University Library



A gateway to Melbourne's research publications

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Mohan, LJ;Caneleo, PIS;Parampalli, U;Harwood, A

Title:
Geo-aware erasure coding for high-performance erasure-coded storage clusters

Date:
2018-02-01

Citation:
Mohan, L. J., Caneleo, P. I. S., Parampalli, U. & Harwood, A. (2018). Geo-aware erasure coding for high-performance erasure-coded storage clusters. Annales Des Telecommunications Annals of Telecommunications, 73 (1-2), pp.139-152. https://doi.org/10.1007/s12243-017-0623-2.

Persistent Link:
https://hdl.handle.net/11343/283018

Geo-aware erasure coding for high performance erasure coded storage clusters

Lakshmi J Mohan*, Pablo Ignacio Serrano Caneleo†, Udaya Parampalli‡ and Aaron Harwood§
Department of Computing and Information Systems,
University of Melbourne, Australia
Email: *ljagathamma@student.unimelb.edu.au, †pabloserranocaneleo@gmail.com,
‡udaya@unimelb.edu.au, §aharwood@unimelb.edu.au

***Abstract*—Erasure code based distributed storage systems are increasingly being used by storage providers for big data storage since they offer same reliability as replication with a significant decrease in the amount of storage required. But, when it comes to a storage system with data nodes spread across a very large geographical area, the node's recovery performance is affected by various factors that are both network and computation related. In this paper, we present a XOR-based code supplemented with the ideas of parity duplication and rack awareness that could be adopted in such storage clusters to improve the recovery performance during node failures and compare it with popular implementations of erasure codes, namely Facebook's Reed Solomon codes and XORBAS local recovery codes. The code performance along with the proposed ideas are evaluated on a geo-diverse cluster deployed on the NeCTAR research cloud. We also present a scheme for intelligently placing blocks of coded storage depending on the design of the code, inspired by local reconstruction codes. The sum of all these propositions could offer a better solution for applications that are deployed on coded storage systems that are geographically distributed; in which storage constraints make triple replication not affordable, at the same time ensuring minimal recovery time is a strict requirement.**

I. Introduction

Large scale data storage centers that store and process big-data are a commonplace today. Most cloud service providers are building geo-distributed network of data centers that have their data nodes spanning wide geographical areas [1]. Typical examples of some of the practical scenarios are traffic monitoring system of a large city, medical diagnostic system connecting various hospitals in a country, climate monitoring and reporting system [2] and other scientific applications [3], to name a few. All these systems have storage nodes distributed across a wide geography and a centralized node acting as master that co-ordinates the storage and computation spread across data nodes. These are real applications that involve large scale distributed processing of massive volumes of data. Availability, reliability and performance efficiency are of utmost importance in these data centers.

For ensuring availability and reliability, many large scale commercial data centers, by default, rely on replication; wherein the same copy of the data is replicated and stored at multiple locations for protecting data from node failure events. Failure events are *a norm rather than the exception* in data centers. For example, in the recent years, Facebook has reported an important number of erasures occasioned by node failures, device malfunctions, scheduled maintenance, network outages and other related events [4], [5], [6]. However, replication involves a monetary cost with increased storage requirement that it brings in, and hence is highly inefficient. Erasure codes serve as a better alternate solution to replication since they offer the same reliability as compared to replication with significant decrease in the storage overhead incurred [7], [8], [4], [9], [10]. In erasure coding, a file to be stored is divided into chunks (blocks) of fixed size, and the code encodes a set of these data blocks, to create parity (code) blocks. The group of data blocks and its corresponding parity blocks is called a *stripe*. When devices fail leading to loss of blocks, the decode operation repairs the lost block by employing the surviving data and parity blocks.

However, erasure codes with an additional cost of computation requirement in the event of recovery from node failures. If replication is used for providing resiliency, the lost data can be recovered simply by copying it from one of the available replicas. But, for erasure coded storage, failed node recovery involves fetching source and parity data from the surviving nodes, resulting in a significant amount of network traffic. More precisely, it involves downloading data from a specific number of the live nodes to a worker node where the repair process is initiated, computation operation at the worker node for repairing the lost data, followed by copying the result to another node in the storage cluster, making it both a computation and network intensive operation. In geo-distributed data centres, data can be stored at any of the available data centres and can be requested to be downloaded from a data centre that is at the other end of the globe. With erasure codes, this translates to significantly larger network latency and cost involved in retrieving the data required for repair. In short, employing erasure codes in geo-distributed storage aggravates recovery performance upon node failures since the network latency and performance-related factors add to the problem, as observed and reported in our previous work [11].

*This work was supported by Data61/CSIRO.
Pablo Ignacio Serrano Caneleo† is now with Federico Santa Maria Technical University, Chile.

A. Our contribution

This paper addresses the open problem of improving repair performance of erasure coded storage nodes in a geo-diverse cluster setting. Though there has been active research on enhancing repair performance in the context of storage in recent times, all existing codes are designed for the conventional setting of co-located storage nodes. When deployed for storage on a cluster with geo-located nodes, they do not fare well in terms of repair performance as reported. Repair performance of storage erasure codes in geo-distributed context needs a different strategy of problem treatment. This work presents the following methodologies in geo-distributed clusters to improve a XOR-based code, that are observed to help improve repair efficiency:

- configuration of topology awareness
- replication of parity blocks

We improve the XOR-based code by extending it in such a way that they decrease the repair bandwidth in geo-diverse storage clusters. Furthermore, we also propose a heuristic termed *code aware* placement of blocks, embracing the design principles behind local repair codes. The results and analysis are reported with real cluster configurations set-up on the NeCTAR [12] research cloud that validate proposed methodologies in improving repair performance for large scale geo-distributed cluster settings.

The XOR-based code with our improvements is compared with the original Reed-Solomon erasure code running on Facebook's storage archives [13] developed on Apache Hadoop and a code based on the idea of local parities [14]. The experiments use a set of storage clusters distributed across diverse geographical locations across Australia on the NeCTAR research cloud for simulating a geo-distributed data center. The technique increases the storage requirement of the data center, but results in decreased recovery time and recovery bandwidth, making it a better choice for big data applications mounted on large geo-distributed data centers.

Our experiments were run on archival type test data (cold data). Hadoop's erasure code module can be used to store frequently accessed (hot) data as well. In such a context, storage is accessed more often. If failure occurs when a user accesses his/her stored data, it has to be repaired and restored to the file system storage with practically non-noticeable delays. This is technically called 'ad-hoc' repair as opposed to the usual repairs that are triggered after being noticed by the *RaidNode* daemon during its routine scan. Hence, ensuring reliability at extremely fast rates to the user becomes highly significant. There are a number of real applications where triple-replicated storage is not cost-effective and user access frequencies are more than that of conventional cold storage. For example, let us consider the example of analyzing a graph of bitcoin transactions that was derived by parsing all blocks since the genesis block in the bitcoin blockchain. The transaction graph is stored in Hadoop file system, and can get very big as the transactions grow. Keeping triple replicas is not practical; thereby, the user erasure codes the block chain data and stores it. Any node failure is to be recovered in 'ad-hoc' mode and reducing recovery time becomes critical in such practical user-facing storage processing applications.

B. Organization of the paper

The remainder of the paper is organized as follows: The related work in the literature is reviewed in the next section; followed by Section-III which presents an overview of erasure codes and the repair problem in the geo-diverse context. The next section gives a brief introduction to Hadoop and its erasure codes implementation. Section-V explains multiple XOR code implementation and the techniques supplementing it, followed by the design of the experimental storage cluster and the experiments done in the following section. The next section describes metrics reported and presents detailed analysis of the results. It also proposes and validates the heuristic that placement of blocks based on the code design would lead to better repair performance. Finally, Section-VIII concludes the paper.

II. Related Work

As indicated earlier in Section-I, classical erasure codes are sub-optimal in distributed storage environments because of the *repair problem*. Even though only a single data block is lost, recovery requires transferring all other blocks in the corresponding stripe to a certain node that will perform the recovery operation, and subsequently regenerate the lost block. This results in consumption of considerable network bandwidth and disk I/O during data recovery, sometimes referred to as recovery overhead in the literature.

Consequently, the problem of decreasing the recovery overhead in the event of node failures in erasure in coded storage systems has received considerable attention in the recent past, both in theory and practice. Binary MDS codes are popular because of their use in disk array systems and have been worked on extensively by researchers. Some examples are EVEN-ODD and RDP codes [15], [16]. These codes have been enhanced to support optimal recovery in the papers [17], [18] respectively. A coding scheme, called *pyramid codes*, to improve read performance during node repairs is presented in [19], but it incurs additional storage space than conventional schemes. In [4], the authors present a new framework with conventional Reed Solomon codes that make single node failure recovery efficient both in terms of network bandwidth and disk I/O, without incurring extra storage space.

Another popular domain of code construction is based on the idea of *reconstruction locality*, which means the number of nodes contacted during repair. There has been substantial research in recent times on code constructions that are based on locality [20], [21], [22], [23], [24]. Implementations of local repair codes in practical storage systems can be found in [7] and [5]. Local repair codes focus on achieving faster recovery with some trade-off in the storage requirement of the system.

Reducing repair latency in distributed storage systems with heterogeneous link capacities is explored in the recent work [25], based on the observation that regeneration time depends

heavily on selection of the participating nodes that help in the process. The use of erasure codes in geo-diverse storage clusters is explained in [26] for storing binary large objects. Geo-replicated XOR coding is deployed alongside with Reed Solomon codes for adding data centre fault tolerance. XOR of storage in two different data centres is computed and stored in a third data centre.

Reducing recovery overhead in coded storage systems is an extensively explored subject. However, extending it to geo-diverse storage has not been addressed in literature so far; this is the focus of our paper.

III. Erasure codes and the repair problem in geo-distributed data centers

A storage system using erasure coding has an array of n disks, each having the same size. Of these n disks, k of them hold data and the remaining m hold coding information, named *parity*, which is calculated form the data. The most popular choice are MDS (Maximum Distance Separable) codes, which have the property that if any m disks fail, the original data may be reconstructed from the remaining k disks, i.e, k out of these n nodes suffice for data recovery. In terms of the redundancy-reliability trade-off, MDS codes are optimal. Reed Solomon codes (RS) [27] are a well-known family of MDS erasure codes.

1) Galois fields: A finite field (Galois field) is a finite set of elements on which the operations additions, subtraction, multiplication and division are defined based on some fundamental rules. The term 'symbol' represents an element of a finite field. RS codes are defined over Galois Fields of size 2^w, represented by $GF(2^w)$. During encoding, a RS algorithm encodes a message of k symbols into $n = m + k$ symbols, where $n \leq 2^w$, in such a way that the original message can be reconstructed from any subset of size k of the encoded symbols. The decoding operation involves solving a set of linear equations with Gaussian elimination or matrix inversion. The CPU complexity for GF operations is expensive and many open source implementations exist that implement libraries for GF operations supporting RS coding; the most popular being Jerasure [28], based on which Hadoop RS codes are built. There is another family of codes, that is purely based only on XOR operations. These codes do not require expensive GF computation and hence perform encoding and decoding faster.

2) Methodology: The repair problem of recovering from node failures involve contacting the required number of survivor nodes based on the code design, downloading data from them, performing the decode operation and writing back the recovered data to the storage. In a geo-distributed cluster, in addition to the computation cost associated with GF operations, there are other issues related to node location and network latency. Making computation cheaper and thinking of other clever mechanisms to improve recovery performance are the solutions to tackle this issue, which is precisely what we have tried to accomplish in this work.

IV. Hadoop and its erasure codes

HDFS is Hadoop's distributed file system which is designed to store very large volumes of data reliably across storage locations and to stream those data at high bandwidth to client applications. Hadoop uses mapreduce paradigm for its computations of very large data sets. By distributing storage and computation across many nodes, the resources can grow with demand. A Hadoop cluster can scale computation capacity, storage capacity and I/O bandwidth just by adding commodity hardware. HDFS stores filesystem metadata on the namenode and application data separately on data nodes. The nodes communicate among themselves based on TCP/IP protocol. By default, the datanodes in HDFS do not rely on data protection mechanisms like RAID for reliability. The implementation of Hadoop is based on the Google File System (GFS) [29].

HDFS-RAID [8] is the module in HDFS that implements erasure codes based on Reed-Solomon codes. This module is an open source and it runs on erasure coded storage system employed in large scale production clusters at Facebook. HDFS-RAID has a list of files that are to be erasure coded and periodically performs erasure coding of these files. It also performs the recovery operation when blocks are missing or get corrupted. The recovery operation calls the decoder of the erasure code for reconstructing the file. The RAID module also handles *degraded read* requests that are redirected from HDFS, in which case the requested block is reconstructed on the fly by the daemon called *RaidNode*. The encoding and decoding operations are carried out as MapReduce jobs in the cluster.

V. Multi-XOR code

In this section, the multi-xor (MXOR) code design and its storage trade-off are explained. Thereafter, stripe design in various codes, location awareness and parity replication are discussed.

A. Code design and storage trade-off

The MXOR code is a simple code design which takes a single stripe of source blocks (amounting to 10 blocks) like XORBAS and FB erasure codes. This code is presented in [6] where it is used to build an adaptive erasure coded system that can switch between hot and cold data storage dynamically depending on the system workload. We use it here with the aim of improving the recovery performance in our geo-diverse cluster setting. These codes belong to the class of block array codes.

MXOR code rearranges ten blocks into two rows of five blocks each, as shown in Fig. 1. and computes five vertical XOR parity blocks and two horizontal parity blocks, resulting in a total of seven parity blocks. Having only XORs as parities ensures that the computation complexity involved in recovery is trivial. It is clearly evident from the design that it is very efficient in handling one node failures by only involving the download of two of the surviving nodes, using only the vertical parities. Let us assume the data node that stores the block *data8* shuts down. To recover it, the code requires only the

blocks *data3* and *parity3* to be downloaded and XORed at the recovery worker node. However, MXOR codes can handle multiple failures, by leveraging the horizontal parities with the vertical parities and fixing the location of the lost block. We have implemented MXOR codes as a new erasure code on top of Facebook's Reed Solomon and XORBAS LRC, by modifying the open-source project available at [14].

A general implementation using an (n, k) Reed Solomon code, requires the download of any k of the remaining data blocks, which makes the reconstruction process slower and inefficient. Thus, MXOR codes are ideal in improving the recovery performance. The trade-off in using this code is that it requires extra storage i.e. 1.7x storage overhead, whereas the Reed Solomon codes used in Facebook's RAID module results in 1.4x storage overhead and the XORBAS code requires 1.6x.

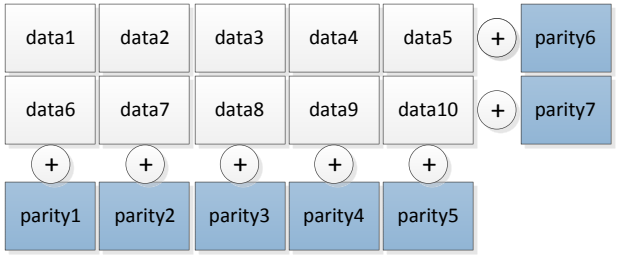


Fig. 1: Multiple XOR code design

B. Stripe design in various codes

With conventional (n, k) erasure codes, data is stored across n nodes in the network in a way that the entire data can be reconstructed by a data collector by connecting to any k nodes. A practical example is $(14, 10)$ Reed-Solomon (RS) code used in Hadoop distributed storage clusters which codes 10 data blocks by adding 4 extra parity blocks, resulting in 14 coded blocks that get stored on to different nodes on the cluster. The set of n blocks that are encoded or decoded together is called a *stripe.*

Let B be the total file size measured in terms of symbols (elements) over a finite field. RS codes treat each fragment stored in a node as a single symbol belonging to the finite field. It is known that when individual nodes are restricted to perform only linear operations, the total amount of data download needed to repair a failed node, can be no smaller than B.

In contrast, *regenerating codes* are codes over a vector alphabet and hence treat each fragment as being comprised of α symbols over the field. Linear operations here permit the transfer of a fraction of the data stored at a particular node. Apart from this new parameter α, two other parameters associated with regenerating codes. A failed node is permitted to connect to a fixed number d of the remaining nodes while downloading $\beta \leq \alpha$ symbols from each node. This process is termed as *regeneration* and the total amount $d\beta$ of data downloaded for repair purposes as the *repair bandwidth*. Typically, with a regenerating code, the average repair bandwidth is small compared to the size of the file B.

Code	Stripe blocks	Stripe size
Facebook's Reed Solomon	10 source + 4 parity blocks	14
XORBAS LRC	10 source + 6 parity blocks	16
MXOR	10 source + 7 parity blocks	17

Table I: Stripe structure in various code schemes studied

Facebooks RS code, XORBAS LRC code and MXOR code are erasure codes that consider a stripe as a simple combination of source blocks and parity blocks in a row. Table I below summarizes stripe structure for various code designs that are analysed in this paper.

Vector code designs: There is a different type of code called *MDS array code*, where each symbol is a vector or a column, and gets stored in a different node. It has r parities and can correct upto r erasures of entire columns. Fig. 2 shows a $(6, 4)$ code with column length $l = 2$ and number of parities $r = 2$. There are four storage nodes that store data blocks and two parity nodes that store code blocks. These codes have the *optimal repair property*, meaning that to repair a node, only a fraction of $1/r$ data needs to be transmitted. This fraction is computed based on multiplying the code matrices with the repair matrices, that are carefully designed.

N1	N2	N3	N4	P1	P2
a	*b*	*c*	*d*	*a + b + c + d*	*2a + w + 2b + 3c + d*
w	*x*	*y*	*z*	*w + x + y + z*	*3w + b + 3x + 2y + z*

Fig. 2: Long MDS code (n=6,k=4,l=2)

In this code example, to repair the loss of $N1$, we transmit the first row from all other remaining nodes. To repair $N2$, transmit the second row from all other nodes. To repair $N3$, we need to transmit the sum of both rows of all other nodes. And to repair N4, we transmit the sum of the first row and 2 times the second row from nodes $N1, N2, N3, P1$ and the sum of the first row and 3 times the second row from node $P2$. The coefficients to be multiplied for the computation and the data to be transmitted are based on the repair matrices that are built for the code design.

The stripe definition here would have to take into consideration all the symbols of the vector from all the corresponding columns. At times, we need to perform some computation on a part of data that is read from a node and then use the resulting value in the recovery process. This idea is popularly referred to by the term *sub-packetization* in the coding theory community.

C. Harnessing the topology awareness

Network topology plays a critical role in clusters that are geographically distributed, because repair performance depends on how close the surviving node is to the recovery worker node. In the experimental test cluster, we have data nodes spanning three locations. We can make Hadoop aware of this geographical assignment by specifying the cluster nodes as belonging to separate racks via a script and making the corresponding changes in the configuration files. The steps to perform this are detailed in the Appendix.

Without the rack awareness, all data nodes in the cluster are treated by Hadoop as belonging to a single location; thereby placing the block replicas of the file randomly. But, according to the default placement policy, the first replica of a block is to be placed in the local rack (where the client data writer node runs), second replica in another node belonging to a different rack and the third replica is to be placed in another node in the same rack where the second replica was placed.

Topology awareness is harnessed by making Hadoop aware of geographically distributed nodes in its cluster. This is done by introducing location-awareness, which in turn leads to the locations treated as separate racks in the cluster. After this modification, block placement happens exactly as per the default placement policy. We have made use of this location awareness of Hadoop in storing extra copies of parities to suit our requirement of improving the recovery performance, as explained in the next sub-section.

D. Parity duplication and the trade-offs

Hadoop, similar to other distributed storage systems, relies on replication as the primary method of ensuring fault-tolerance [30]. We extend this analogy to replicating parity blocks with the idea that having more copies of parities will increase the chances of locality in our geo-diverse cluster setting. We store two replicas of parities aiming to bringing down the time taken for recovery from node failure. The trade-off is the storage overhead incurred by having an additional replica of the parity blocks. The resulting storage overhead for various erasure codes that have been evaluated in our work is shown in Table. II.

Code	Replication	Storage Overhead
Facebook's Reed Solomon	2	1.8x
XORBAS LRC	2	2.2x
MXOR code	2	2.4x

Table II: Impact of Double Replicating parities in various codes

In XORBAS and MXOR, despite increasing the storage overhead, this can bring huge benefits in our cluster by ensuring locality; the scenario when the parity replica is available at the location where the repair worker node is assigned by Hadoop to carry out the recovery process. Having an extra replica of parity blocks leads to higher chances of contacting a node belonging to the same location as the worker node; thereby resulting in locality which in turn leads to faster recovery from failure. We monitored this in our test data cluster and our results confirm that parity replication brings the recovery times down in our cluster.

In the case of MXOR, since only two blocks are needed for a single node failure recovery, parity duplication further enhances the performance of reconstruction if worker and the required blocks belong to the same location; which is a highly probable scenario. Increasing the parity replicas beyond 2 seems not to be a viable option, since it increases the storage requirement close to or more than triple replication; and thereby losing the benefit gains of using codes for storage.

However, in a data center with nodes belonging to more number of locations, increasing parity replicas will certainly improve the recovery performance but at the cost of increasing storage requirement.

VI. Design and set up of the Experimental storage cluster

The experimental cluster is built on the national research cloud, NeCTAR (National e-Research Collaboration Tools and Resources). It is a federated Australian Research Cloud which partners with Australian universities and research institutions to create a national cloud for Australian researchers. It is located at eight different organisations (availability zones) around Australia, operating as one cloud system. Our test cluster has 45 data nodes spread across Australia, at three locations namely Tasmania, Queensland and Perth. These are controlled by a single master node that is located at Tasmania zone. The node locations with the average ping times from the master node in Tasmania to the three data node locations are shown in Fig. 3.

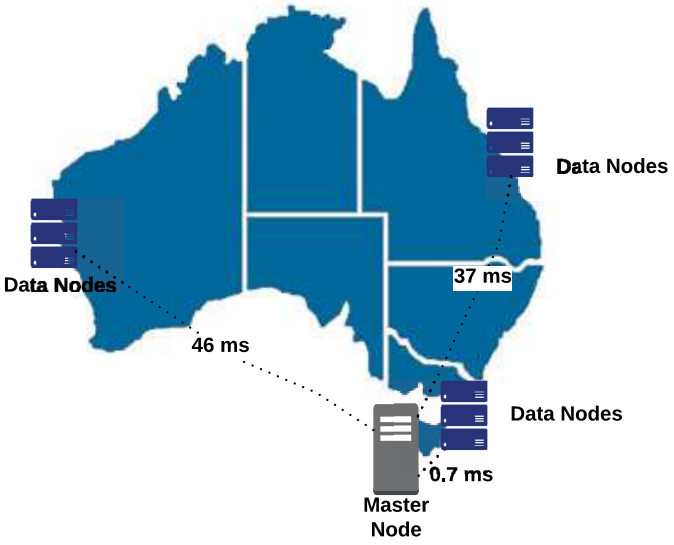


Fig. 3: Node locations along with ping times from the master node in the experimental cluster

The instances run on virtual machines of type m1.small which are machines with 1 core, 4GB RAM, 30GB hard disk. The framework and the bash scripts developed as part of our previous work [11] are used here for setting up the cluster and

automation of the test execution process. All node failures are simulated by stopping the datanode manually. This is done by ssh-logging to the datanode and running the bash script 'stop-datanode.sh' from its terminal. Given below are the steps followed in the experiments:

1) Upload files triple-replicated (default case)
2) Enable RAID for the file path and wait for parity blocks to be generated, extra replicas to be deleted and block movements to prevent co-located placements to be completed
3) Simulate single/double node failure (node to fail is chosen randomly) by stopping data node script on the node manually
4) Record repair time taken for repairing the lost block
5) Iterate the steps (1-4) twenty times; for all three codes, with/without parity replication and with/without location awareness

VII. Results and Analysis

The experimental results are focussed only on single node and double node recovery scenarios. Single node failure is the most common one in practice, followed by double node failures. It is reported in [31] that 98.08% of all repairs involve recovering a single block in a stripe.

A. Metrics of interest

The primary metric used for evaluating the recovery performance is repair time in milliseconds taken by the recovery worker to perform the repair, which is the total of decode time and wait time. Repair time is the metric used very commonly for measuring the decode performance of erasure codes [32], [4], [6], [5]. We know that to repair a block, some surviving blocks to be read and downloaded. HDFS-RAID does this by opening parallel input data streams to data nodes that store those required blocks. This time corresponds to *read time*, and is dependent on disk I/O, network bandwidth and latency. Decoding operation happens simultaneously with read operation, and both operations are implemented as concurrent threads. The decoder process, which is responsible for repairing the failed block, needs to wait at times for input data to be read and made available for decoding to happen. This period of time, when there is no computation happening, and the thread waits for read data to come in, is denoted as *wait time.* The actual CPU time consumed for the decode computations is called *decode time.* Reading time reflects the disk I/O performance; decoding time represents the CPU workload and waiting time reveals the implementation and network efficiency. Fig. 4. shows the relationship between the three times as a function of time flow.

Apart from the repair time, the amount of data read for repair (denoted as HDFSBytesRead) by the repair worker node and the total CPU time taken for repair process are also measured from the logs.

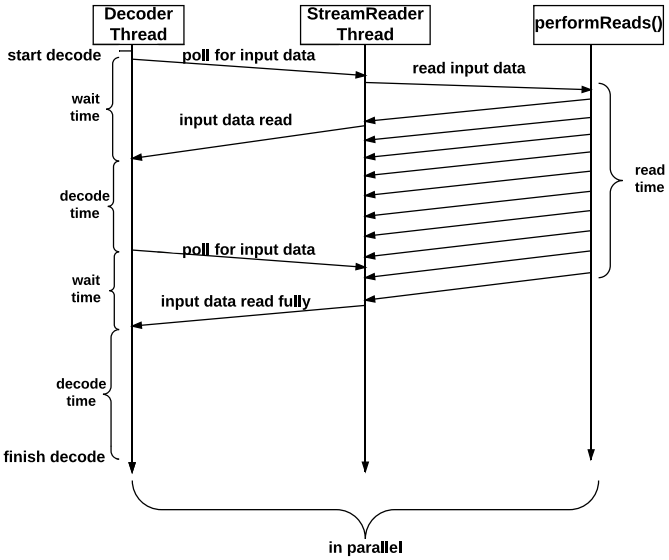


Fig. 4: Read, decode and wait times of the repair process: read and decode happen concurrently; the time CPU is idle waiting for data to be read in-order to decode, is the wait time.

B. Analysis of Results

The results of single node repair performance is discussed first. The plots in Fig. 5 show the repair times consumed during repairing a single node failure under all three experimental test cases that are studied. Fig. 5(a) shows single node recovery times when the setting is run without location awareness and parity replication. Fig.5(b) reports the times when Hadoop is aware of the the location but parity meta replication is not enabled. In Fig. 5(c), plotted are the results when both location awareness and parity replication are set. The values in the box plots are obtained by running 20 iterations of the experiment script.

In all three cases, it is clearly evident that MXOR codes perform faster recovery as compared to FB RS codes and XORBAS LRC codes, confirmed by reduction in all metrics that were plotted. XORBAS LRC is observed to perform confirming the claimed performance in the paper [5] with a decrease in repair times by (10-15)% as compared to FB RS, over all test cases. In the absence of location awareness, repair efficiency of code is affected by the surviving nodes for repair being far from the recovery worker node; increasing the repair times, as seen from Fig. 5(a). The plots in Fig. 5(b) confirm that setting location awareness has resulted in decreasing the decode and wait times of RS and XORBAS codes. This idea is also discussed in previous papers such as [7], [32], [5] and [10]. Fig. 6 shows the average recovery times (sum of decode and wait times) of the three codes under assessment in the above mentioned settings.

Plot in Fig. 5(c) shows that replicating parities, along with rack awareness has contributed to further reduction in recovery times of RS and XORBAS codes. Replication of parity blocks increases their availability; which in turn helps in perform-

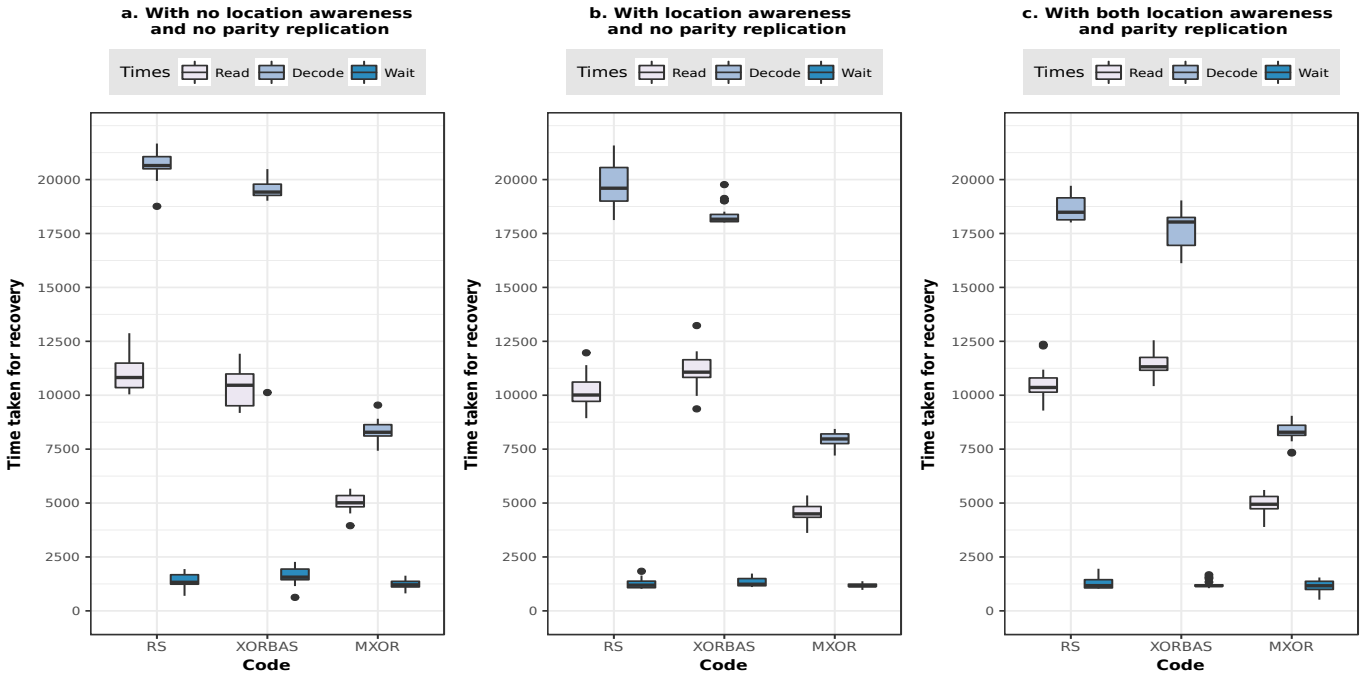


Fig. 5: Repair times during single node failure

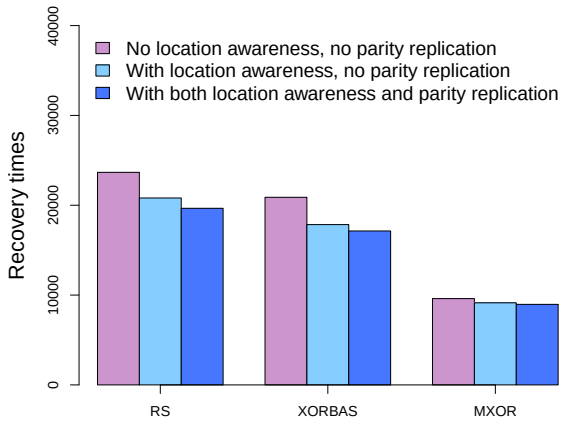


Fig. 6: Average recovery times (single node failure) of codes under different settings

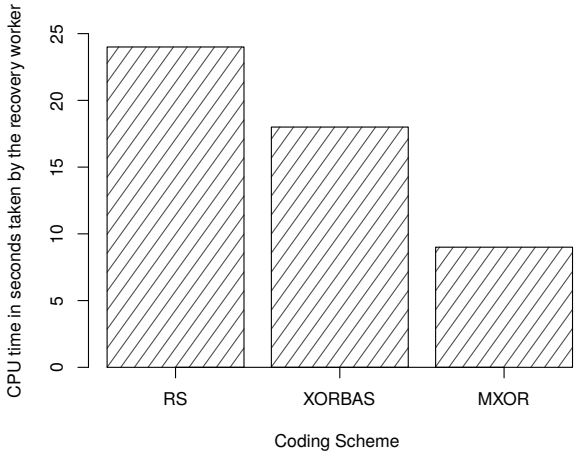


Fig. 7: Storage code scheme vs. CPU time (single node failure)

ing faster repairs. The motivation behind replicating parities was triple replication inherent in Hadoop's architecture, that ensures reliability of hot data. There has been recent works which map the replication idea to erasure coded storage [32], [10].

Figure 7 shows the total CPU time in seconds taken by the worker node to perform the repair of single node failure for the three codes under analysis. Figure 8 shows the codes assessed vs. the mega bytes read during repair process during single node failure. It is seen that both the parameters decrease as we move from RS to XORBAS to MXOR codes. Reducing the bytes read has a direct impact on the network bandwidth consumed during recovery process; lesser the bytes read, better it is. Decreasing CPU times is evident of the fact that the repair job needs lesser computations with XORBAS and MXOR codes as compared to RS codes. Considering double node failure scenario, the same set of experiments were repeated simulating two nodes failure scenario with the same configuration settings and the metrics were measured. Fig. 9 shows the recovery times across various test cases considered. The recovery times are found to be decreasing with the addition of the ideas of topology awareness and parity replication. RS and XORBAS code schemes have gained good benefits from location awareness and parity meta replication.

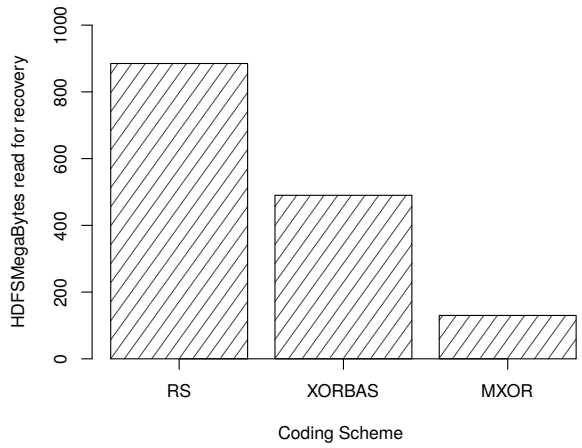


Fig. 8: Storage code scheme vs. MegaBytes Read (single node failure)

C. Observations

From the results of experiments simulating single and double node failures, it is observed that parity replication, augmented with location awareness offers significant improvements in the recovery performance of RS and XORBAS codes in the cluster. It is interesting to note that MXOR code does not gain benefit in recovery performance as compared to RS and XORBAS codes. During single node failure, it requires downloading only one parity block for recovery, hence having extra copies of parities become less relevant resulting in no improvements. Double node failure demands downloading only a subset of surviving nodes depending on the blocks that are lost. This is in contrast to RS codes, where each of the two failures involve downloading k of the remaining surviving blocks.

D. Towards erasure-code aware block placement policies

In this section, we start with the proposition that in a geographically distributed cluster, if the block placement is done intelligently, improvements in recovery performance is achieved. Any erasure code has a structure, and geographical diversity demands spreading blocks of the coded stripe across nodes at disparate locations. We propose a heuristic that, when this spreading is done taking into consideration the structure of the code, it will result in better recovery performance. The results of two different experiments and their observations are given below to support the proposed heuristic.

Results and observations on fully distributed vs. fully centralized clusters: Towards this, the results of recovery performance of Facebook's RS codes and XORBAS code are presented. The tests are run on two clusters set up on the NeCTAR research cloud that are drastically opposite in the geo-distribution of their data nodes. Both clusters have 25 data nodes each, with a single master node located in Tasmania availability zone. Cluster-1 has all nodes confined to a single location: Tasmania, whereas Cluster-5 has nodes distributed across five disparate locations across Australia, namely Tasmania, Queensland, South Australia, New South Wales and Melbourne. To be more precise, Cluster-1 has no geo-diversity at all; while Cluster-5 has the most geo-diversity in terms of the locations of their nodes. In both cases, single node failure is simulated (by choosing a node randomly to fail) and recovery times are measured.

In Fig. 10, the recovery performance of Facebook's RS codes is compared with XORBAS local parity codes, with data results obtained by running 40 iterations of test script on Cluster-5. The local parities in XORBAS only require XOR operations to recover from node failures. With reference to the plots in Fig. 10(a) and (b), local parities have helped in improving read performance in Cluster-5. This directly translates to reduction in wait times, because required blocks are already read and available for decoding computation. The peak points of reading times represent the recovery processes that involved the interaction among helper nodes located farthest from each other. Fig. 11 presents the test case results when run on Cluster-1. The reading and decoding processes are observed to perform faster here as compared to the case in Cluster-5, the reason is attributed to the fact that all data nodes available at the same location. To conclude, results show that repair is faster when blocks of a stripe are placed into nodes at a single location as opposed to placement on nodes spread across a distributed cluster.

Results and observations on MXOR codes before and after code aware block placement: The results and discussions presented below motivate the heuristic further. From the results and analysis presented in Section-VI (B), MXOR codes give the best repair performance. Hence, the following discussion is focussed only on MXOR codes.

Let us consider a storage cluster that has nodes spread across L geographic locations. Let N be the total number of nodes and l_i represent the number of nodes at any location i. Let n denote the number of blocks of a coded stripe. Given this, the total number of possibilities with which cluster nodes can be chosen to place coded blocks of any code is:

$$\binom{\sum_L l_i}{n}$$

In MXOR code, during single node failure, the code design demands only two (one source and one parity) among the sixteen live blocks (please refer Fig. 1) to be downloaded for recovery process to be initiated. The fastest recovery happens when these three blocks are placed in the same location. The code blocks can be separated into five locality groups, as shown in Fig. 12. The two horizontal parities are not required for repairing a single node failure, and hence they can be grouped together with any of the other five groups.

After RAIDing, the blocks are placed according to a placement policy that co-locates blocks belonging to the same group, so that it will lead to optimal recovery performance i.e, the placement of blocks belonging to any single group is restricted to the same location. This restriction makes the number of possibilities of placing coded blocks $5 * \binom{l_i}{n}$. The heuristic was implemented and tested on HDFS-RAID module.

RAIDing is a two-step process in Hadoop-20's HDFS-RAID

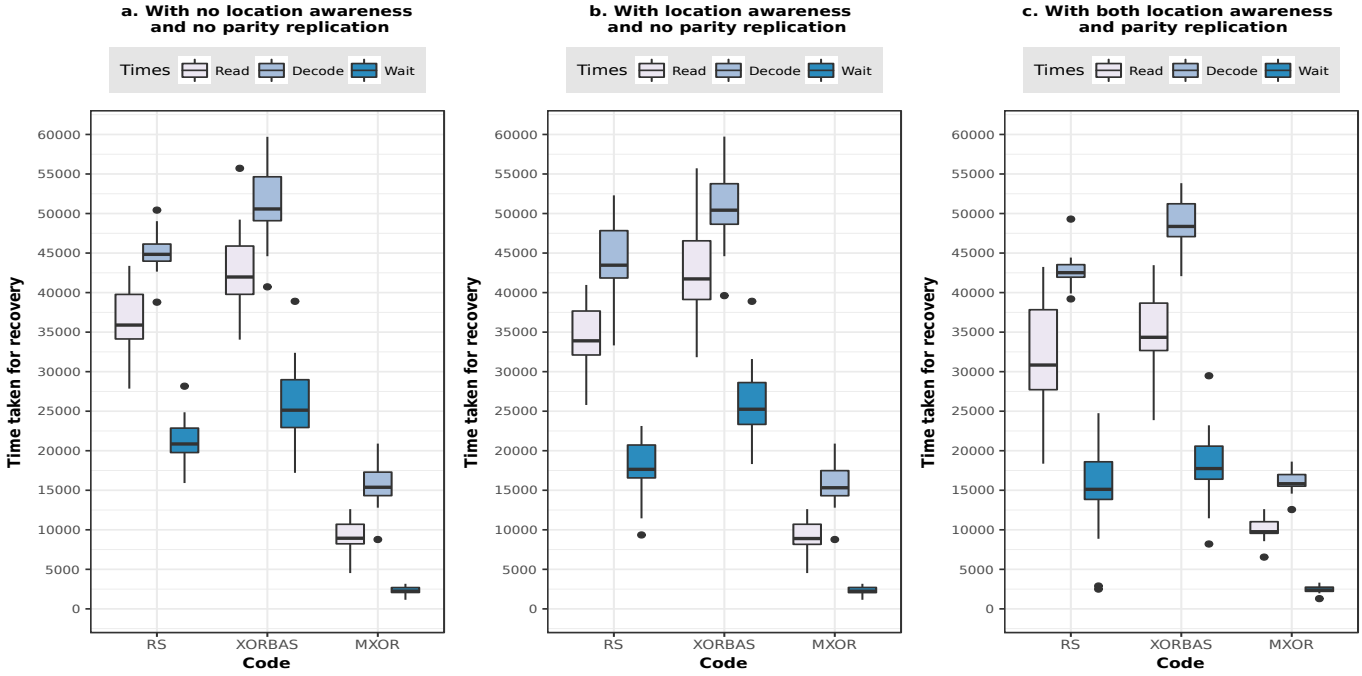
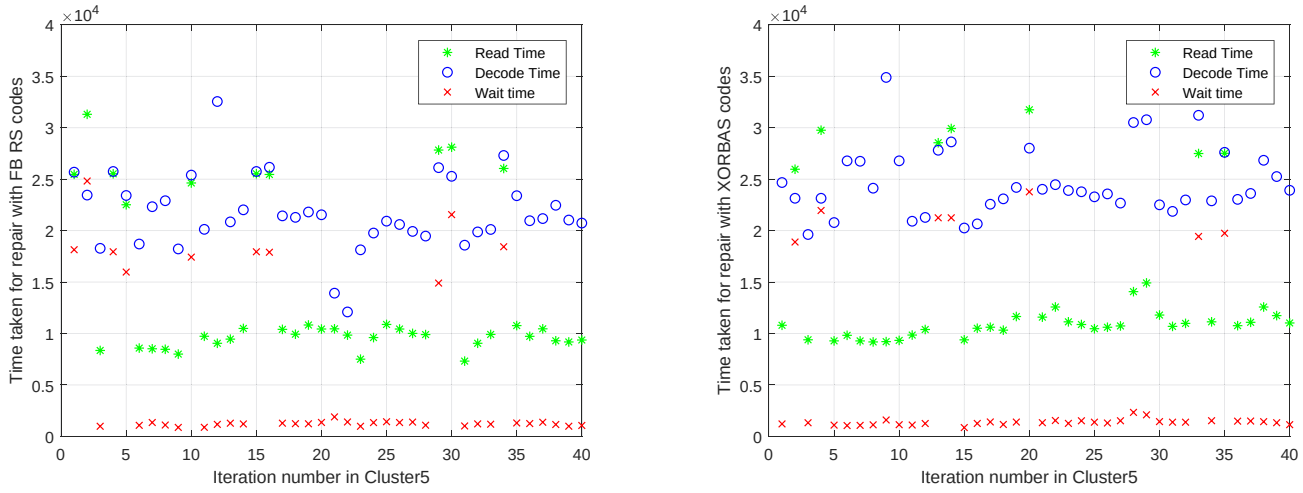


Fig. 9: Repair times during double node failure

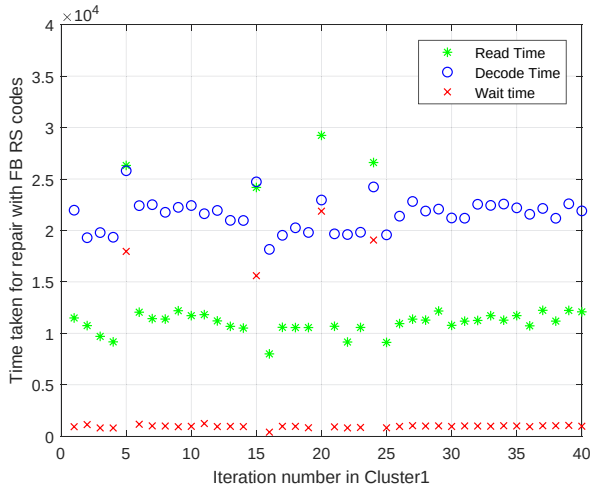


(a) Facebook Hadoop in Cluster-5

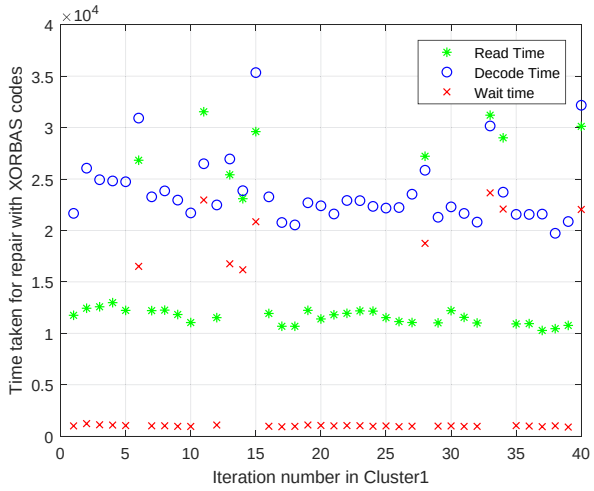
(b) XORBAS Hadoop in Cluster-5

Fig. 10: Comparison of recovery times of Facebook Hadoop and XORBAS Hadoop on Cluster5

module. Any file to be uploaded is triple-replicated first and stored to file system (please refer to the steps mentioned in Section-V) by default. Thereafter, storage is chosen to be RAIDed by the *RaidNode* daemon once a specific time period (which can be set) elapses. The storage is arranged into stripes and encoding operation computes parity blocks corresponding to the source blocks in all stripes. After RAIDing completes, extra replicas of all source blocks are deleted from the storage, and replication is set to 1. The *RaidNode* has a thread called *PlacementMonitor* that periodically scans all the RAIDed directories at regular intervals. It checks if blocks of a stripe are co-located in a node; if so, it prepares block move requests and forwards them to the *BlockMover*. The *BlockMover* executes the move operation by moving the block to a different node in the cluster. We modified *PlacementMonitor* to check whether blocks are placed according to the locality group placement in Fig. 12. If not, the *BlockMover* moves them to their respective groups. The locations of blocks after introducing the code aware placement strategy is shown in Fig. 13. The test cluster used here is the same that was set-up for testing recovery



(a) Facebook Hadoop in Cluster-1



(b) XORBAS Hadoop in Cluster-1

Fig. 11: Comparison of recovery times of Facebook Hadoop and XORBAS Hadoop on Cluster1

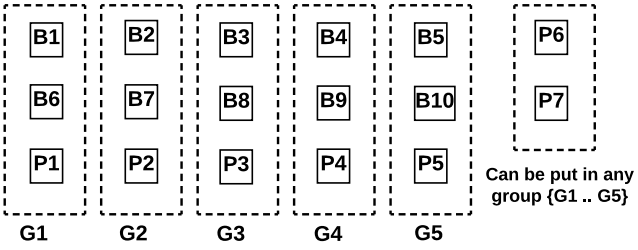


Fig. 12: Locality groups in MXOR codes

performance during single and double node failure scenarios.

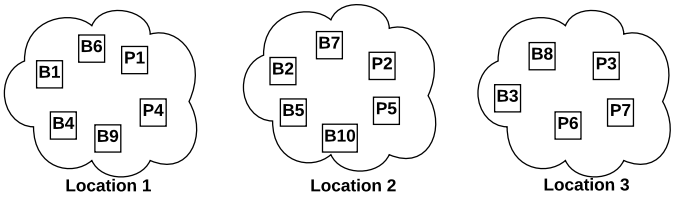


Fig. 13: Erasure code aware blocks placement in MXOR codes

The experiment for simulating single node failure (with location awareness) was conducted. Fig. 14 shows the recovery times before and after code aware block placement. Results reported are averaged over 10 iterations and present improvement in recovery performance with code aware placement of blocks.

The new heuristic can be applied to any erasure code in general. Let us consider the case of RS and XORBAS codes. XORBAS, by design, has locality, and coded blocks are separated into three groups. In a 3 location cluster, it is the mere task of restricting blocks of a group to a single location.

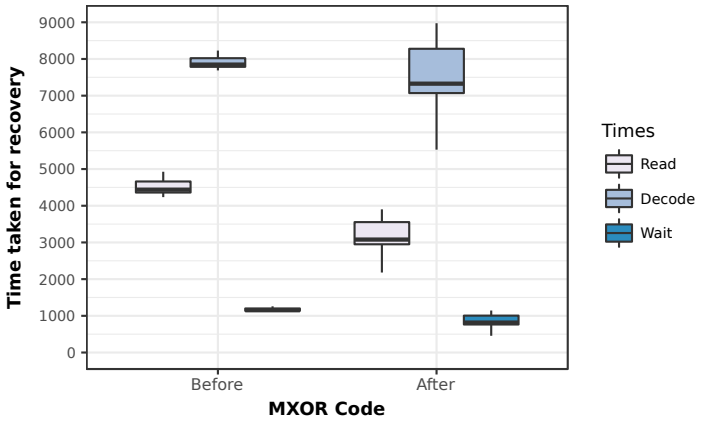


Fig. 14: Recovery times of MXOR codes before and after code aware placement strategy

In RS codes, all blocks of a stripe are needed to repair a missing block; hence with 3 location cluster we do a uniform distribution. This groups the blocks into three groups of 4 blocks each plus extra 2 blocks. These three groups can be placed to the three locations, and the left out 2 blocks can be placed randomly in any of the three locations.

From the above observations, it can be concluded that, a better design that takes into account how the coded blocks are placed to suit a geo-diverse cluster can help increase recovery performance of any coded storage system. The results show that placing blocks respecting the structure of the underlying erasure code enhances performance during recovery from node failures. It affirms the heuristic that there is possibility of optimization based on erasure code design, and also points out further possibilities of optimizing based on other properties of network characteristics that support coded storage. Modelling storage network topologies with different cost parameters

like bandwidth and delay has been an open research area and has not been addressed previously in literature. There has been active research by different groups on codes that reduce repair bandwidth. Influence of network topology on codes has been investigated in the context of trees [33]. Most of the storage networks today are much more diverse, with different communication capacities and topologies. Minimum communication needed to repair an erasure in such contexts has not been studied in detail so far. This research area is identified earlier by Dimakis et.al. in their celebrated paper on network coding [34]. We leave this open problem for our future work.

VIII. Conclusion

When compared to an erasure coded storage system which has all data nodes located at the same geographical location, an erasure coded large scale distributed storage system spread across wide geography entails more complex recovery from node failure. To tackle this issue, we presented an assessment of three popular codes along with two simple ideas of managing location awareness information and maintaining additional copies of parities; following which we presented a heuristic that placing blocks intelligently based on the code design would result in better repair performance. The results of our study have revealed new facets of erasure codes when implemented on Hadoop storage system in a geo-distributed environment. Erasure codes, in particular are not a silver-bullet solution for providing reliability.

The experimental results confirm that topology awareness and metareplication improve recovery performance to some extent. To get further improvements, a new block placement policy that optimizes recovery performance and follows the erasure code design shall be considered. The sum of all these ideas could offer a better solution for a code based storage system spanning a large geographical area.

Acknowledgement

This research was supported by use of the Nectar Research Cloud, a collaborative Australian research platform supported by the National Collaborative Research Infrastructure Strategy (NCRIS).

References

[1] James C. Corbett and Jeffrey et. al. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 251–264, Berkeley, CA, USA, 2012. USENIX Association.

[2] Constantinos Evangelinos and Chris N. Hill. Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. Cloud Computing and Its Applications, October 2008.

[3] Jie Li, Marty Humphrey, Deborah A. Agarwal, Keith R. Jackson, Catharine van Ingen, and Youngryel Ryu. escience in the cloud: A modis satellite data reprojection and reduction pipeline in the windows azure platform. In *IPDPS*, pages 1–10. IEEE.

[4] K.V. Rashmi, Nihar B. Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A "hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers. *SIGCOMM Comput. Commun. Rev.*, 44(4):331–342, August 2014.

[5] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. Xoring elephants: Novel erasure codes for big data. *Proceedings of the VLDB Endowment*, 2013.

[6] Mingyuan Xia, Mohit Saxena, Mario Blaum, and David A. Pease. A tale of two erasure codes in hdfs. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST'15, pages 213–226, Berkeley, CA, USA, 2015. USENIX Association.

[7] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. Erasure coding in windows azure storage. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[8] HDFS-RAID. http://wiki.apache.org/hadoop/HDFS-RAID. [Online; accessed 10-5-2017].

[9] Runhui Li, Jian Lin, and Patrick P. C. Lee. CORE: augmenting regenerating-coding-based recovery for single and concurrent failures in distributed storage systems. *CoRR*, abs/1302.3344, 2013.

[10] Kyumars Sheykh Esmaili, Lluis Pamies-Juarez, and Anwitaman Datta. The CORE storage primitive: Cross-object redundancy for efficient data repair & access in erasure coded storage. *CoRR*, abs/1302.5192, 2013.

[11] Lakshmi J. Mohan, Renji Luke Harold, Pablo Ignacio Serrano Caneleo, Udaya Parampalli, and Aaron Harwood. Benchmarking the performance of hadoop triple replication and erasure coding on a nation-wide distributed cloud. In *2015 International Symposium on Network Coding, NetCod 2015, Sydney, Australia, June 22-24, 2015*, pages 61–65, 2015.

[12] NeCTAR. https://www.nectar.org.au. [Online; accessed 10-5-2017].

[13] Facebook Hadoop-20. https://github.com/facebookarchive/hadoop-20. [Online; accessed 10-5-2017].

[14] Hadoop-USC. https://github.com/madiator/HadoopUSC. [Online; accessed 10-5-2017].

[15] M. Blaum, J. Brady, J. Bruck, and Jai Menon. Evenodd: an efficient scheme for tolerating double disk failures in raid architectures. *IEEE Transactions on Computers*, 44(2):192–202, Feb 1995.

[16] V. R. Cadambe, S. A. Jafar, H. Maleki, K. Ramchandran, and C. Suh. Asymptotic interference alignment for optimal repair of mds codes in distributed storage. *IEEE Transactions on Information Theory*, 59(5):2974–2987, May 2013.

[17] Z. Wang, A. G. Dimakis, and J. Bruck. Rebuilding for array codes in distributed storage systems. In *2010 IEEE Globecom Workshops*, pages 1905–1909, Dec 2010.

[18] Liping Xiang, Yinlong Xu, John C.S. Lui, and Qian Chang. Optimal recovery of single disk failure in rdp code storage systems. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '10, pages 119–130, New York, NY, USA, 2010. ACM.

[19] Cheng Huang, Minghua Chen, and Jin Li. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *Trans. Storage*, 9(1):3:1–3:28, March 2013.

[20] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin. On the locality of codeword symbols. *IEEE Transactions on Information Theory*, 58(11):6925–6934, Nov 2012.

[21] G. M. Kamath, N. Prakash, V. Lalitha, and P. V. Kumar. Codes with local regeneration and erasure correction. *IEEE Transactions on Information Theory*, 60(8):4637–4660, Aug 2014.

[22] L. Pamies-Juarez, H. D. L. Hollmann, and F. Oggier. Locally repairable codes with multiple repair alternatives. In *2013 IEEE International Symposium on Information Theory*, pages 892–896, July 2013.

[23] N. Silberstein, A. S. Rawat, and S. Vishwanath. Error-correcting regenerating and locally repairable codes via rank-metric codes. *IEEE Transactions on Information Theory*, 61(11):5765–5778, Nov 2015.

[24] Itzhak Tamo and Alexander Barg. A family of optimal locally recoverable codes. *IEEE Transactions on Information Theory*, 60(8):4661–4676, 2014.

[25] Q. Gong, J. Wang, D. Wei, J. Wang, and X. Wang. Optimal node selection for data regeneration in heterogeneous distributed storage systems. In *2015 44th International Conference on Parallel Processing*, pages 390–399, Sept 2015.

[26] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, and Sanjeev Kumar. f4: Facebook's warm blob storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 383–398, Broomfield, CO, October 2014. USENIX Association.

[27] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.
[28] Jerasure. http://lab.jerasure.org/jerasure/jerasure. [Online; accessed 10-5-2017].
[29] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.
[30] Data Replication- HDFS Architecture guide. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Data+Replication. [Online; accessed 10-5-2017].
[31] K. V. Rashmi, Nihar B. Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the facebook warehouse cluster. In *Proceedings of the 5th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'13, pages 8–8, Berkeley, CA, USA, 2013. USENIX Association.
[32] M. Nikhil Krishnan, N. Prakash, V. Lalitha, Birenjith Sasidharan, P. Vijay Kumar, Srinivasan Narayanamurthy, Ranjit Kumar, and Siddhartha Nandi. Evaluation of codes with inherent double replication for hadoop. In *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*, Philadelphia, PA, June 2014. USENIX Association.
[33] J. Li, S. Yang, X. Wang, and B. Li. Tree-structured data regeneration in distributed storage systems with regenerating codes. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9, March 2010.
[34] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A survey on network codes for distributed storage. *Proceedings of the IEEE*, 99(3):476–489, March 2011.
[35] Rack awareness. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/RackAwareness.html. [Online; accessed 10-5-2017].

APPENDIX

Location Awareness

This section briefly introduces the notion of topology awareness with a custom network layout. The Hadoop software framework provides a ready-to-use and flexible implementation of custom network topologies based on bash scripts. It basically uses the rack awareness idea in a data center to implement topology awareness in a cluster. In order to configure any custom geographic topology, some lines should be added to the Hadoop configuration file $core-site.xml$. The following lines explain the complete process:

```
<property>
   <name>topology.script.file.name</name>
   <value>/usr/local/hadoop/conf/rack_topology.sh
   </value>
   <description>Identifies the network layout.
   </description>
</property>

<property>
    <name>topology.script.number.args</name>
    <value>1</value>
     <description>List of IPs to check
   </description>
</property>
```

The bash script used in our experiments is based on the standard Hadoop topology awareness code, provided at Hadoop Wiki [35]. We modified the code, based on other community references, generating the following code for our geo-diverse cluster:

```
#!/bin/bash

# Adjust/Add the property
# "net.topology.script.file.name"
# to core-site.xml with the "absolute" path the this
# file.  ENSURE the file is "executable".

# the input is one or more IP values (like 127.0.0.1),
# hostnames can be misinterpreted

# Supply appropriate rack prefix
RACK_PREFIX=default

# To test, supply a hostname as script input:
if [ $# -gt 0 ]; then

CTL_FILE=${CTL_FILE:-"rack_topology.data"}

# changed folder to $HADOOP_HOME (system variable)
HADOOP_CONF=${HADOOP_CONF:-"$HADOOP_HOME/conf"}

if [ ! -f ${HADOOP_CONF}/${CTL_FILE} ]; then
  echo -n "/$RACK_PREFIX/rack"
  exit 0
fi

while [ $# -gt 0 ] ; do
  nodeArg=$1
  exec< ${HADOOP_CONF}/${CTL_FILE}
  result=""
  while read line ; do
    ar=( $line )
    if [ "${ar[0]}" = "$nodeArg" ] ; then
      result="${ar[1]}"
      # echo "$result"
      # this returns the 2nd column in
      # the rack_topology.data file
    fi
  done
  shift
  if [ -z "$result" ] ; then
    echo -n "/$RACK_PREFIX/rack"
  else
    echo -n "/$RACK_PREFIX/rack_\$result"
  fi
done

else
  echo -n "/$RACK_PREFIX/rack"
fi
```

The above script reads a topology information file "rack_toplology.data" that specifies racks (in our case, locations) and machines in a key-pair relationship using a simple format. Given that our clusters were distributed around Australia, it was natural to organize the different racks using the different locations available on the NeCTAR cloud.