data Research, 1, pp.14-22.

[7] Shen, J., Li, Y., Sheng, G., Zhou, Y. and Wang, X., 2018, October. Efficient Memory Caching for Erasure Coding Based Key-Value Storage Systems. In CCF Conference on Big Data (pp. 512-539). Springer, Singapore.

[8] Chen et. al. "Giza: Erasure Coding Objects across Global Data Centers", USENIX ATC 2017

[9] Wei, Q., Veeravalli, B., Gong, B., Zeng, L. and Feng, D., 2010, September. CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster. In Cluster Computing (CLUSTER), 2010 IEEE International Conference on (pp. 188-196). IEEE.

[10] Cheng, Z., Luan, Z., Meng, Y., Xu, Y., Qian, D., Roy, A., Zhang, N. and Guan, G., 2012, September. Erms: An elastic replication management system for hdfs. In Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on (pp. 32-40). IEEE.

[11] Masur, R.G. and Mcintosh, S.K., 2017, August. Preliminary performance analysis of Hadoop 3.0. 0-alpha3. In Scientific Data Summit (NYSDS), 2017 New York (pp. 1-3). IEEE.

[12] Shubham, S. (2018). Hadoop 3 | What's New in Hadoop 3.0 | Hadoop 3 Enhancements | Edureka. [online] Edureka Blog. Available at: https://www.edureka.co/blog/hadoop-3/ [Accessed 24 Sep. 2018].

[13] Hadoop.apache.org. (2018). Apache Hadoop 2.9.1 – Rack Awareness. [online] Available at: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/RackAwareness.html [Accessed 23 Sep. 2018].

[14] Apache (2018). Apache Hadoop 3.0.0-alpha1-SNAPSHOT – HDFS Erasure Coding. [online] Iwasakims.github.io. Available at: http://iwasakims.github.io/HDFS-9884/hadoop-project/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html [Accessed 3 Oct. 2018].

[15] Chiniah, A., Esmaili, K.S. and Datta, A., 2013, October. Efficient updates in cross-object erasure-coded storage systems. In 2013 IEEE International Conference on Big Data (pp. 28-32). IEEE.

[16] Chiniah A. and Einstein M.U.A. (2019) HIVE-EC: Erasure Code Functionality in HIVE Through Archiving. In: Advances in Information and Communication Networks. FICC 2018. Advances in Intelligent Systems and Computing, vol 887. Springer,Cham.