DESIGN AND DEVELOPMENT OF ERASURE-CODED DATA ARCHIVAL USING ENHANCED GREEDY BLOCK FAILURE RECOVERY ALGORITHM ON SPARK RDD FRAMEWORK

DHANAPRIYA L [1]
M.Phil Research Scholar, Department of Computer Science,
Sri Ramakrishna College of Arts and Science for Women,
Coimbatore, India.

Dr. V. UMARANI [2]
Associate professor, Department of Computer Science,
Sri Ramakrishna College of Arts and Science for Women,
Coimbatore, India.

Abstract: Big-data systems allow storage and examination of huge amounts of data, and are fuelling the data revolution that is impacting roughly all walks of human undertaking today. The establishment of any big-data system is an extensive, circulated, data storage system. To increase in the huge amount of data to be read and transferred during recovery for an erasure-coded system results in two major problems: high degraded read latency and longer reconstruction time. To overcome these problems, this paper presents an enhanced method of Erasure-Coded Data Archival System for Hadoop Clusters using Enhanced Greedy Block Failure Recovery Algorithm (EGBFRA), which combines Spark framework and load balancing strategy to solve the optimization problem during Big data processing. The proposed system implemented using Spark 1.6 with Hadoop 2.6 platform under goes MapReduce framework. According to the EGBFRA strategy it can work with the large file database for finding the mining data block failures. The results shown that the performance of the new scheme is effective compared with other aHDFS algorithms. As the Experimental results show, Enhanced Greedy Block Failure Recovery Algorithm clearly outperforms HDFS-EC and aHDFS.

Keywords: Hadoop distributed file system, Spark, erasure codes, Greedy search.

I. INTRODUCTION

In data mining, Big Data is a term used to illustrate a group of data that is enormous in size and however rising exponentially with time. In short such data is so huge and composite that none of the established data management tools are able to store it or process it efficiently. “Big Data” is any attribute that challenges the constraints of computing systems. Gartner [1-2] refers as the big data with the “3V”: high volume, high velocity and high variety data that need a new way for being processed. So we can say that big data is not only big for size, but also big for the high volume of data generators (i.e. a small quantity generated by a big variety of sources).

Disk-based archival storage systems are a handful of studies focusing on data archival techniques to save data storage space. With the rapid growth of data volume in many enterprises, large-scale data processing becomes a challenging issue, attracting plenty of attention in both the academic and industrial fields. Many research approaches typically produce possible ranges or hash partitions, which are then evaluated using heuristics and cost models.

While fixing multiple failures is not uncommon in the distributed storage system [3] or even is intended, we can design erasure codes that reconstruct data from multiple failures in batches rather than separately, such that even less network transfer will be incurred than MSR codes. Meanwhile, we can significantly save disk I/O as we can read existing blocks only once instead of multiple times. However, while the optimal network transfer to reconstruct multiple blocks has been theoretically established, there has been no explicit construction of erasure codes that achieve both the optimal network transfer to reconstruct exact data of multiple failed servers and the optimal storage overhead simultaneously, except for those that impose strict constraints on system parameters.

The erasure code replication technology provides redundancy by simply creating copies of original data; erasure coding technology encodes the data. Specifically, given an object, erasure coding system first divides it into d fragments, then produces another e fragments based on generator matrix, and these $d + e$ fragments are defined as an erasure coding set. $d/d+e < 1$ gives the encoding rate of the set. One of the most attractive features of erasure coding is that any d fragments in one set can be used for reconstructing the original d fragments. To maximize fault tolerance, fragments of a set are stored in different nodes.

Figure 1 presents examples of encoding and decoding processes with $d = 3$ and $e = 2$. Erasure codes are a superset of replicated and Redundant Array of Inexpensive Disks (RAID) systems. When $d = 1$ and $e = 2$, three replicas will be created for a block, which is the same as the replication. RAID 4 can be described as erasure coding with $d = 4$, $e = 1$, while RAID6 means $e = 2$.

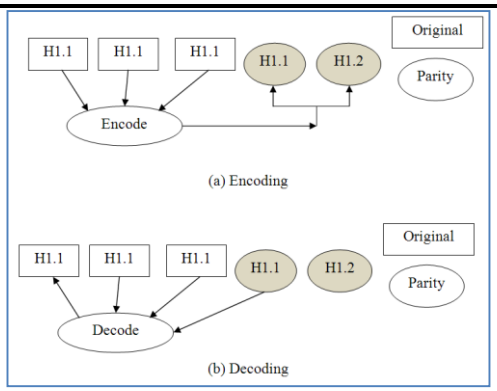

Fig. 1: Erasure coding with d=3 and e=2

In this paper presents an Erasure-Coded Data Archival System for Hadoop Clusters using Enhanced Greedy Block Failure Recovery Algorithm (EGBFRA), which combines Spark framework and load balancing strategy to solve the optimization problem during big data processing.

The main contributions of this paper are as follows:

- Evaluate the EGBFRA in spark-hadoop-apache framework will improve performance in iterative and interactive analytics approach by reusing data across multiple parallel operations.
- Analyze the performance measures of running time, shuffling cost, and mining cost with existing techniques.

The rest of the paper is organized as follows: Related work is detailed in Sect. 2. In Sect. 3, Research methodology in Sect. 3, Experimental results are described in Sect. 4; finally conclusion is in Sect. 5.

II. RELATED WORK

(*B. Calder, et al.*, **2011**) [5] Windows Azure Storage (WAS) is a cloud storage system that provides customers the ability to store seemingly limitless amounts of data for any duration of time. WAS customers have access to their data from anywhere at any time and only pay for what they use and store. In WAS, data is stored durably using both local and geographic replication to facilitate disaster recovery. Currently, WAS storage comes in the form of Blobs (files), Tables (structured storage), and Queues (message delivery). The authors described the WAS architecture, global namespace, and data model, as well as its resource provisioning, load balancing, and replication systems.

(*D. Borthakur*, **2012**) [6] author discussed a Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

(*M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly*, **2013**) [7] authors discussed a Quantcast File System (QFS) is an efficient alternative to the Hadoop Distributed File System (HDFS). QFS is written in C++, is plug-in compatible with Hadoop MapReduce, and offers several efficiency improvements relative to HDFS: 50 per cent disk space savings through erasure coding instead of replication, a resulting doubling of write throughput, a faster name node, support for faster sorting and logging through a concurrent append feature, a native command line client much faster than hadoop *fs*, and global feedback-directed I/O device management. As QFS works out of the box with Hadoop, migrating data from HDFS to QFS involves simply executing hadoop dist *cp*.

(*J. Wang, P. Shang, and J. Yin*, **2014**) [8] authors developed a new Data-gRouping-AWare (DRAW) data placement scheme to address the above-mentioned problem. DRAW dynamically scrutinizes data access from system log files. It extracts optimal data groupings and re-organizes data layouts to achieve the maximum parallelism per group subjective to load balance. By experimenting two real-world MapReduce applications with different data placement schemes on a 40-node test bed, we conclude that DRAW increases the total number of local map tasks executed up to 59.8 per cent, reduces the completion latency of the map phase up to 41.7 per cent, and improves the overall performance by 36.4 per cent, in comparison with Hadoop's default random placement.

(*M. Xia, M. Saxena, M. Blaum, and D. A. Pease*, **2015**) [9] authors presented Hadoop Adaptively-Coded Distributed File System (HACFS), a new erasure-coded storage system that instead uses two different erasure codes and dynamically adapts to workload changes. It uses a fast code to optimize for recovery performance and a compact code to reduce the storage overhead. A novel conversion mechanism is used to efficiently upcode and downcode data blocks between fast and compact codes. We show that HACFS design techniques are generic and successfully apply it to two different code families: Product and LRC codes.

(*Yuanqi Chen, et al.*, **2017**) [10] proposed an erasure-coded data archival system called aHDFS for Hadoop clusters, where RS(k + r, k) codes are employed to archive data replicas in the Hadoop distributed file system or HDFS. They developed two archival strategies (i.e., aHDFS-Grouping and aHDFS-Pipeline) in aHDFS to speed up the data archival process. aHDFS-Grouping - a MapReduce-based data archiving scheme - keeps each mapper's intermediate output Key-Value pairs in a local key-value store.

III. RESEARCH METHODOLOGY

In this paper, proposed Erasure-Coded Data Archival System for Hadoop Clusters using Enhanced Greedy Block Failure Recovery Algorithm (EGBFRA), which combines Spark framework and load balancing strategy to solve the optimization problem during big data processing in Java JDK 1.8.0_20 and Spark 1.6.0 with inbuild Hadoop 2.6 framework [4]. In order to know the methodology of EGBFRA in System Model, Grouping Strategy, Pipeline Archiving and Mapreduce-Based Pipelined Data Archiving is performed. The proposed work flow diagram is described in figure 2.

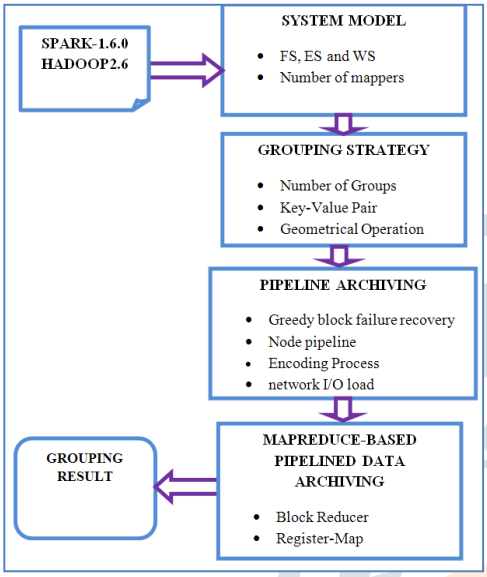

Fig. 2: Proposed flow diagram

A. SYSTEM MODEL

The system model assumes a homogenous Hadoop cluster, which is composed of a Namenode and a few Datanodes. As depicted in Figure 3, the Datanodes are grouped into three sets: Fundamental Set (FS), Extended Set (ES), and Waiting Set (WS). FS is similar to the covering subset (CS) which stores only one copy of all data blocks. Redundant data blocks are generated following a redundancy mechanism, and they are stored in ES nodes. The size of ES, i.e., the number of Datanodes contained in ES, is proportional to that of FS and we store approximately the same number of blocks in each node of FS and ES.

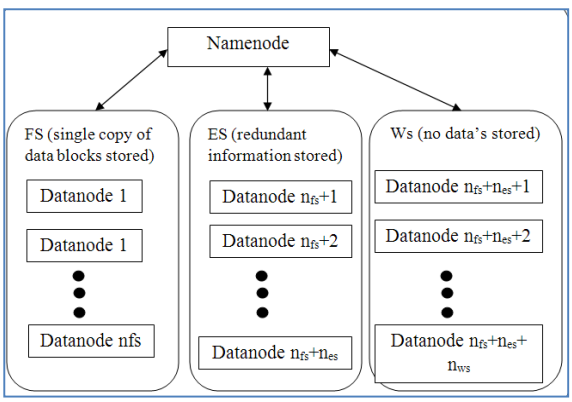

Fig. 3: System Model

This is because a same amount of redundancy is maintained for every unique block. Datanodes in WS are not assigned to store any data and they are used as backup nodes for failure recovery. The size of WS is flexible and will not be discussed in this thesis. When there is no node failure, only nodes in FS are required to be active all the time to respond to data requests while nodes in ES and WS can be turned off to save energy.

B. GROUPING STRATEGY

The grouping or clustering strategy is MapReduce-based method grouping intermediate output from the multiple mappers (i.e., mappers 1 - n). A conventional wisdom is to deliver an intermediate result created by each mapper to a reducer through the shuffling phase. To optimize the performance of our parallel archiving scheme, we group multiple intermediate results sharing the same key into one Key-Value pair to be transferred to the reducer. During the course of grouping, of course, the XOR operations are performed to generate the value in the Key-Value pair. The advantage of our new group strategy makes it possible to shift the grouping and computing load traditionally handled by reducers to mappers.

In grouping strategy default, in HDFS, replication approach is used for maintaining data redundancy. Accordingly, the case where r copies of all data blocks are stored in ES nodes. As mentioned, to divide all the Datanodes into three sets, FS, ES, and WS, and the size of ES is proportional to that of FS. If a total number of m unique data blocks are stored in n FS nodes, then r - m data blocks are maintained in r - n ES nodes. In both FS and ES, data blocks are randomly distributed and stored and replicas of a data block are kept in different nodes. Thus, on average, a FS/ES node gets around k = m/n data blocks.

C. PIPELINE ARCHIVING

In pipeline archiving, when there is a FS node failure, only partial rather than all data blocks get lost. To restore a few blocks, it is unnecessary to activate all Datanodes in ES. In order to achieve an energy-efficient failure recovery, desire to minimize the number of nodes to be turned on during the recovery process. This energy-efficient failure recovery reduces to the set covering problem because we need to identify the smallest number of sets (i.e., sets of data blocks in nodes) whose union contains all lost data blocks. Since the set covering problem is NP-hard, the research work develops an enhanced greedy algorithm to solve it. With this enhanced greedy failure recovery algorithm, one replica of each lost data block will be found and sent to a newly activated WS node for restoration. Algorithm 1 shows the details of the energy-efficient failure recovery algorithm when we employ replication as the redundancy technique in Extended Set. The algorithm uses a "pipeline archive" function to identify

common blocks shared by two nodes. The function pipeline archive is as follows:

Function 1: Pipeline Archive

Input: *x*[*m*]: an array that records data blocks stored in the failed node, *y*[*m*]: an array that records data blocks stored in an *ES* node

Output: frequent blocks: the number of common blocks of the two nodes

Process:

```
Pipeline archive (boolean x[m]; boolean y[m])
        {
        int frequent blocks = 0;
        for(int i = 0; i < m - 1; i + + )
        if(x[i] == TRUE && y[i] == TRUE)
        //The block exists in both node a and b
common blocks++;
        return frequent blocks;
        }
```

End Procedure

In Algorithm1, firstly, we record the lost data blocks in an array failed node and then we calculate the number of unavailable data blocks (lost blocks) caused by the failure (lines 1-8). To recover the lost blocks, all unmarked ES nodes are examined to identify the node (indexed by max row) that contains the largest number of replicas for lost data blocks (lines 10-21). That node is then marked and activated to send out the corresponding data copies (common blockList). Specifically, the data copies are transferred to an activated node in WS. After the transmission, the activated ES node will be shut down automatically to save energy. This process continues until all lost blocks are recovered. failed node, lost blocks are updated accordingly to reflect the blocks remained to be recovered (lines 23-27).

ALGORITHM 1: GREEDY FAILURE RECOVERY WITH REPLICATION

Input: *dataFS*[*n*][*m*]: a two-dimensional boolean array that records the data placement information of the n Datanodes in FS, which can be got from Namenode. dataFS[i] represents the data placement information of the i^{th} node. If the jth block in is stored in the ith node, dataFS[i][j] = TRUE, otherwise dataFS[i][j] = FALSE.

Input: dataES[$r \times n$][*m*]: similar to dataFS, dataES records the data placement information of ES nodes.

Process: *block failed:* the index of the failed node in FS, where $0 \leq$ failed $<$ n.

Step 1: boolean failed node[ ] = *dataFS*[*block failed*]
Step 2: int lost blocks = 0
Step 3: List **Pipeline Archive** blockList
Step 4: for index = 0 → m -1 **do**
Step 5: if *bock failed node*[*index*] == TRUE **then**
Step 6: lost blocks + +
Step 7: end if
Step 8: end for
Step 9: Turn on a WS node
Step 10: while *lost blocks*! = 0 **do**
Step 11: common_blocks = 0
Step 12: max_row = 0
Step 13: for i = 0 → r × n - 1 **do**
Step 14: if the i_{th} ES node is marked **then**
Step 15: Continue
Step 16: end if
Step 17: Result = *Pipeline archive* (*failed node*, *dataES*[*i*])
Step 18: if Result > *common_blocks* **then**
Step 19: common blocks = Result
Step 20: *max row* = *i*
Step 21: end if
Step 22: end for
Step 23: if common blocks > 0 then
Step 24: lost blocks- = common blocks
Step 25: *common blockList:empty()*
Step 26: endif
Step 27: endwhile

D. MAPREDUCE-BASED PIPELINED DATA ARCHIVING

The greedy pipeline data archiving strategy is an extension of the parallel data archiving scheme proposed in previous section. In the mapreduce-based pipelined archiving strategy, the last mapper writes key-value pairs to its subsequent node's local key-value store rather than a reducer. The last mapper in each node propagates key-value pairs to the node's subsequent node's key-value store. The last node in an archiving pipeline has no subsequent node, it is the last node's responsibility to write a parity block to HDFS.

IV. RESULT AND DISCUSSIONS

The proposed research work evaluate the performance of An Erasure-Coded Data Archival System for Hadoop Clusters using enhanced greedy block failure recovery algorithm (EGBFRA) in Spark 1.6.0 Hadoop 2.6 cluster equipped with data nodes. Each node has an Intel I5-6500 series 3.20 GHz 4 core processor, 8GB main memory, and runs on the Windows operating system, on which Java JDK 1.8.0_20 and Spark1.6.0 with inbuild Hadoop 2.6 are installed. To evaluate the performance of the proposed EGBFRA, we generate synthetic datasets using the IBM Quest Market-Basket Synthetic Data Generator [11], which can be flexibly configured to create a wide range of data sets to meet the needs of various test requirements. The research work compared the performance of EGBFRA, HDFS-EC and aHDFS [11] when the number k of pivots varies from 20 to 180. In this experiment, performance measures reveals the running time, shuffling cost, and mining cost of EGBFRA, HDFS-EC and aHDFS processing the 4G 61-block T40I10D dataset on the 8-node cluster.

To evaluate the performance of map execution time of the testing datasets.

$$Map\ Exection\ Time = \frac{Total\ time\ taken\ by\ all\ Map\ tasks}{Number\ of\ Map\ Tasks}$$

Table 1: Comparison of Map Execution time with existing HDFS-EC, aHDFS and proposed EGBFRA

Methods	1k	8k	64k	512k	4m
HDFS-EC	81	40	35	26	25
aHDFS	64	28	20	18	17
EGBFRA	**20**	**21**	**19**	**16**	**15**

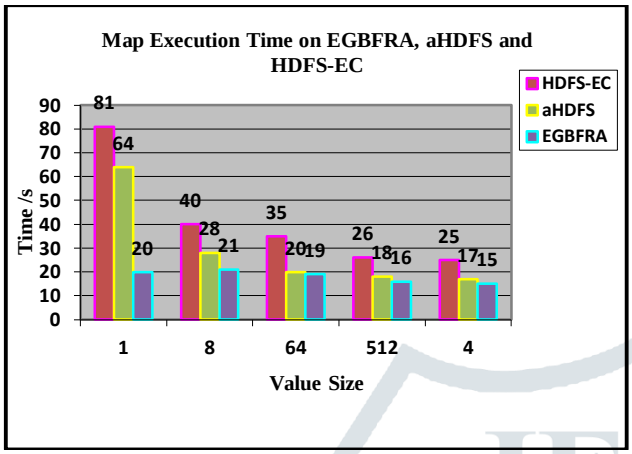

Fig. 4: Archival Map Execution time

To evaluate the performance of reduce time of the testing datasets.

$$Reduce\ Time = \frac{Total\ time\ taken\ by\ all\ Reduce\ tasks}{Number\ of\ Reduce\ tasks}$$

Table 2: Reduce Time measures with existing HDFS-EC, AHDFS and proposed EGBFRA

Methods	1k	8k	64k	512k	4m
HDFS-EC	45	42	38	35	41
aHDFS	15	18	19	12	17
EGBFRA	**14**	**15**	**13**	**11**	**14**

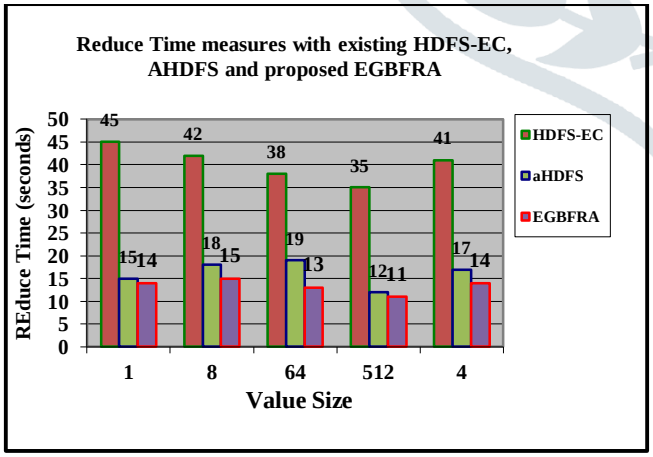

Fig. 5: Archival Reduce time

Shuffle Time: The expected shuffle cost of the parallel clustering approach is a function of the number of reducer's r to receive the data and the amount of data to be shuffled s, which is given by:

$$\text{Shuffling Time (r, s)} = \frac{s\,.\,D_r}{r}.\frac{1}{N_s}$$

The majority of the shuffling cost is related to shipping data between distinct machines through the network. Whenever possible, spark Mapreduce minimizes the cost by assigning reduce tasks to the machines that already have required data in local disks. D_r is the ratio of data actually shipped between distinct machines relative to the total amount of data processed. Thus, the total amount of data be shipped is s.D_r bytes. The data will be received in paralleled by r reducers, each one receiving in average N_s byters per second.

Table 3: Shuffling Time with existing HDFS-EC, AHDFS and proposed EGBFRA

Methods	1k	8k	64k	512k	4m
HDFS-EC	100	92	90	87	85
aHDFS	58	10	12	17	18
EGBFRA	**42**	**9**	**10**	**15**	**17**

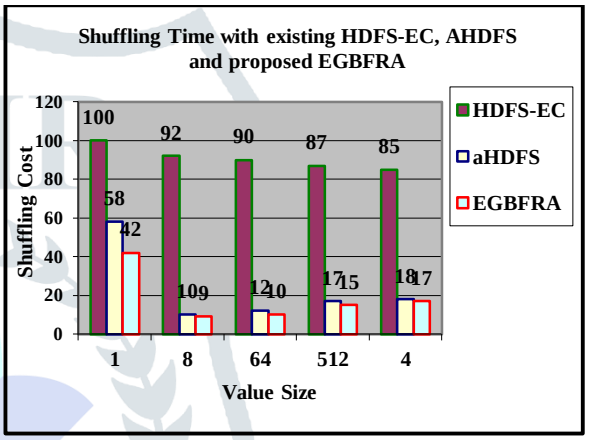

Fig. 6: Archival Shuffling time

V. CONCLSUION

In this paper, presents an enhanced method of Erasure-Coded Data Archival System for Hadoop Clusters using Enhanced Greedy Block Failure Recovery Algorithm (EGBFRA), which combines Spark framework and load balancing strategy to solve the optimization problem during Big data processing. In the EGBFRA model presents a data reconstruction system to deal with block failure issues on Hadoop clusters. The proposed technique that iteratively computes the optimal solution with the help of a big dataset program solves. Our scheme is implemented using Spark 1.6 with Hadoop 2.6 platform under goes MapReduce framework. According to the EGBFRA strategy it can work with the large file database for finding the mining data block failures. The results shown that the performance of the new scheme is effective compared with other aHDFS algorithms. As the Experimental results show, Enhanced Greedy Block Failure Recovery Algorithm clearly outperforms HDFS-EC and aHDFS.

REFERENCES

[1] Gartner says solving 'big data' challenge involves more than just managing volumes of data.

[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, May 2010, pp. 1–10.

[3] D. Ford, et al., "Availability in globally distributed storage systems," in Proc. 9th USENIX Symp. Operating Syst. Des. Implementation, 2010, pp. 61–74.

[4] D. Borthakur, "The hadoop distributed file system: Architecture and design, 2007," Apache Software Foundation, Forest Hill, MD, USA, 2012.

[5] B. Calder, et al., "Windows azure storage: A highly available cloud storage service with strong consistency," in Proc. 23rd ACM Symp. Operating Syst. Principles, 2011, pp. 143–157.

[6] D. Borthakur, "The hadoop distributed file system: Architecture and design, 2007," Apache Software Foundation, Forest Hill, MD, USA, 2012.

[7] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly, "The quantcast file system," Proc. VLDB Endowment, vol. 6, no. 11, pp. 1092–1101, 2013.

[8] J. Wang, P. Shang, and J. Yin, "Draw: A new data-grouping-aware data placement scheme for data intensive applications with interest locality," in Cloud Computing for Data-Intensive Applications. Berlin, Germany: Springer, 2014, pp. 149–174.

[9] M. Xia, M. Saxena, M. Blaum, and D. A. Pease, "A tale of two erasure codes in HDFS," in Proc. 13th USENIX Conf. File Storage Technol., 2015, pp. 213–226.

[10] Yuanqi Chen, et al., "aHDFS: An Erasure-Coded Data Archival System for Hadoop Clusters," IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 28, NO. 11, NOVEMBER 2017.ACM SIGMOD Int. Conf. Manage. Data, 2017, pp. 1013–1020.

[11] L. Cristofor, "ARtool: Association rule mining algorithms and tools," 2006.