XORing Elephants: Novel Erasure Codes for Big Data

Maheswaran Sathiamoorthy
University of Southern California
Los Angeles, CA, USA
msathiam@usc.edu

Megasthenis Asteris
University of Southern California
Los Angeles, CA, USA
asteris@usc.edu

Dimitris Papailiopoulous
University of Southern California
Los Angeles, CA, USA
papailio@usc.edu

Alexandros G. Dimakis
University of Southern California
Los Angeles, CA, USA
dimakis@usc.edu

Ramkumar Vadali
Facebook
ramkumar.vadali@fb.com

Scott Chen
Facebook
sc@fb.com

Dhruba Borthakur
Facebook
dhruba@fb.com

ABSTRACT

Distributed storage systems for large clusters typically use replication to provide reliability. Recently, erasure codes have been used to reduce the large storage overhead of three-replicated systems. Reed-Solomon codes are the standard design choice and their high repair cost is often considered an unavoidable price to pay for high storage efficiency and high reliability.

This paper shows how to overcome this limitation. We present a novel family of erasure codes that are efficiently repairable and offer higher reliability compared to Reed-Solomon codes. We show analytically that our codes are optimal on a recently identified tradeoff between locality and minimum distance.

We implement our new codes in Hadoop HDFS and compare to a currently deployed HDFS module that uses Reed-Solomon codes. Our modified HDFS implementation shows a reduction of approximately $2\times$ on the repair disk I/O and repair network traffic. The disadvantage of the new coding scheme is that it requires 14% more storage compared to Reed-Solomon codes, an overhead shown to be information theoretically optimal to obtain locality. Because the new codes repair failures faster, this provides higher reliability, which is orders of magnitude higher compared to replication.

1. INTRODUCTION

MapReduce architectures are becoming increasingly popular for big data management due to their high scalability properties. At facebook, large analytics clusters store petabytes of information and handle multiple analytics jobs using Hadoop MapReduce. Standard implementations rely on a distributed file system that provides reliability by replication. Typically, triple replication is used for most files, often ensuring one replica of each block is placed in the same rack to ensure fast repairs when nodes fail. The major disadvantage of replication is the very large storage overhead of 200%, which reflects on the cluster costs. This overhead is becoming a major bottleneck as the amount of managed data grows faster than data center infrastructure.

For this reason, facebook and many others are transitioning to erasure coding techniques (typically, classical Reed-Solomon codes) to introduce redundancy while saving storage [4, 22], especially for the data that is more archival in nature. In this paper we show that classical codes are highly suboptimal for distributed MapReduce architectures. We introduce new erasure codes that address the main challenges of distributed data reliability and information theoretic bounds that show the optimality of our construction. We rely on measurements from a large facebook production cluster (more than 3000 nodes, 30 PB of logical data storage) that uses Hadoop MapReduce for data analytics. We recently started deploying an open source HDFS Module called HDFS RAID ([3, 9]) that relies on Reed-Solomon (RS) codes. In HDFS RAID, the replication factor of "cold" (*i.e.*, rarely accessed) files is lowered to 1 and a new parity file is created, consisting of parity blocks.

Using the parameters of facebook clusters, the data blocks of each large file are grouped in stripes of 10 and for each such set, 4 parity blocks are created. This system (called a RS $(10, 4)$) can tolerate any 4 block failures and has a storage overhead of only 40%. RS codes are therefore significantly more robust and storage efficient compared to replication. In fact, this storage overhead is the minimal possible, for this level of reliability [8]. Codes that achieve this optimal storage-reliability tradeoff are called Maximum Distance Separable (MDS) [31] and Reed-Solomon codes [28] form the most widely used MDS family.

Classical erasure codes are suboptimal for distributed environments because of the so-called *Repair problem:* When a single node fails, typically one block is lost from each stripe. RS codes are usually repaired with the simple method that requires transferring 10 blocks and recreating the original 10 data blocks even if a single block is lost [29], hence creating a $10\times$ overhead in repair bandwidth and disk I/O.

Recently, information theoretic results established that it is possible to repair erasure codes with much less network bandwidth compared to this naive method [7]. There has been significant amount of very recent work on designing such efficiently repairable codes, see section 6 for an overview of this literature.

Our Contributions: We introduce a new family of erasure codes called *Locally Repairable Codes (LRCs)*, that are efficiently repairable both in terms of network bandwidth and disk I/O. We analytically show that our codes are information theoretically optimal in terms of their locality, *i.e.*, the number of other blocks needed to repair *single block failures.* We present both randomized and explicit LRC constructions starting from generalized Reed-Solomon parities.

We also design and implement *HDFS-Xorbas*, a module that replaces Reed-Solomon codes with LRCs in HDFS-RAID. We evaluate HDFS-Xorbas on experiments on Amazon EC2 and a cluster in social network X. Our experiments show that Xorbas enables approximately a $2\times$ reduction in disk I/O and repair network traffic compared to the Reed-Solomon code currently used in production. The disadvantage of the new code is that it requires 14% more storage compared to RS, an overhead shown to be information theoretically optimal for the obtained locality.

One interesting side benefit is that because Xorbas repairs failures faster, this provides higher availability, due to more efficient degraded reading performance. Under a simple Markov model evaluation, Xorbas has 2 more zeros in Mean Time to Data Loss (MTTDL) compared to RS $(10, 4)$ and 6 more zeros compared to 3-replication.

1.1 Importance of repair traffic

At X, large analytics clusters store petabytes of information and handle multiple MapReduce analytics jobs.

In a 3000 node production cluster storing approximately 230 million blocks (each of size 256MB), only 8% of the data is currently RS encoded ('RAIDed'). *Our goal is to design more efficient coding schemes that would allow this fraction to increase, hence saving petabytes of storage.*

Figure 1 shows a recent trace of node failures in this production cluster. It is quite typical to have 20 or more node failures per day that trigger repair jobs, even when most repairs are delayed to avoid transient failures. A typical data node will be storing approximately 15 TB and the repair traffic with the current configuration is estimated around $10 - 20\%$ of the total average of 2 PB/day cluster network traffic. As discussed, RS encoded blocks require approximately $10\times$ more network repair overhead per bit compared to replicated blocks. We estimate that if 50% of the cluster was RS encoded, the repair network traffic would completely saturate the cluster network links.

There are four additional reasons why efficiently repairable codes are becoming increasingly important in coded storage systems. The first is the issue of *degraded reads.* Transient errors with no permanent data loss correspond to 90% of data center failure events [10, 22]. During the period of a transient failure event, block reads of a coded stripe will be *degraded* if the corresponding data blocks are unavailable. In this case, the missing data block can be reconstructed by a repair process, which is not aimed at fault tolerance but at higher data availability. The only difference with standard repair is that the reconstructed block does not have to be written in disk. For this reason, efficient and fast repair can significantly improve data availability.

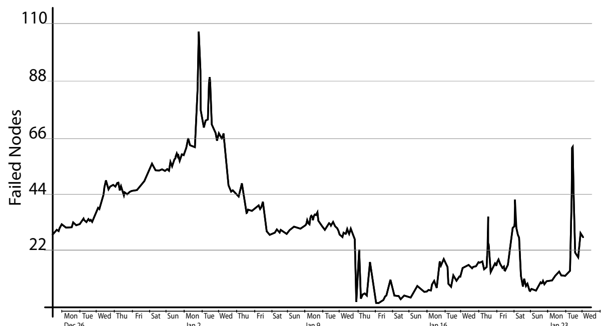


Figure 1: Number of failed nodes over a single month period in a 3000 node production cluster of X.

The second is the problem of efficient *node decommissioning.* Hadoop offers the decommission feature to retire a faulty data node. Functional data has to be copied out of the node before decommission, a process that is complicated and time consuming. Fast repairs allow to treat node decommissioning as a scheduled repair and start a MapReduce job to recreate the blocks without creating very large network traffic.

The third reason is that repair influences the *performance* of other concurrent MapReduce jobs. Several researchers have observed that the main bottleneck in MapReduce is the network [5]. As mentioned, repair network traffic is currently consuming a non-negligible fraction of the cluster network bandwidth. This issue is becoming more significant as the disk capacity is increasing disproportionately fast compared to network bandwidth in data centers. This increasing storage density trend emphasizes the importance of local repairs when coding is used.

Finally, local repair would be a key in facilitating *geographically distributed* file systems across data centers. Geodiversity has been identified as one of the key future directions for improving latency and reliability [15]. Traditionally, sites used to distribute data across data centers via replication. This, however, dramatically increases the total storage cost. Reed-Solomon codes across geographic locations at this scale would be completely impractical due to the high bandwidth requirements across wide area networks. Our work makes local repairs possible at a marginally higher storage overhead cost.

Replication is obviously the winner in optimizing the four issues discussed, but requires a very large storage overhead. On the opposing tradeoff point, MDS codes have minimal storage overhead for a given reliability requirement, but suffer in repair and hence in all these implied issues. One way to view the contribution of this paper is a new intermediate point on this tradeoff, that sacrifices some storage efficiency to gain in these other metrics.

The remainder of this paper is organized as follows: We initially present our theoretical results, the construction of Locally Repairable Codes and the information theoretic optimality results. We defer the more technical proofs to the Appendix. Section 3 presents the HDFS-Xorbas architecture and Section 4 discusses a Markov-based reliability analysis. Section 5 discusses our experimental evaluation on Amazon EC2 and X's cluster. We finally survey related work in Section 6 and conclude in Section 7.

2. THEORETICAL CONTRIBUTIONS

Maximum distance separable (MDS) codes are often used in various applications in communications and storage systems [31]. A $(k, n-k)$-MDS code[1] of rate $R = \frac{k}{n}$ takes a file of size M, splits it in k equally sized blocks, and then encodes it in n coded blocks each of size $\frac{M}{k}$. Here we assume that our file has size exactly equal to k data blocks to simplify the presentation; larger files are separated into stripes of k data blocks and each stripe is coded separately.

A $(k, n-k)$-MDS code has the property that any k out of the n coded blocks can be used to reconstruct the entire file. It is easy to prove that this is the best fault tolerance possible for this level of redundancy: any set of k blocks has an aggregate size of M and therefore no smaller set of blocks could possibly recover the file.

Fault tolerance is captured by the metric of *minimum distance.*

Definition 1 (Code Minimum Distance:). *The minimum distance d of a code of length n, is equal to the minimum number of erasures of coded blocks after which the file cannot be retrieved.*

MDS codes, as their name suggests, have the largest possible distance which is $d_{\text{MDS}} = n - k + 1$. For example the minimum distance of (10,4) RS is $n-k+1 = 5$ which means that five or more block erasures are needed to create data loss.

The second metric we will be interested in is *Block Locality*:

Definition 2 (Block Locality). *An $(k, n-k)$ code has block locality r, when each coded block is a function of at most r other coded blocks of the code.*

Codes with block locality r have the property that, upon *any* single block erasure, fast repair of the lost coded block can be performed by computing a function on r existing blocks of the code. This concept was recently and independently introduced in [12, 25, 26].

When we require small locality, each single coded block should be repairable by using only a *small* subset of existing coded blocks $r << k$, even when n, k grow. The following fact shows that locality and good distance are *in conflict:*

Lemma 1. *MDS codes with parameters $(k, n-k)$ cannot have locality smaller than k.*

Lemma 1 implies that MDS codes have the *worst possible* locality since any k blocks suffice to reconstruct the *entire* file, not just a single block. This is exactly the cost of optimal fault tolerance.

The natural question is what is the best locality possible if we settled for "almost MDS" code distance. We answer this question and construct the first family of near-MDS codes with non-trivial locality. We provide a randomized an *explicit* family of codes that have *logarithmic* locality on all coded blocks and distance that is asymptotically equal to that of an MDS code. We call such codes $(k, n-k, r)$ Locally Repairable Codes (LRCs) and present their construction in the following section.

Theorem 1. *There exist $(k, n-k, r)$ Locally Repairable codes with logarithmic block locality $r = \log(k)$ and distance $d_{LRC} = n - (1+\delta_k)\,k + 1$. Hence, any subset of $k(1+\delta_k)$ coded blocks can be used to reconstruct the file, where $\delta_k = \frac{1}{\log(k)} - \frac{1}{k}$.*

Observe that if we fix the code rate $R = \frac{k}{n}$ of an LRC and let k grow, then its distance d_{LRC} is almost that of a $(k, n-k)$-MDS code; hence the following corollary.

Corollary 1. *For fixed code rate $R = \frac{k}{n}$, the distance of LRCs is asymptotically equal to that of $(k, n-k)$-MDS codes $\lim_{k\to\infty} \frac{d_{LRC}}{d_{MDS}} = 1$.*

LRCs are constructed on top of MDS codes (and the most common choice will be a Reed-Solomon code).

The MDS encoded blocks are grouped in logarithmic sized sets and then are combined together to obtain parity blocks of logarithmic degree. We prove that LRCs have the optimal distance for that specific locality, due to an information theoretic tradeoff that we establish. Our locality-distance tradeoff is universal in the sense that it covers linear or nonlinear codes and is a generalization of recent result of Gopalan *et al.* [12] which established a similar bound for linear codes. Our proof technique is based on building an information flow graph gadget, similar to the work of Dimakis *et al.*[7, 8]. Our analysis can be found in the Appendix.

2.1 LRC implemented in Xorbas

We now describe the explicit $(10, 6, 5)$ LRC code we implemented in HDFS-Xorbas. For each stripe, we start with 10 data blocks $X_1, X_2, \ldots, X_{10}$ and use a $(10, 4)$ Reed-Solomon over a binary extension field $\mathbb{F}_{2^m}$ to construct 4 parity blocks $P_1, P_2, \ldots, P_4$. This is the code currently used in X production clusters that can tolerate any 4 block failures due to the RS parities. The basic idea of LRCs is very simple: we make repair efficient by adding additional *local parities.* This is shown in figure 2.

By adding the local parity $S_1 = c_1X_1 + c_2X_2 + c_3X_3 + c_4X_5$, a single block failure can be repaired by accessing only 5 other blocks. For example, if block X_3 is lost (or degraded read while unavailable) it can be reconstructed by

$$X_3 = c_3^{-1}(S_1 - c_1X_1 - c_2X_2 - c_4X_4 - c_5X_5). \qquad (1)$$

The multiplicative inverse of the field element c_3 exists as long as $c_3 \neq 0$ which is the requirement we will enforce for all the local parity coefficients. It turns out that the coefficients c_i can be selected to guarantee that all the linear equations will be linearly independent. In the Appendix we present a randomized and a deterministic algorithm to construct such

[1]In classical coding theory literature, codes are denoted by (n, k) where n is the number of data plus parity blocks, classically called blocklength. A (10,4) RS would be classically denoted by (n=14,k=10) Reed-Solomon. Reed-Solomon codes form the most well-known family of MDS codes.

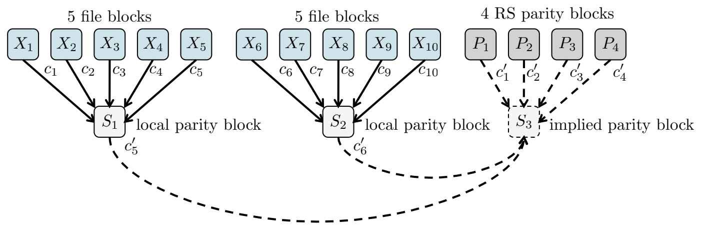

5 file blocks
5 file blocks
4 RS parity blocks
X_1 X_2 X_3 X_4 X_5
X_6 X_7 X_8 X_9 X_{10}
P_1 P_2 P_3 P_4
c_1 c_2 c_3 c_4 c_5
c_6 c_7 c_8 c_9 c_{10}
c'_1 c'_2 c'_3 c'_4
S_1 local parity block
S_2 local parity block
S_3 implied parity block
c'_5
c'_6

Figure 2: Locally repairable code implemented in HDFS-Xorbas. The four parity blocks P_1, P_2, P_3, P_4 are constructed with a standard RS code and the local parities provide efficient repair in the case of single block failures. The main theoretical challenge is to choose the coefficients c_i to maximize the fault tolerance of the code.

coefficients. We emphasize that the complexity of the deterministic algorithm is exponential in the code parameters (n, k) and therefore useful only for small code constructions.

The disadvantage of adding these local parities is the extra storage requirement. While the original RS code was storing 14 blocks for every 10, the three local parities increase the storage overhead into 17/10. There is one additional optimization that we can perform: We show that the coefficients $c_1, c_2, \ldots c_{10}$ can be chosen so that the local parities satisfy an additional *alignment equation* $S1+S2+S3 = 0$. We can therefore not store the local parity S_3 and instead consider it an *implied parity.* Note that to obtain this in the figure, we set $c'_5 = c'_6 = 1$.

When a single block failure happens in a RS parity, the implied parity can be reconstructed and used to repair that failure. For example, if P_2 is lost, it can be recovered by reading 5 blocks P_1, P_3, P_4, S_1, S_2 and solving the equation

$$P_2 = (c'_2)^{-1}(-S_1 - S_2 - c'_1 P_1 - c'_3 P_3 - c'_4 P_4). \qquad (2)$$

In our theoretical analysis we show how to find non-zero coefficients c_i (that must depend on the parities P_i but are not data dependent) for the alignment condition to hold. We also show that for the Reed-Solomon code implemented in HDFS RAID, choosing $c_i = 1$ and therefore performing simple XOR operations is sufficient. We further prove that this code has the largest possible distance ($d = 5$) for this given locality $r = 5$ and blocklength $n = 16$.

3. SYSTEM DESCRIPTION

HDFS-RAID is an open source module that implements RS encoding and decoding over Apache Hadoop [3]. It provides a Distributed Raid File system (DRFS) that runs above HDFS. Files stored in DRFS are divided into stripes, *i.e.*, groups of several blocks. For each stripe, a number of parity blocks are calculated and stored as a separate, parity file corresponding to the original file. HDFS-RAID is implemented in Java (approximately 12,000 lines of code) and is currently used in production by several organizations, including social network X.

The module consists of several components, among which RaidNode and BlockFixer are the most relevant here:

- The RaidNode is a daemon responsible for the creation and maintenance of parity files for all data files stored in the DRFS. One node in the cluster is generally designated to run the RaidNode. The daemon periodically scans the HDFS file system and decides whether a file is to be RAIDed or not, based on its size and age. In large clusters, RAIDing is done in a distributed manner by assigning MapReduce jobs to nodes across the cluster. After encoding, the RaidNode lowers the replication level of RAIDed files to one.
- The BlockFixer is a separate process that runs at the RaidNode and periodically checks for lost or corrupted blocks among the RAIDed files. When blocks are tagged as lost or corrupted, the BlockFixer rebuilds them using the surviving blocks of the stripe, again, by dispatching repair MapReduce jobs.

Both RaidNode and BlockFixer rely on an underlying component: ErasureCode. ErasureCode implements the erasure encoding/decoding functionality. In X's HDFS-RAID, an RS $(10, 4)$ erasure code is implemented through ErasureCode (4 parity blocks are created for every 10 data blocks).

3.1 HDFS-Xorbas

Our system, **HDFS-Xorbas** (or simply Xorbas), is a modification of HDFS-RAID that incorporates Locally Repairable Codes (LRC). To distinguish it from the HDFS-RAID implementing RS codes, we refer to the latter as **HDFS-RS**. In Xorbas, the ErasureCode component has been extended to implement LRC on top of traditional RS codes. The RaidNode and BlockFixer components were also subject to modifications in order to take advantage of the new coding scheme.

HDFS-Xorbas is designed for deployment in a large-scale Hadoop data warehouse, such as X's clusters. For that reason, our system provides backwards compatibility: Xorbas understands both LRC or RS codes and can incrementally modify RS encoded files into LRCs by adding only local XOR parities. To provide this integration with HDFS-RS, the specific LRCs we use are designed as extension codes of the (10,4) Reed-Solomon codes used in X. First, a file is coded using X's RS code and then a small number of additional local parity blocks are created to provide local repairs.

3.1.1 Encoding

Once the RaidNode detects a file which is suitable for RAIDing (according to parameters set in a configuration file) it launches the encoder for the file. The encoder initially divides the file into stripes of 10 blocks and calculates 4 RS parity blocks. Depending on the size of the file, the last stripe may contain fewer than 10 blocks and incomplete stripes are "zero-padded" before the parities are created.

HDFS-Xorbas computes two extra parities for a total of 16 blocks per stripe (10 data blocks, 4 RS parities and 2 Local XOR parities), as shown in 2. Similarly to the calculation of the RS parities, Xorbas calculates all parity blocks in a distributed manner through MapReduce encoder jobs. All blocks are spread across the cluster according to Hadoop's current block placement policy.

3.1.2 Decoding & Repair

RaidNode starts a decoding process when corrupt files are detected. Xorbas uses two decoders: the light-decoder aimed at single block failures per stripe and the heavy-decoder, employed when the light-decoder fails.

When the BlockFixer detects a missing block, it uses the structure of the LRC to determine which 5 blocks are required to recover this block (or repair a corruption). Then, streams are opened to these blocks and the light-decoder attempts to repair by performing a simple XOR of these 5 blocks. A MapReduce job is used to perform the decoding and the recovered block is positioned according to the block placement policy.

If multiple failures have happened, the 5 required blocks may not be available. In that case the light-decoder fails and the heavy decoder is initiated. The heavy decoder operates in the same way as in Reed-Solomon: streams to all the blocks of the stripe are opened and decoding consists of trying to solve linear equations for the lost blocks. For RS, the system of linear equations has a Vandermonde structure [31] which allows small CPU utilization.

In the currently deployed HDFS-RS implementation, even when a single block is corrupt, the BlockFixer opens streams to all 13 other blocks of the stripe (which could be reduced to 10 with a more efficient implementation). The benefit of Xorbas should therefore be clear: for all the single block failures and also many double block failures (as long as the two missing blocks belong to different local XORs), the network and disk I/O overheads will be significantly smaller.

4. RELIABILITY ANALYSIS

Compared to HDFS-RS, Xorbas has faster repairs but stores 16 blocks for every 10 data blocks and hence should expect slightly higher fault rate per stripe. We provide a data reliability analysis by estimating the MTTDL (Mean Time To Data Loss) under a standard Markov model to see how RS and LRCs compare to replication in this metric. Ford *et al.* [10], report values from Google clusters (cells) and show that RS $(9, 4)$ has about six orders of magnitude better reliability than 3-way replication (for their cluster parameters). There exists significant literature on analyzing the reliability of replication, RAID storage [32] and erasure codes [13]. Most of this work relies on standard Markov models to derive the MTTDL analytically for the various storage schemes. We use a similar approach here to compare the reliability of Replication, HDFS-RS and HDFS-Xorbas. We note that the values derived here are not particularly meaningful when looked in isolation, but are useful for comparison purposes (see also [14]).

In our analysis, the total cluster data is denoted by $\mathcal{C}$ and S denotes the stripe size. We use the values $N = 3000$, $\mathcal{C} = 30$ PB, a mean time to failure of 4 years ($= 1/\lambda$) for nodes, and set the block size to be $B = 256$MB (the default at X's warehouses). The network is the main bottleneck for reliability. Based on X's cluster measurements, we use $\gamma =$ 1Gbps for repairs, a value that is limited since coded stripes create traffic across racks. This is because all the blocks of a coded stripe are placed in different racks to provide fault tolerance for correlated failures.

The MTTDL values we computed using the Markov model for replication, HDFS-RS and Xorbas are shown in Table 1. We find that faster repair overcompensates for the storage overhead of LRCs and allows Xorbas LRC (10,6,5) to have 2 more zeros of reliability compared to (10,4) Reed-Solomon. The reliability of 3-replication is substantially lower than both coded schemes, similar to what has been observed in related studies [10].

Scheme	MTTDL (days)
3-replication	$2.3079E + 10$
RS $(10, 4)$	$3.3118E + 13$
LRC $(10, 6, 5)$	$1.2180E + 15$

Table 1: Markov chain based reliability results for Replication, Reed Solomon encoded HDFS and HDFS-Xorbas. Mean Time to Data Loss (MTTDL) is measured in days and assumes independent failures.

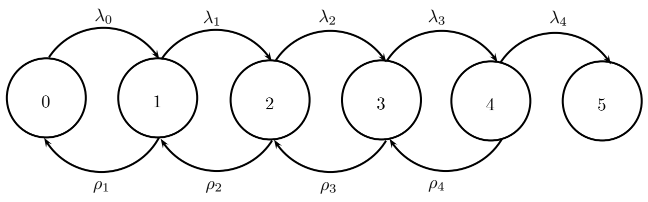

$$\begin{tikzcd}
0 \arrow[r, bend left, "\lambda_0"] & 1 \arrow[l, bend left, "\rho_1"] \arrow[r, bend left, "\lambda_1"] & 2 \arrow[l, bend left, "\rho_2"] \arrow[r, bend left, "\lambda_2"] & 3 \arrow[l, bend left, "\rho_3"] \arrow[r, bend left, "\lambda_3"] & 4 \arrow[l, bend left, "\rho_4"] \arrow[r, bend left, "\lambda_4"] & 5
\end{tikzcd}$$

Figure 3: The Markov model used to calculate the MTTDL

5. EVALUATION

In this section, we provide details on a series of experiments we performed to evaluate the performance of HDFS-Xorbas in two environments: Amazon's Elastic Compute Cloud (EC2) [1] and a test cluster in Social Network X.

5.1 Evaluation Metrics

We rely primarily on the following metrics to evaluate HDFS-Xorbas against HDFS-RS. The metrics are HDFS Bytes Read, Network Traffic and Repair Duration. HDFS Bytes Read corresponds to the total amount of data read by the jobs initiated for repair. The corresponding measurements are collected from the JobTracker reports through a web interface and aggregated across all jobs per failure event. Network traffic represents the total amount of data sent from nodes in the cluster (measured in GB). Since the cluster does not handle any external traffic, this value is equal to the amount of data moving into nodes. Network traffic was measured using Amazon's AWS Cloudwatch monitoring tools. Repair Duration is simply calculated as the time interval between the starting time of the first repair job and the ending time of the last repair job. Note that AWS also provides a metric called DiskReadBytes, representing the number of actual bytes read from the disk. These measurements are similar to HDFS Bytes Read (small deviations can be justified due to memory caching) and hence, we only focus on HDFS Bytes Read which allows us to isolate the amount of data read specifically for repair.

5.2 Amazon EC2

For every experiment, we created two Hadoop clusters on EC2, one running HDFS-RS and the other HDFS-Xorbas. Each cluster consisted of 51 instances of type m1.small, which corresponds to a 32-bit machine with 1.7 GB memory, 1 compute unit and 160 GB of storage, running Ubuntu/Linux-2.6.32. One instance in each cluster served as a master, hosting Hadoop's NameNode, JobTracker and RaidNode daemons, while the remaining 50 instances served as slaves for HDFS and MapReduce, each hosting a DataNode and a TaskTracker daemon, thereby forming a Hadoop cluster of total capacity roughly equal to 7.4 TB. Unfortunately, no information is provided by EC2 on the topology of the cluster.

In total, three experiments were performed on the above setup, increasing the number of files stored for each experiment (50, 100 and 200 files), to understand the impact of the amount of data stored on system performance. The objective of the experiments was to measure key properties such as the number of HDFS bytes read and the real network traffic caused by the repairs. Towards this end, the clusters were loaded with the same amount of logical data and we manually triggered a series of failure events in both the clusters simultaneously to study the dynamics of data recovery.

We used a block size of 64 MB, and all our files were of size 640 MB. Therefore, each file yields a single stripe with 14 and 16 full size blocks in HDFS-RS and HDFS-Xorbas respectively. This choice is representative of the majority of stripes in a production Hadoop cluster: extremely large files are split into many stripes, so in total only a small fraction of the stripes will have a smaller size. In addition, it allows us to better predict the total amount of data that needs to be read in order to reconstruct missing blocks and hence interpret our experimental results. Finally, since block repair depends only on blocks of the same stripe, using larger files that would yield more than one stripe would not affect our results. An experiment involving arbitrary file sizes, is discussed in Section 5.3.

During the course of each single experiment, the two clusters were initially loaded with files. Once all files were RAIDed, we introduced node failures in which a pre-determined number of DataNodes were terminated in both clusters simultaneously. After some time, the Block fixer started MapReduce repair jobs which were used to create the measurements of interest.

In each experiment, we carried out a total of eight failure events, each resulting in termination of one or more DataNodes. The first four failure events consisted of single DataNodes terminations, the next two were terminations of triplets of DataNodes and finally two terminations of pairs of DataNodes. For each failure event, the metrics mentioned above were recorded. After each failure event, sufficient time was provided for both clusters to complete the repair process for all missing blocks, allowing measurements corresponding to distinct events to be isolated. For example in Fig. 4 and Fig. 5a we present measurements from the 200 file experiment. The other experiments involving 50 and 100 files produce similar results and are not shown. The measurements of all the experiments are combined in Figure 6.

For every failure event, while selecting DataNodes to be terminated for each cluster, we made sure that the total number of blocks lost were roughly the same for both clusters. We followed this choice because our objective was to compare the two systems for each block lost. However, since Xorbas has an additional storage overhead, a random failure event would in expectation, lead to loss of 14.3% more blocks in Xorbas compared to RS. In any case, results can be adjusted to take this into account, without significantly affecting the gains observed in our experiments.

Finally, in Figure 6, we present the measurements of HDFS bytes read, network traffic and repair duration versus the number of blocks lost, for all three experiments carried out in EC2. We also plot the linear least squares fitting curve for these measurements.

5.2.1 HDFS Bytes Read

Figure 4a depicts the total number of HDFS bytes read by the BlockFixer jobs initiated during each failure event. The bar plots show that HDFS-Xorbas reads $41\% - 52\%$ the amount of data that RS reads to reconstruct the same number of lost blocks. These measurements are consistent with the theoretically expected values, given that more than one blocks per stripe are occasionally lost (note that $12.14/5 = 41\%$). In Figure 6a it is shown that the number of HDFS bytes read is linearly dependent on the number of blocks lost, as expected. The slopes give us the average number of HDFS bytes read per block for Xorbas and HDFS-RS. The average number of blocks read per lost block are estimated to be 11.5 and 5.8, showing the $2\times$ benefit of HDFS-Xorbas.

5.2.2 Network Traffic

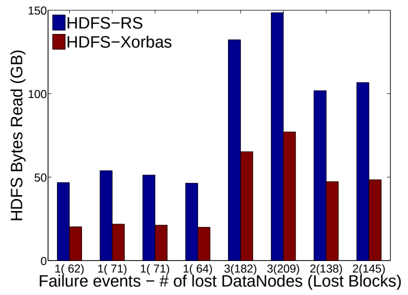
Figure 4b depicts the network traffic produced by the BlockFixer jobs during the entire repair procedure. In particular, it shows the outgoing network traffic produced in the cluster, aggregated across instances. Incoming network traffic is similar since the cluster only communicates information internally. Throughout our experiments, we consistently observed that network traffic was roughly equal to twice the number of bytes read. Therefore, gains in the number of HDFS bytes read, immediately translate to network traffic gains. In Figure 5a, we present the Network Traffic plotted continuously during the course of the 200 file experiment, with a 5-minute resolution. The sequence of failure events is clearly visible. The fact that the traffic peaks of the two systems are different is an indication that the available bandwidth was not saturated. However, the bottleneck in MapReduce tasks is reportedly the network [5, 16, 17]. This is due to the fact that when the amount of data increases, more MapReduce tasks need to run in parallel, draining network resources. In these large scale environments, link saturation is expected to limit the data transfer rates and we expect higher recovery times for HDFS-RS.

5.2.3 Repair Duration

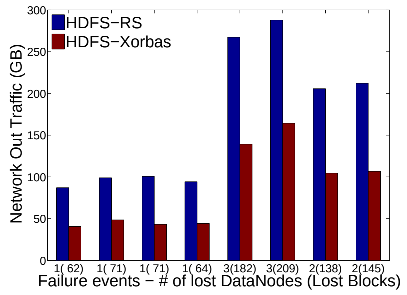
Figure 4c depicts the total duration of the recovery procedure *i.e.*, the interval from the launch time of the first block fixing job to the termination of the last one. Combining measurements from all the experiments, Figure 6c shows the repair duration versus the number of blocks repaired. These figures show that Xorbas finishes 25% to 45% faster than HDFS-RS. This is primarily due to the reduced amount of data that need to be downloaded during repair compared to HDFS-RS. From the CPU Utilization plots we conclude that HDFS RS and Xorbas have very similar CPU requirements and this does not seem to influence the repair times.

5.3 Social Network X's cluster

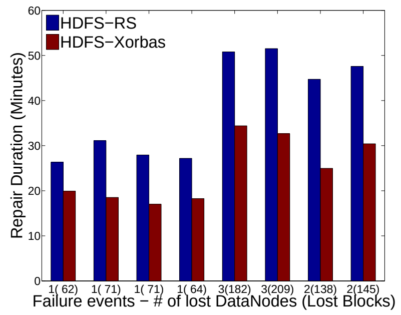
In addition to the series of controlled experiments performed over EC2, we performed one more experiment on Social Network X's test cluster. This test cluster consisted of 35 nodes configured with a total capacity of 370 TB. Instead of placing files of pre-determined sizes as we did in EC2, we utilized the existing set of files in the cluster: 3,262 files, totaling to approximately 2.7 TB of logical data. The block size used was 256 MB (same as in X's production clusters). Roughly 94% of the files consisted of 3 blocks and the remaining of 10 blocks, leading to an average 3.4 blocks per



(a) HDFS Bytes Read per failure event.

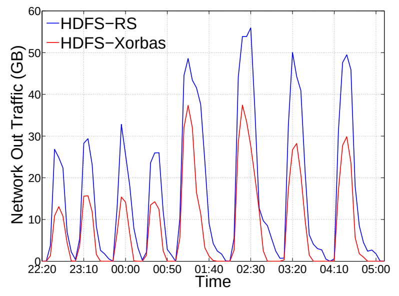


(b) Network Out Traffic per failure event.

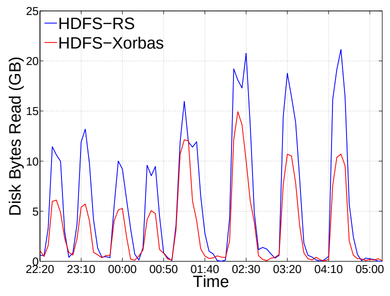


(c) Repair duration per failure event.

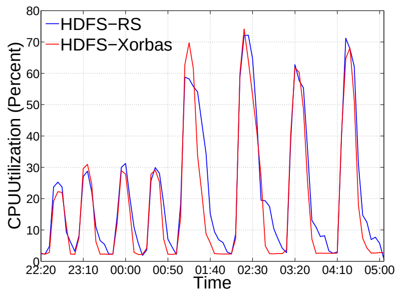
Figure 4: The metrics measured during the 200 file experiment. Network-in is similar to Network-out and so it is not displayed here. During the course of the experiment, we simulated eight failure events and the x-axis gives details of the number of DataNodes terminated during each failure event and the number of blocks lost are displayed in parentheses.



(a) Cluster network traffic.



(b) Cluster Disk Bytes Read.



(c) Cluster average CPU utilization.

Figure 5: Measurements in time from the two EC2 clusters during the sequence of failing events.

file.

	Blocks Lost	HDFS GB read Total	HDFS GB read /block	Repair Duration
RS	369	486.6	1.318	26 min
Xorbas	563	330.8	0.58	19 min

Table 2: Experiment on Social Network X's Cluster Results.

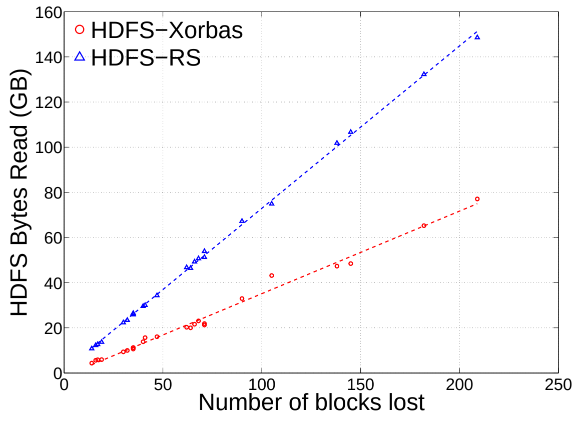
For our experiment, HDFS-RS was deployed on the cluster and upon completion of data RAIDing, a random DataNode was terminated. HDFS Bytes Read and the Repair Duration measurements were collected. Unfortunately, we did not have access to Network Traffic measurements. The experiment was repeated, deploying HDFS-Xorbas on the same set-up. Results are shown in Table 2. Note that in this experiment, HDFS-Xorbas stored 27% more than HDFS-RS (ideally, the overhead should be 13%), due to the small size of the majority of the files stored in the cluster. As noted before, files typically stored in HDFS are large (and small files are typically archived into large HAR files). Further, it may be emphasized that the particular dataset used for this experiment is by no means representative of the dataset stored in Social Network X's production clusters.

In this experiment, the number of blocks lost in the second run, exceed those of the first run by more than the storage overhead introduced by HDFS-Xorbas. However, we still observe benefits in the amount of data read and repair duration, and the gains are even more clearer when normalizing by the number of blocks lost.

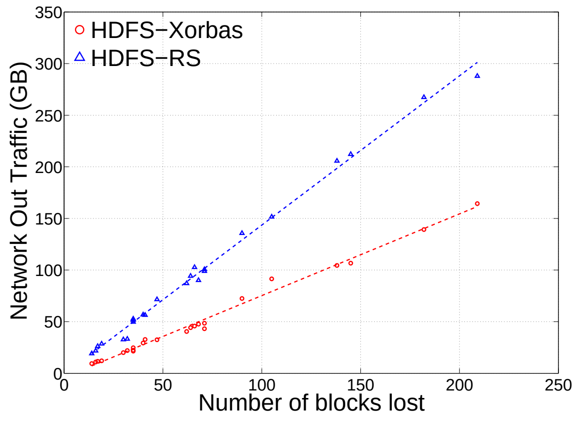
6. RELATED WORK

Optimizing erasure code designs for efficient single block repair is a topic that has recently attracted significant attention due to its relevance to distributed systems. Dimakis *et al.* [7] showed that the repair network traffic can be significantly reduced but at a cost of higher disk I/O. Subsequent work has focused on achieving these information theoretic bounds with explicit code constructions (see *e.g.* [8, 27, 30, 24]). While this work is closely related, it requires processing at the surviving nodes before transmission, a feature that can increase the required disk I/O. Further, most constructions require exponential field size and sub-packetization to achieve the information theoretic bounds. To the best of our knowledge there are currently no known practical regenerating codes with low complexity for storage overheads below 2, which is the regime of interest.

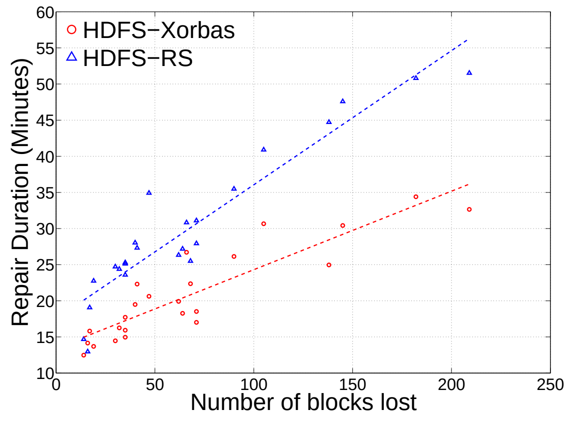
A second line of work has focused on optimizing repair disk I/O and network bandwidth with non-MDS code constructions (e.g. [19, 23, 12]) The codes we introduce are optimal in terms of their locality and match the bound shown in [12]. In our BBB recent prior work [**?**] we generalize the locality bounds of [12] and define locally repairable codes that can store more slightly more information per block,



(a) HDFS Bytes Read versus blocks lost



(b) Network-Out Traffic



(c) Repair Duration versus blocks lost

Figure 6: Measurement points of failure events versus the total number of blocks lost in the corresponding events. Measurements are from all three experiments. Linear least squares curves.

defining a tradeoff between storage, distance and locality.

The main theoretical innovation of this paper is an deterministic construction of LRCs that relies on Reed-Solomon global parities. We show how the concept of implied parities can save storage and show how to explicitly achieve parity alignment if the global parities are Reed-Solomon. Further, we present a system implementation and measurements showing the network and disk IO benefits.

7. CONCLUSIONS

Modern storage systems are transitioning to erasure coding. We introduced a new family of codes called Locally Repairable Codes (LRCs) that have marginally suboptimal storage but significantly smaller repair disk I/O and network bandwidth requirements. In our implementation, we observed 2× disk I/O and network reduction for the cost of 14% more storage, a price that seems reasonable for many scenarios.

One related area where we believe locally repairable codes can have a significant impact is purely archival clusters. In this case we can deploy large LRCs (*i.e.*, stipe sizes of 50 or 100 blocks) that can simultaneously offer high fault tolerance and small storage overhead. This would be impractical if Reed-Solomon codes are used since the repair traffic grows linearly in the stripe size. Local repairs would further allow spinning disks down [6] since very few are required for single block repairs.

In conclusion, we believe that LRCs create a new operating point that will be practically relevant in large-scale cloud storage systems, especially when the network bandwidth is the main performance bottleneck.

8. ADDITIONAL AUTHORS

9. REFERENCES

[1] Amazon EC2. *http://aws.amazon.com/ec2/*.

[2] Apache HDFS. *http://hadoop.apache.org/hdfs*.

[3] HDFS-RAID wiki. *http://wiki.apache.org/hadoop/HDFS-RAID*.

[4] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, et al. Windows azure storage: A highly available cloud storage service with strong consistency. In *SOSP*, 2011.

[5] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I. Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. *SIGCOMM Comput. Commun. Rev.*, 41:98–109, August 2011.

[6] A. Donnelly D. Narayanan and A. Rowstron. Write off-loading: Practical power management for enterprise storage. *Proceedings of 6th USENIX Conference on File and Storage Technologies (FAST)*, 2008.

[7] A.G. Dimakis, P.B. Godfrey, Y. Wu, M.J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *Information Theory, IEEE Transactions on*, 56(9):4539–4551, 2010.

[8] 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, 2011.

[9] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson. Diskreduce: Raid for data-intensive scalable computing. In *Proceedings of the 4th Annual*

Workshop on Petascale Data Storage, pages 6–10. ACM, 2009.

[10] D. Ford, F. Labelle, F.I. Popovici, M. Stokely, V.A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *Proc. of OSDI*, 2010.

[11] S. Ghemawat, H. Gobioff, and S.T. Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[12] Parikshit Gopalan, Cheng Huang, Huseyin Simitci, and Sergey Yekhanin. On the locality of codeword symbols. *CoRR*, abs/1106.3625, 2011.

[13] K.M. Greenan. *Reliability and power-efficiency in erasure-coded storage systems*. PhD thesis, University of California, Santa Cruz, December 2009.

[14] K.M. Greenan, J.S. Plank, and J.J. Wylie. Mean time to meaningless: Mttdl, markov models, and storage system reliability. In *Proceedings of the 2nd USENIX conference on Hot topics in storage and file systems*, pages 5–5. USENIX Association, 2010.

[15] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. *Computer Communications Review (CCR)*, 2009.

[16] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, 39:51–62, August 2009.

[17] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. *SIGCOMM Comput. Commun. Rev.*, 38:75–86, August 2008.

[18] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, October 2006.

[19] C. Huang, M. Chen, and J. Li. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *IEEE International Symposium on Network Computing and Applications (NCA)*, 2007.

[20] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen. Polynomial time algorithms for multicast network code construction. *Information Theory, IEEE Transactions on*, 51(6):1973–1982, 2005.

[21] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and Jon Crowcroft. XORs in the air: Practical wireless network coding. *Proc. of ACM SIGCOMM*, 2006.

[22] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang. Rethinking erasure codes for cloud file systems: Minimizing i/o for recovery and degraded reads. In *Usenix Conference on File and Storage Technologies (FAST)*, 2012.

[23] O. Khan, R. Burns, J. S. Plank, and C. Huang. In search of I/O-optimal recovery from disk failures. In *HotStorage '11: 3rd Workshop on Hot Topics in Storage and File Systems*, Portland, June 2011. USENIX.

[24] J. C.S. Lui L. Xiang, Y. Xu and Q. Chang. Optimal recovery of single disk failure in rdp code storage systems. *ACM SIGMETRICS Performance Evaluation Review*, 2010.

[25] F. Oggier and A. Datta. Self-repairing homomorphic codes for distributed storage systems. In *INFOCOM, 2011 Proceedings IEEE*, pages 1215 –1223, april 2011.

[26] D.S. Papailiopoulos, J. Luo, A.G. Dimakis, C. Huang, and J. Li. Simple regenerating codes: Network coding for cloud storage. *Arxiv preprint arXiv:1109.0264*, 2011.

[27] K.V. Rashmi, N.B. Shah, and P.V. Kumar. Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction. *Information Theory, IEEE Transactions on*, 57(8):5227 –5239, aug. 2011.

[28] I.S. Reed and G. Solomon. Polynomial codes over certain finite fields. In *Journal of the SIAM*, 1960.

[29] R. Rodrigues and B. Liskov. High availability in dhts: Erasure coding vs. replication. *Peer-to-Peer Systems IV*, pages 226–239, 2005.

[30] Itzhak Tamo, Zhiying Wang, and Jehoshua Bruck. Mds array codes with optimal rebuilding. *CoRR*, abs/1103.3737, 2011.

[31] S. B. Wicker and V. K. Bhargava. Reed-solomon codes and their applications. In *IEEE Press*, 1994.

[32] Q. Xin, E.L. Miller, T. Schwarz, D.D.E. Long, S.A. Brandt, and W. Litwin. Reliability mechanisms for very large storage systems. In *Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*, pages 146–156. IEEE, 2003.

10. APPENDIX A

10.1 Distance and Locality through Entropy

In the following, we use a characterization of the code distance d of a length n code that is based on the entropy function. This characterization is universal in the sense that it covers any linear or nonlinear code designs.

Let $\mathbf{x}$ be a file of size M that we wish to split and store with redundancy $\frac{k}{n}$ in n blocks, where each block has size $\frac{M}{k}$. Without loss of generality, we assume that the file is split in k blocks of the same size $\mathbf{x} \overset{\triangle}{=} [X_1 \dots X_k] \in \mathbb{F}^{1\times k}$, where $\mathbb{F}$ is the finite field over which all operations are performed. The entropy of each file block is $H(X_i) = \frac{M}{k}$, for all $i \in [k]$, where $[n] = \{1, \dots, n\}$.[2] Then, we define an encoding (generator) map $G : \mathbb{F}^{1\times k} \mapsto \mathbb{F}^{1\times n}$ that takes as input the k file blocks and outputs n coded blocks $G(\mathbf{x}) = \mathbf{y} = [Y_1 \dots Y_n]$, where $H(Y_i) = \frac{M}{k}$, for all $i \in [n]$. The encoding function G defines a $(k, n-k)$ code $\mathcal{C}$ over the vector space $\mathbb{F}^{1\times n}$. We can calculate the effective rate of the code as the ratio of the entropy of the file blocks to the sum of the entropies of the n coded blocks

$$R = \frac{H(X_1, \dots, X_k)}{\sum_{i=1}^{n} H(Y_i)} = \frac{k}{n}. \qquad (3)$$

The distance d of the code $\mathcal{C}$ is equal to the minimum number of erasures of blocks in $\mathbf{y}$ after which the entropy of

[2]Equivalently, each block can be considered as a random variable that has entropy $\frac{M}{k}$.

the remaining blocks is strictly less than M

$$d = \min_{H(\{Y_1,\dots,Y_n\}\setminus\mathcal{E})<M} |\mathcal{E}| = n - \max_{H(\mathcal{S})<M} |\mathcal{S}|, \qquad (4)$$

where $\mathcal{E} \in 2^{\{Y_1,\dots,Y_n\}}$ is a block erasure pattern set and $2^{\{Y_1,\dots,Y_n\}}$ denotes the power set of $\{Y_1,\dots,Y_n\}$, i.e., the set that consists of all subset of $\{Y_1,\dots,Y_n\}$. Hence, for a code $\mathcal{C}$ of length n and distance d, any $n-d+1$ coded blocks can reconstruct the file, i.e., have joint entropy at least equal to M. It follows that when d is given, $n-d$ is the maximum number of coded variables that have entropy less than M.

The locality r of a code can also be defined in terms of coded block entropies. When a coded block Y_i, $i \in [n]$, has locality r, then it is a function of r other coded variables $Y_i = f_i(Y_{\mathcal{R}(i)})$, where $\mathcal{R}(i)$ indexes the set of r blocks Y_j, $j \in \mathcal{R}(i)$, that can reconstruct Y_i, and f_i is some function (linear or nonlinear) on these r coded blocks. Hence, the entropy of Y_i conditioned on its repair group $\mathcal{R}(i)$ is identically equal to zero $H(Y_i|f_i(Y_{\mathcal{R}(i)})) = 0$, for $i \in [n]$. This functional dependency of Y_i on the blocks in $\mathcal{R}(i)$ is fundamentally the only code structure that we assume in our derivations.[3] This generality is key to providing universal information theoretic bounds on the code distance of $(k, n-k)$ linear, or nonlinear, codes that have locality r. Our following bounds can be considered as generalizations of the Singleton Bound on the code distance when locality is taken into account.

10.2 Information theoretic limits of Locality and Distance

We consider $(k, n-k)$ codes that have block locality r. We find a lower bound on the distance by lower bounding the largest set $\mathcal{S}$ of coded blocks whose entropy is less than M, i.e., a set that cannot reconstruct the file. Effectively, we solve the following optimization problem that needs to be performed over all possible codes $\mathcal{C}$ and yields a best-case minimum distance

$$\min_{\mathcal{C}} \max_{\mathcal{S}} |\mathcal{S}| \text{ s.t.: } H(\mathcal{S}) < M, \;\; \mathcal{S} \in 2^{\{Y_1,\dots,Y_n\}}.$$

We are able to provide a bound by considering a single property: each block is a member of a repair group of size $r+1$.

Definition 3. *For a code $\mathcal{C}$ of length n and locality r, a coded block Y_i along with the blocks that can generate it, $Y_{\mathcal{R}(i)}$, form a repair group $\Gamma(i) = \{i, \mathcal{R}(i)\}$, for all $i \in [n]$. We refer to these repair groups, as $(r+1)$-groups.*

It is easy to check that the joint entropy of the blocks in a single $(r+1)$-group is at most as much as the entropy of r file blocks

$$\begin{aligned} H\left(Y_{\Gamma(i)}\right) &= H\left(Y_i, Y_{\mathcal{R}(i)}\right) = H\left(Y_{\mathcal{R}(i)}\right) + H\left(Y_i | Y_{\mathcal{R}(i)}\right) \\ &= H\left(Y_{\mathcal{R}(i)}\right) \le \sum_{j\in\mathcal{R}(i)} H(Y_j) = r\frac{M}{k}, \end{aligned}$$

for all $i \in [n]$. To determine the upper bound on minimum distance of $\mathcal{C}$, we construct the maximum set of coded blocks $\mathcal{S}$ that has entropy less than M. We use this set to derive the following theorem.

[3]In the following, we consider codes with uniform locality, i.e., $(k, n-k)$ codes where all encoded blocks have locality r. These codes are referred to as non-canonical codes in [12].

Theorem 2. *For a code $\mathcal{C}$ of length n, where each coded block has entropy $\frac{M}{k}$ and locality r, the minimum distance is bounded as*

$$d \le n - \left\lceil \frac{k}{r} \right\rceil - k + 2. \qquad (5)$$

Proof: Our proof follows the same steps as the one in [12]. We start by building the set $\mathcal{S}$ in steps and denote the collection of coded blocks at each step as $\mathcal{S}_i$. The algorithm that builds the set is in Fig. 7. The goal is to lower bound the cardinality of $\mathcal{S}$, which results in an upper bound on code distance d, since $d \le n - |\mathcal{S}|$. At each step we denote the difference in cardinality of $\mathcal{S}_i$ ans $\mathcal{S}_{i-1}$ and the difference in entropy as $s_i = |\mathcal{S}_i| - |\mathcal{S}_{i-1}|$ and $h_i = H(\mathcal{S}_i) - H(\mathcal{S}_{i-1})$, respectively.

step	
1	Set $\mathcal{S}_0 = \emptyset$ and $i = 1$
2	`WHILE` $H(\mathcal{S}_{i-1}) < M$
3	Pick a coded block $Y_j \notin \mathcal{S}_{i-1}$
4	`IF` $H(\mathcal{S}_{i-1} \cup \{Y_{\Gamma(j)}\}) < M$
5	set $\mathcal{S}_i = \mathcal{S}_{i-1} \cup Y_{\Gamma(j)}$
6	`ELSE IF` $H(\mathcal{S}_{i-1} \cup \{Y_{\Gamma(j)}\}) \ge M$
7	pick $\mathcal{Y}_s \subset Y_{\Gamma(j)}$ s.t. $H(\mathcal{Y}_s \cup \mathcal{S}_{i-1}) < M$
8	set $\mathcal{S}_i = \mathcal{S}_{i-1} \cup \mathcal{Y}_s$
9	$i = i + 1$

Figure 7: The algorithm that builds set $\mathcal{S}$.

At each step (depending on the possibility that two $(r+1)$-groups overlap) the difference in cardinalities s_i is bounded as $1 \le s_i \le r+1$, that is $s_i = r+1-p$, where $\left|\{Y_{\Gamma(j)}\} \cap \mathcal{S}_{i-1}\right| = p$. Now there exist two possible cases. First, the case where the last step set $\mathcal{S}_l$ is generated by line 5. For this case we can also bound the entropy as $h_i \le (s_i - 1)\frac{M}{k} \Leftrightarrow s_i \ge \frac{k}{M}h_i + 1$ which comes from the fact that, at least one coded variable in $\{Y_{\Gamma(j)}\}$ is a function of variables in $\mathcal{S}_{i-1} \cup Y_{\mathcal{R}(j)}$. Now, we can bound the cardinality $|\mathcal{S}_l| = \sum_{i=1}^{l} s_i \ge \sum_{i=1}^{l}\left(\frac{kh_i}{M} + 1\right) = l + \frac{k}{M}\sum_{i=1}^{l} h_i$. We now have to bound l and $\sum_{i=1}^{l} h_i$. First, observe that since l is our "last step," this means that the aggregate entropy in $\mathcal{S}_l$ should be less than the file size, i.e., it should have a value $M - c \cdot \frac{M}{k}$, for $0 < c \le 1$. If $c > 1$ then we could collect another variable in that set. On the other hand, if $c = 0$, then the coded blocks in $\mathcal{S}_l$ would have been sufficient to reconstruct the file. Hence, $M - \frac{M}{k} \le \sum_{i=1}^{l} h_i < M$. We shall now lower bound l. The smallest $l' \le l$ (i.e., the fastest) upon which $\mathcal{S}_{l'}$ reaches an aggregate entropy that is greater than, or equal to M, can be found in the following way: if we could only collect $(r+1)$-groups of entropy $r\frac{M}{k}$, without "entropy losses" between these groups, i.e., if there were no further dependencies than the ones dictated by locality, then we would stop just before $\mathcal{S}_{l'}$ reached an entropy of M, that is $\sum_{i=1}^{l'} h_{l'} < M \Leftrightarrow l'r\frac{M}{k} < M \Leftrightarrow l' < \left\lceil \frac{k}{r} \right\rceil$. However, l' is an integer, hence $l' = \left\lceil \frac{k}{r} \right\rceil - 1$. We apply the above to bound the cardinality $|\mathcal{S}_l| \ge k - 1 + l' \ge k - 1 + \left\lceil \frac{k}{r} \right\rceil - 1 = k + \left\lceil \frac{k}{r} \right\rceil - 2$, in which case we obtain $d \le n - \left\lceil \frac{k}{r} \right\rceil - k + 2$.

We move to the second case where we reach line 6 of the building algorithm: the entropy of the file can be covered only by collecting $(r+1)$ groups. This depends on the remainder of the division of M by $r\frac{M}{k}$. Posterior to collecting the $(r+1)$-groups, we are left with some entropy that needs to be covered by at most $r-1$ additional blocks not in $\mathcal{S}_{l'}$. The entropy not covered by the set $\mathcal{S}_{l'}$ is $M - l'r\frac{M}{k} = M - \left(\left\lceil \frac{k}{r} \right\rceil - 1\right) r\frac{M}{k} = M - \left\lceil \frac{k}{r} \right\rceil \frac{M}{k} + r\frac{M}{k}$. To cover that we need an additional number of blocks $s \geq \left\lceil \frac{M - l'r\frac{M}{k}}{\frac{M}{k}} \right\rceil = k - l'r = k - \left(\left\lceil \frac{k}{r} \right\rceil - 1\right) r$. Hence, our final set $\mathcal{S}_l$ has size

$$|\mathcal{S}_l| + s - 1 = l(r+1) + s - 1 \geq l'(r+1) + k - \left(\left\lceil \frac{k}{r} \right\rceil - 1\right) - 1$$
$$= \left(\left\lceil \frac{k}{r} \right\rceil - 1\right)(r+1) + k - r\left(\left\lceil \frac{k}{r} \right\rceil - 1\right) - 1 = \left\lceil \frac{k}{r} \right\rceil + k - 2.$$

Again, due to the fact that the distance is bounded by $n - |\mathcal{S}|$ we have $d \leq n - \left\lceil \frac{k}{r} \right\rceil - k + 2$. □

From the above proof we obtain the following corollary.

COROLLARY 2. *In terms of the code distance, non-overlapping $(r+1)$-groups are optimal.*

In [12], it was proven that $(k, n-k)$ linear codes have minimum code distance that is bounded as $d \leq n - k - \left\lceil \frac{k}{r} \right\rceil + 2$. As we see from our distance-locality bound, the limit of linear codes is information theoretic optimal, i.e., linear codes suffice to achieve it. Indeed, in the following we show that the distance bound is tight and we present randomized and explicit codes that achieve it.[4]

10.3 Achievability of the Bound

In this section, we show that the bound of Theorem 2 is achievable using a random linear network coding (RLNC) approach as the one presented in [18] Our proof uses a variant of the information flow graph that was introduced in [7]. We show that a distance d is feasible if a cut-set bound on this new flow graph is sufficiently large for multicast sessions to run on it.

In the same manner as [7], the information flow graph represents a network where the k input blocks are depicted as sources, the n coded blocks are represented as intermediate nodes of the network, and the sinks of the network are nodes that need to decode the k file blocks. The innovation of the new flow graph is that it is "locality aware" by incorporating an appropriate dependency subgraph that accounts for the existence of repair groups of size $(r+1)$. The specifications of this network, i.e., the number and degree of blocks, the edge-capacities, and the cut-set bound are all determined by the code parameters $k, n-k, r, d$. For coding parameters that do not violate the distance bound in Theorem 2, the minimum $s-t$ cut of such a flow graph is at least M. The multicast capacity of the induced network is achievable using random linear network codes. This achievability scheme corresponds to a *scalar linear* code with parameters $k, n-k, r, d$.

In Fig. 8, we show the general structure of an information flow graph. We refer to this *directed* graph as $\mathcal{G}(k, n-k, r, d)$ with vertex set

$$\mathcal{V} = \Big\{\{X_i; i \in [k]\}, \ \left\{\Gamma_j^{\text{in}}, \Gamma_j^{\text{out}}; j \in [n]\right\},$$
$$\left\{Y_j^{\text{in}}, Y_j^{\text{out}}; j \in [n]\right\}, \{\text{DC}_l; \forall l \in [T]\}\Big\}.$$

[4]In our following achievability proof of the above information theoretic bound we assume that $(r+1)|n$ and we consider non-overlapping repair groups. This means that $\Gamma(i) \equiv \Gamma(j)$ for all $i, j \in \Gamma(i)$.

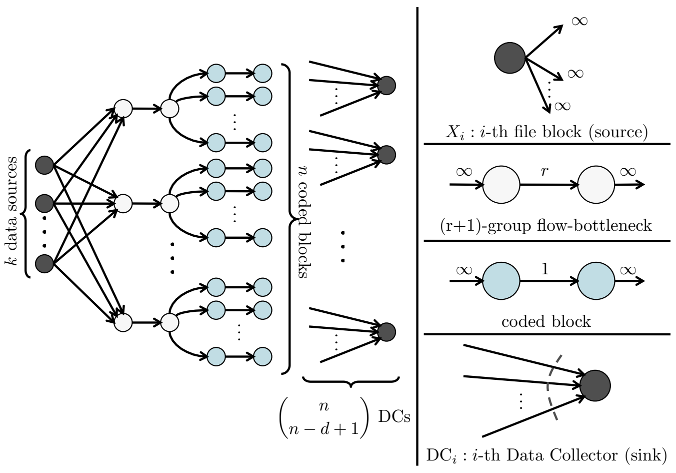

k data sources
n coded blocks
$\binom{n}{n-d+1}$ DCs
∞
∞
∞
X_i : i-th file block (source)
∞
r
∞
(r+1)-group flow-bottleneck
∞
1
∞
coded block
DC$_i$: i-th Data Collector (sink)

Figure 8: The $\mathcal{G}(k, n-k, r, d)$ information flow graph.

The directed edge set is implied by the following edge capacity function

$$c_e(v,u) = \begin{cases} \infty, (v,u) \in \left(\{X_i; i \in [k]\}, \left\{\Gamma_j^{\text{in}}; j \in \left[\frac{n}{r+1}\right]\right\}\right) \\ \quad \cup \left(\left\{\Gamma_j^{\text{out}}; j \in \left[\frac{n}{r+1}\right]\right\}, \left\{Y_j^{\text{in}}; j \in [n]\right\}\right) \\ \quad \cup \left(\left\{Y_j^{\text{out}} j \in [n]\right\}, \ \{\text{DC}_l; l \in [T]\}\right), \\ \frac{M}{k}, (v,u) \in \left(\left\{Y_j^{\text{in}} j \in [n]\right\}, \ \left\{Y_j^{\text{out}} j \in [n]\right\}\right), \\ 0, \text{otherwise}. \end{cases}$$

The vertices $\{X_i; i \in [k]\}$ correspond to the k file blocks and $\left\{Y_j^{\text{out}}; j \in [n]\right\}$ correspond to the coded blocks. The edge capacity between the in- and out- Y_i vertices corresponds to the entropy of a single coded block. When, $r+1$ blocks are elements of a group, then their "joint flow," or entropy, cannot exceed $r\frac{M}{k}$. To enforce this entropy constraint, we bottleneck the in-flow of each group by a node that restricts it to be at most $r\frac{M}{k}$. For a group $\Gamma(i)$, we add node Γ_i^{in} that receives flow by the sources and is connected with an edge of capacity $r\frac{M}{k}$ to a new node Γ_i^{out}. The latter connects to the $r+1$ blocks of the i-th group. The file blocks travel along the edges of this graph towards the sinks, which we call Data Collectors (DCs). A DC needs to connect to as many coded blocks as such that it can reconstruct the file. This is equivalent to requiring $s-t$ cuts between the file blocks and the DCs that are at least equal to M, i.e., the file size. We should note that when we are considering a specific group, we know that any block within that group can be repaired from the remaining r blocks. When a block is lost, the functional dependence among the blocks in an $(r+1)$-group allow a newcomer block to compute a function on the remaining r blocks and reconstruct what was lost.

Observe that if the distance of the code is d, then there are $T = \binom{n}{n-d+1}$ DCs, each with in-degree $n-d+1$, whose incident vertices originate from $n-d+1$ blocks. The cut-set bound of this network is defined by the set of minimum cuts between the file blocks and each of the DCs. A source-DC cut in $\mathcal{G}(k, n-k, r, d)$ determines the amount of flow that travels from the file blocks to the DCs. When d is consistent with the bound of Theorem 2, the minimum of all the $s-t$ cuts is at least as much as the file size M. The following lemma states that if d is consistent with the bound of Theorem 2, then the minimum of all the cuts is at least as much as the file size M.

Lemma 2. *The minimum source-DC cut in $\mathcal{G}(k, n-k, r, d)$ is at least M, when $d \leq n - \lceil \frac{k}{r} \rceil - k + 2$.*

Proof : Omitted due to lack of space. □
Lemma 2 verifies that for given n, k, r, and a valid distance d according to Theorem 2, the information flow graph is consistent with the bound: the DCs have enough entropy to decode all file blocks, when the minimum cut is more than M. The above results imply that the flow graph $\mathcal{G}(k, n-k, r, d)$ captures both the blocks locality and the DC requirements. Then, a successful multicast session on $\mathcal{G}(k, n-k, r, d)$ is equivalent to all DCs decoding the file.

Theorem 3. *If a multicast session on $\mathcal{G}(k, n-k, r, d)$ is feasible, then there exist a $(k, n-k)$ code $\mathcal{C}$ of locality r and distance d .*

Hence, the random linear network coding (RLNC) scheme of Ho *et al.* [18] achieves the cut-set bound of $\mathcal{G}_r(k, n-k, r, d)$, i.e., there exist capacity achieving network codes, which implies that there exist codes that achieve the distance bound of Theorem 2. Instead of the RLNC scheme, we could use the deterministic construction algorithm of Jaggi *et al.* [20] to construct explicit capacity achieving linear codes for multicast networks. Using that scheme, we could obtain in time polynomial in T explicit $(k, n-k)$ codes of locality r.

Lemma 3. *For a network with E edges, k sources, and T destinations, where η links transmit linear combination of inputs, the probability of success of the RLNC scheme is at least $\left(1 - \frac{T}{q}\right)^{\eta}$. Moreover, using the algorithm in [20], a deterministic linear code over $\mathbb{F}$ can be found in time $\mathcal{O}\left(ETk(k+T)\right)$.*

The number of edges in our network is $E = \frac{n(k+2r+3)}{r+1} + (n-d+1)\binom{n}{k+\lceil \frac{k}{r} \rceil - 1}$ hence we can calculate the complexity order of the deterministic algorithm, which is $ETk(k+T) = \mathcal{O}\left(T^3k^2\right) = \mathcal{O}\left(k^2 8^{nH_2\left(\frac{r}{(r+1)R}\right)}\right)$, where $H_2(\cdot)$ is the binary entropy function. The above and Lemma 3 give us the following existence theorem

Theorem 4. *There exists a linear code over $\mathbb{F}$ with locality r and length n, such that $(r+1)|n$, that has distance $d = n - \lceil \frac{k}{r} \rceil - k + 2$, if $|\mathbb{F}| = q > \binom{n}{k+\lceil \frac{k}{r} \rceil - 1} = \mathcal{O}\left(2^{nH_2\left(\frac{r}{(r+1)R}\right)}\right)$. Moreover, we can construct explicit codes over $\mathbb{F}$, with $|\mathbb{F}| = q$, in time $\mathcal{O}\left(k^2 8^{nH_2\left(\frac{r}{(r+1)R}\right)}\right)$.*

Observe that by setting $r = \log(k)$, we obtain Theorem 1. Moreover, we would like to note that if for each $(r+1)$-group we "deleted" a coded block, then the remaining code would be a $(k, n'-k)$-MDS code, where $n' = n - \frac{n}{r+1}$, assuming no repair group overlaps. This means that LRCs are constructed on top of MDS codes by adding r-degree parity coded blocks. A general construction that operated over small fields and could be constructed in time polynomial in the number of DCs is an interesting open problem.

10.4 An Explicit LRC using Reed-Solomon Parities

We design a $(10, 6, 5)$-LRC based on Reed-Solomon Codes and Interference Alignment. We use as a basis for that a $(10, 4)$-RS code defined over a binary extension field $\mathbb{F}_{2^m}$. We concentrate on these specific instances of RS codes since these are the ones that are implemented in practice and in particular in the HDFS RAID component of Hadoop. We continue introducing a general framework for the desing of $(k, n-k)$ Reed-Solomon Codes.

The $k \times n$ (Vandermonde type) parity-check matrix of a $(k, n-k)$-RS code defined over an extended binary field $\mathbb{F}_{2^m}$, of order $q = 2^m$, is given by $[\mathbf{H}]_{i,j} = a_{j-1}^{i-1}$, where $a_0, a_1, \ldots, a_{n-1}$ are n distinct elements of the field $\mathbb{F}_{2^m}$. The order of the field has to be $q \geq n$. The $n-1$ coefficients $a_0, a_1, \ldots, a_{n-1}$ are n *distinct* elements of the field $\mathbb{F}_{2^m}$. We can select α to be a generator element of the cyclic multiplicative group defined over $\mathbb{F}_{2^m}$. Hence, let α be a primitive element of the field $\mathbb{F}_{2^m}$. Then, $[\mathbf{H}]_{i,j} = \alpha^{(i-1)(j-1)}$, for $i \in [k], j \in [n]$. The above parity check matrix defines a $(k, n-k)$-RS code. It is a well-known fact, that due to its determinant structure, any $(n-k) \times (n-k)$ submatrix of $\mathbf{H}$ has a nonzero determinant, hence, is full-rank. This, in terms, means that a $(k, n-k)$-RS defined using the parity check matrix $\mathbf{H}$ is an MDS code, i.e., has optimal minimum distance $d = n - k + 1$. We refer to the $k \times n$ generator matrix of this code as $\mathbf{G}$.

Based on a $(14, 10)$-RS generator matrix, we will introduce 2 simple parities on the first 5 and second 5 coded blocks of the RS code. This, will yield the generator matrix of our LRC

$$\mathbf{G}_{\text{LRC}} = \left[\mathbf{G} \left| \sum_{i=1}^{5} \mathbf{g}_i \quad \sum_{i=6}^{10} \mathbf{g}_i \right.\right], \tag{6}$$

where $\mathbf{g}_i$ denotes the i-th column of $\mathbf{G}$, for $i \in [14]$. We would like to note that even if $\mathbf{G}_{\text{LRC}}$ is not in systematic form, i.e., the first 10 blocks are not the initial file blocks, we can easily convert it into one. To do so we need to apply a full-rank transformation on the rows of $\mathbf{G}_{\text{LRC}}$ in the following way: $\mathbf{A}\mathbf{G}_{\text{LRC}} = \mathbf{A}\left[\mathbf{G}_{:,1:10} \;\; \mathbf{G}_{:,11:15}\right] = \left[\mathbf{I}_{10} \;\; \mathbf{A}\mathbf{G}_{:,11:15}\right]$, where $\mathbf{A} = \mathbf{G}_{:,1:10}^{-1}$ and $\mathbf{G}_{:,i:j}$ is a submatrix of $\mathbf{G}$ that consists of columns with indices from i to j. This transformation renders our code systematic, while retaining its distance and locality properties. We proceed to the main result of this section.

Theorem 5. *The code $\mathcal{C}$ of length 16 defined by $\mathbf{G}_{LRC}$ has locality 5 for all coded blocks and optimal distance $d = 5$.*

Proof: We first prove that all coded blocks of $\mathbf{G}_{\text{LRC}}$ have locality 5. Instead of considering block locality, we can equivalently consider the locality of the columns of $\mathbf{G}_{\text{LRC}}$, without loss of generality. First let $i \in [5]$. Then, $\mathbf{g}_i$ can be reconstructed from the XOR parity $\sum_{j=1}^{5} \mathbf{g}_j$ if the 4 other columns $\mathbf{g}_i$, $j \in \{6, \ldots, 10\} \backslash i$, are subtracted from it. The same goes for $i \in \{6, \ldots, 10\}$, i.e., $\mathbf{g}_i$ can be reconstructed by subtracting $\mathbf{g}_j$, for $j \in \{6, \ldots, 10\} \backslash i$, from the XOR parity $\sum_{j=6}^{10} \mathbf{g}_j$. However, it is not straightforward how to repair the last 4 coded blocks, i.e., the parity blocks of the systematic code representation. At this point we make use of Interference Alignment. Specifically, we observe the following: since the all-ones vector of length n is in the span of the rows of the parity check matrix $\mathbf{H}$, then it has to be orthogonal to the generator matrix $\mathbf{G}$, i.e., $\mathbf{G}\mathbf{1}^T = \mathbf{0}_{k \times 1}$ due to the fundamental property $\mathbf{G}\mathbf{H}^T = \mathbf{0}_{k \times (n-k)}$. This means that $\mathbf{G}\mathbf{1}^T = \mathbf{0}_{k \times 1} \Leftrightarrow \sum_{i=1}^{14} \mathbf{g}_i = \mathbf{0}_{k \times 1}$ and any columns of $\mathbf{G}_{\text{LRC}}$ between the 11-th and 14-th are also a function of 5 other columns. For example, for Y_{11} observe that we have $\mathbf{g}_{11} = \left(\sum_{i=1}^{5} \mathbf{g}_i\right) + \left(\sum_{i=6}^{10} \mathbf{g}_i\right) + \mathbf{g}_{12} + \mathbf{g}_{13} + \mathbf{g}_{14}$, where $\left(\sum_{i=1}^{5} \mathbf{g}_i\right)$ is the first XOR parity and $\left(\sum_{i=6}^{10} \mathbf{g}_i\right)$ is the second and "$-$"s become "$+$"s due to the binary extended field. In the same manner as $\mathbf{g}_{11}$, all other columns can be repaired using 5 columns of $\mathbf{G}_{\text{LRC}}$. Hence all coded blocks have locality 5.

It should be clear that the distance of our code is at least equal to its $(14,10)$-RS precode, that is, $d \geq 5$. We prove that $d = 5$ is the maximum distance possible for a length 16 code has block locality 5. Let all codes of locality $r = 5$ and length $n = 16$ for $M = 10$. Then, there exist 6-groups associated with the n coded blocks of the code. Let, $Y_{\Gamma(i)}$ be the set of 6 coded blocks in the repair group of $i \in [16]$. Then, $H(Y_{\Gamma(i)}) \leq 5$, for all $i \in [16]$. Moreover, observe that due to the fact that $5 \not| 16$ there have to exist at least two distinct overlapping groups $Y_{\Gamma(i_1)}$ and $Y_{\Gamma(i_2)}$, $i_1, i_2 \in [16]$, such that $\left|Y_{\Gamma(i_1)} \cap Y_{\Gamma(i_2)}\right| \geq 1$. Hence, although the cardinality of $\left|Y_{\Gamma(i_1)} \cup Y_{\Gamma(i_2)}\right|$ is 11 its joint entropy is bounded as $H(Y_{\Gamma(i_1)}, Y_{\Gamma(i_2)}) = H(Y_{\mathcal{R}(i_1)}) + H(Y_{\mathcal{R}(i_2)}|Y_{\mathcal{R}(i_1)}) < 10$, i.e., at least one additional coded block has to be included to reach an aggregate entropy of $M = 10$. Therefore, any code of length $n = 16$ and locality 5 can have distance at most 5, i.e., $d = 5$ is optimal for the given locality. $\square$