Storage Codes: Managing Big Data with Small Overheads

Anwitaman Datta, Frédérique Oggier
Nanyang Technological University, Singapore
{anwitaman,frederique}@ntu.edu.sg
WWW: http://sands.sce.ntu.edu.sg/CodingForNetworkedStorage/

***Abstract*—Erasure coding provides a mechanism to store data redundantly for fault-tolerance in a cost-effective manner. Recently, there has been a renewed interest in designing new erasure coding techniques with different desirable properties, including good repairability and degraded read performance, or efficient redundancy generation processes. Very often, these novel techniques exploit the computational resources available 'in the network', i.e., leverage on storage units which are not passive entities supporting only read/write of data, but also can carry out some computations. This article accompanies an identically titled tutorial at the IEEE International Symposium on Network Coding (NetCod 2013), and portrays a big picture of some of the important processes within distributed storage systems, where erasure codes designed by explicitly taking into account the nuances of distributed storage systems can provide significant performance boosts.**

***Index Terms*—Erasure codes, Storage systems, Fault-tolerance, Repairability, Degraded reads, Data-insertion.**

I. Introduction

Erasure codes, originally invented for fault-tolerant communication over lossy channels, have long been adapted for storage centric applications and systems - starting from the CD technology. When a data object $\mathbf{o}$ is stored, it is first split into say k pieces $o_1, \ldots, o_k$, which are then mapped into n pieces $x_1, \ldots, x_n$, $n > k$. These n encoded pieces $x_1, \ldots, x_n$ are then stored across the storage device. When failures of the storage device occur, some of the x_i are lost. The erasure code then ensures that $\mathbf{o}$ can be recovered from the remaining x_i, similarly as in the case of a lossy channel, assuming enough x_i are left. Under many practical settings, erasure codes then provide desirable levels of resilience while incurring significantly lower storage overhead with respect to a naive strategy of replicating the data many-folds. In other words, there are erasure codes with a good erasure recovery capability and a higher rate than repetition codes.

In distributed storage systems, ranging from decentralized peer-to-peer systems to well administered data-centers, data is spread over multiple interconnected storage nodes: typically, every encoded piece x_i of an object $\mathbf{o}$ is stored in a distinct node. Many of the storage nodes are augmented with computing power. This provides an opportunity of exploiting these 'storage network' resources in order to achieve functionalities and performance boosts not achievable in systems where the storage nodes play a passive role. Erasure coding techniques were not originally designed to either leverage on the network resources, nor were they intended to suit the needs of distributed storage systems. Instead, they were designed for the recovery of the whole message (or data object), and so were the corresponding coding and decoding processes.

While the recovery of the data object in case of failures remains a crucial concern (that of fault-tolerance), many other properties are desirable, and have motivated the study of novel codes for distributed storage systems, among which (not necessarily by order of importance):

a) Repairability: When some nodes fail, the recovery ability of the code protects the stored data. However over time, more and more failures may occur, and data needs to be replenished to maintain the level of redundancy across the storage system. For example, if the data is simply replicated three times, once a first node fails, a repair consists of copying the data over a live node, so that three copies are again available. Note that the reason behind three copies is that once the first one is gone, it is too dangerous to have only one copy left, in case of another failure, there might not be enough time to safely copy the data. Repairability encompasses a variety of subproblems: how to detect a failure, when to trigger a repair, how many live nodes are needed to be contacted to perform a repair, how long does it take, how much bandwidth is needed, to name a few.

b) Degraded reads: Many failures in a storage system are transient, but these failures may still delay job processes, and actually form a key bottleneck. Suppose a job tries to access a block of data which is not available, it will have to wait. Alternatively, it may compute the data it wanted to read by accessing different live nodes, which is called a degraded read. Degraded reads are somehow treated as repairs, however with the following main differences: (1) they are repairs that do not need to be written on the storage medium, (2) they are referring to actual pieces of data, not encoded pieces of data.

c) Decentralized encoding: In a theoretical model, one just assumes that the storage system is keeping encoded pieces of data objects. However, the encoding needs to be actually computed. This may happen in two different scenarios. (1) One node might be in charge of computing the encoded pieces before injecting them into the network, bearing the computational burden of the encoding, and that of uploading

A. Datta's work was funded by A*STAR SERC TSRP Grant 102 158 0038. F. Oggier's work was funded by the Nanyang Technological University under Research Grant M58110049.

Fig. 1. Erasure coding techniques for different distributed storage system functions and processes: The data is inserted in the storage system. Pipelined insertion leads to the insertion of several replicas, which are in turn archived using a distributed encoding process, or encoded data can be directly inserted and computed using an in-network coding process. Once the data is erasure encoded, it needs to support repair, degraded reads, fault-tolerance, without too much cost in storage overhead.

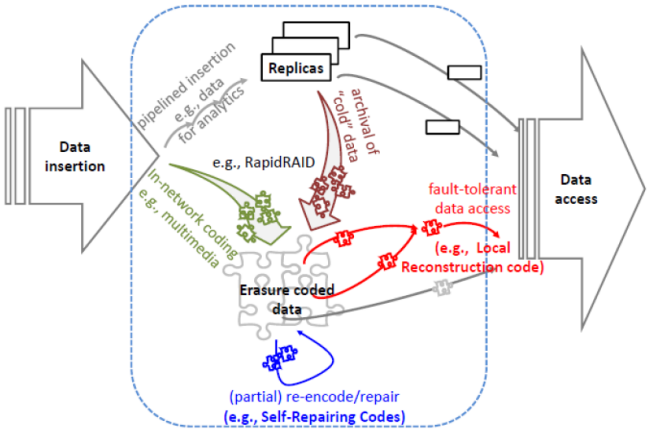
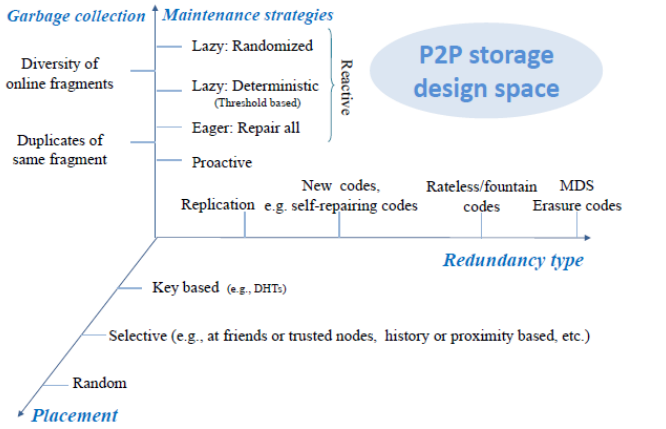


Fig. 2. Peer-to-Peer storage systems design choices: Many of the techniques developed in the context of P2P storage are also relevant in other environments, while some others are not. For instance, centrally administered data-centers exhibit significantly different workload and environment characteristics, and typical P2P issues such as user churn or trust relations, etc. are not relevant.



not only k pieces of data, but $n > k$ of them. (2) Most systems store replicas of the data as long as it is "hot", and manipulated, e.g. for analytics purposes. Once the data gets "cold", it is archived using erasure codes. Typically a single node carries out the encoding process per object. In both cases, decentralizing the encoding process by distributing it across several nodes may speed it up, and reduce the network cost.

The above properties all help boost the performance of a distributed storage system in different manners. Figure 1 provides a big picture of some of the core functionalities and processes within distributed storage systems, and how, many of these can benefit from novel coding techniques. While at the moment, there is arguably no single code instance which achieves all these properties simultaneously, in this article we will explore the individual novel codes in a piecemeal manner, which together lay the foundations for the search of codes that would satisfy all the desirable properties together.

Specifically, we will look at codes designed to achieve good repairability & degraded reads (see Section III), and codes designed to improve the throughput for populating the storage system with erasure coded data - either by converting already existing replicated data, or immediately at the time of data insertion (Section IV). While most of these works assume a trusted and collaborative environment, tolerance against Byzantine faults - where data is corrupted - either during the process of storage or repair, is also becoming an important topic of study, which we will briefly highlight.

Complementing the coding theoretical results, we also discuss some of the systems issues (in Section V), which give rise to the study of, and in turn, harness, the above mentioned novel code properties. Scalability as well as fault-tolerance needs may dictate different degrees of distribution - spanning across multiple storage nodes spread over a single data-center or multiple data-centers (both administered centrally by a single trusted entity), or even across multiple cloud service providers (thus involving multiple trusted entities administering the subparts) or over a peer-to-peer system (decentralized, with no trusted entity or global knowledge and control). We will look at some representative erasure coding based storage system designs which span the spectrum of distribution.

II. Erasure codes for distributed storage

The idea of adding some non-replication based redundancy, e.g. parity bits (adding one bit which is the sum of all/some data bits), has long been used in storage systems, prominently in RAID [29] (Redundant Arrays of Inexpensive Disks) systems. RAID systems are devised for creating a single logical unit comprising of multiple disks. The parity based redundancy employed, say in RAID-4, can be seen as a very simple erasure code. More recently, specifically in RAID-6, variations of Reed-Solomon erasure codes [33] are explicitly being adopted.

Erasure codes have also enjoyed a particularly high level of attention and usage in peer-to-peer (P2P) storage systems, especially in the OceanStore project [19], and many follow-up works since then [3], [14]. Because of high churn (membership dynamics of users going offline and coming back online), a critical issue with applying erasure codes in P2P systems is the problem of maintaining a desirable level of redundancy efficiently. Several approaches (which are not necessarily mutually exclusive) have been pursued in the literature, which roughly define the design-space of P2P storage systems, as illustrated in Figure 2. The prominent aspects include: how much redundancy to initialize the system with [3], what strategy to apply to replenish redundancy (reactive: eager or lazy [3], [5], versus proactive [7]), what strategy to carry out garbage collection in case unnecessary redundancy is created in the system [21], or how to choose the peers where the erasure coded data is stored (based on history of availability, trust, locality). These works typically address the problem by considering the underlying coding technique as a "black box", which takes k data symbols as input, outputs n symbols, which

have the property that any choice of k of them allows the data object to be recovered. This property, called the MDS property, is the best recovery capability that an erasure can enjoy.

An alternative approach to address the issue of P2P storage system maintainability is to rethink the underlying coding technique itself. Several independent initiatives, prominently on Regenerating codes [6], Pyramid codes [17] and Hierarchical codes [8] pioneered this line of work - that of designing new codes rather than assume a black box model. While Regenerating and Hierarchical codes focused on the repairability issue, Pyramid codes were aimed at degraded reads rather than repairability, but implicitly, the property could be harnessed for improved repairability as well.

In the meanwhile, erasure codes have transcended P2P and RAID storage systems, and have become an integral part of off-the-shelf solutions such as CleverSafe [4] and Pivot3 [30], and of major industry players, including the new version of Google file system (GFS) and Microsoft's Azure system [10][1]. Accordingly, the problem of designing codes for distributed storage systems is now addressed to suit the different nuances of various storage systems, ranging from RAID and P2P storage, to data centers.

III. Repairable erasure codes

The lack of ability to repair lost data efficiently is arguably one of the main reasons that slowed down the adoption of erasure codes in distributed storage systems. This is also probably why most of the focus in designing new codes has been on providing codes which enable repairs. Different families of codes have targeted different desirable features of the repair processes.

Regenerating codes [6] are a family of codes designed to reduce the bandwidth needed to repair a node failure. Roughly speaking, they consist of an erasure code (typically MDS) to store the data across different nodes, and of a network code, that enables repairs. A trade-off between the storage capacity of the nodes and the amount of repair bandwidth needed was established in [6], and regenerating codes achieve this trade-off. They bring improvements in terms of bandwidth needed to perform a repair with respect to a naive strategy that consists of recovering the data object first. They however require a large number of live nodes to be contacted per repair.

In contrast, the ability to carry out a repair using a low number of nodes necessitate a non-MDS code, but may offer several benefits including lower number of I/O operations, possibility to carry out multiple repairs in parallel, reduction in the network bandwidth usage (w.r. to the naive strategy), naturally enabling better degraded reads, etc. We will next discuss representative classes of codes which require a small number of live nodes to carry out a repair process.

A. Pyramid/local reconstruction codes

Take an erasure code that maps $\mathbf{o} = [o_1, \ldots, o_8]$ to $\mathbf{x} = [x_1, \ldots, x_{11}]$, where $\mathbf{x}$ is computed from $\mathbf{o}$ using an 8×11 (generator) matrix G, that is $\mathbf{x} = \mathbf{o}G$. The matrix G can be chosen so that

$$[x_1, \ldots, x_{11}] = [o_1, \ldots, o_8, c_1, c_2, c_3],$$

that is the first 8 columns of G form an identity matrix. Consider for example a Reed-Solomon code, which is an MDS code, and thus any choice of 8 coefficients from $\mathbf{x}$ allows to recover $\mathbf{o}$.

A Pyramid code can be built from this base code, by retaining the pieces $o_1, \ldots, o_8$, and two of the other pieces (without loss of generality, lets say, c_2, c_3). Additionally, split the data blocks into two groups $o_1, \ldots, o_4$ and $o_5, \ldots, o_8$, and compute some more redundancy coefficients for each of the two groups, which is done by picking a first symbol $c_{1,1}$ corresponding to c_1 by setting $o_5 = \ldots = o_8 = 0$ and $c_{1,2}$ corresponding to c_1 with $o_1 = \ldots = o_4 = 0$.

This results in an erasure code which maps an object of length 8 into 12 encoded pieces, looking like

$$[o_1, \ldots, o_8, c_{1,1}, c_{1,2}, c_2, c_3]$$

where $c_{1,1} + c_{1,2}$ is equal to the original code's c_1:

$$c_{1,1} + c_{1,2} = c_1.$$

This example illustrates the idea behind the design of Pyramid codes [17]. Redundancy is available in two forms: "local redundancy" $c_{1,1}$ and $c_{1,2}$ which can be used to repair a lost block without accessing blocks outside the subgroup (in this example, using 4 blocks for repair when only any one of $o_1, \ldots, o_4, c_{1,1}$ fails, instead of requiring 8 blocks), while if there are too many errors within a subgroup, then the "global redundancy" (c_2 and c_3 in this example) at that level may be used. In a multi-level Pyramid, one can move further up the Pyramid until repair is eventually completed. Use of local redundancy means that a small number of nodes is contacted, which translates into a smaller bandwidth footprint, as well as fewer disk I/O operations. Furthermore, if multiple isolated (in the hierarchy) failures occur, they can be repaired independently and in parallel. A new version of Pyramid codes, where the coefficients used in the encoding have been numerically optimized, namely Locally Reconstructable Codes [18] has more recently been proposed and is being used (primarily for better degraded reads) in the Azure [10] system.

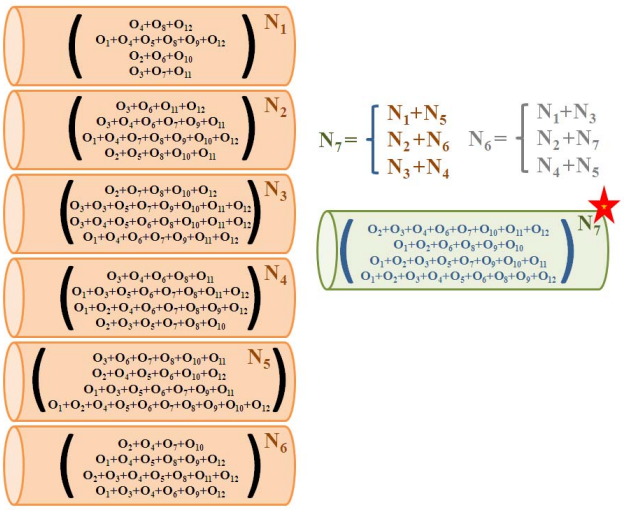
B. Locally/Self-Repairable codes

Pyramid codes provide locality to the 'data', but not so much for the 'parity' blocks. Specifically, if any of the global parity blocks are to be repaired, then the repair cost would be equivalent to the naive strategy of repairing the base (MDS) code used to create the Pyramid code. They are thus optimized for degraded reads, but not so much for repairs per say.

Self-repairing codes (SRC) [23] (example shown in Fig 3), to the best of our knowledge, were the first codes of length n designed to achieve $d = 2$ per repair for up to $\frac{n-1}{2}$ simultaneous failures, irrespective of which encoded blocks failed. Thus to say, all the encoded blocks are 'locally repairable'. The term 'locally repairable' is inspired from

[1]Many of these cloud-centric and data-center based systems nevertheless leverage on the design principles matured while developing P2P systems.

Fig. 3. An example of self-repairing codes from [23]: the object $\mathbf{o}$ has length 12, and is cut into $k = 3$ pieces, which are encoding as shown, resulting in a code stored across $n = 7$ nodes. If one of these nodes fail, say N_7, then it can be repaired in three independent manners, each using only two other live nodes. This code can repair any three simultaneous failures, where each repair process is carried out by contacting (distinct sets of) only two live nodes.



locally decodable and *locally correctable* codes, and was coined in a subsequent work, [28]. Other families of locally repairable codes based on projective geometric construction (Projective Self-repairing Codes) [22] and puncturing of Reed-Mueller codes [32], among many others have since been proposed. Note that achieving a very small repair degree has advantages in terms of repair time, parallelization, I/O accesses and bandwidth, however, it also affects other code parameters (such as its rate). Such trade-offs are yet to be fully understood, though some early works have recently been carried out [13].

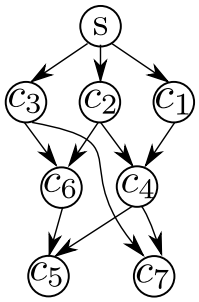
C. Security issues: Byzantine fault-tolerance

Since most of these novel repair approaches carry out the process in a decentralized manner with only a partial information, they are susceptible to poisoning attacks during the repair process, which may affect not only a specific repair operation, but also any future operations involving the affected block(s). Generalization of the network flow based analysis [6] of regenerating codes, taking into account such Byzantine faults during the repair operations was carried out in [24], and there has in the recent years been a growing interest (e.g., [15], [31]) in the security issues related to the repair process of erasure coded storage.

IV. Creating erasure coded redundancy

When data is stored using replication, the redundant data can be created using a pipelining process [12] so that no single node becomes a bottleneck: the source passes the data to a first storage node, which, in addition to locally keeping a copy, relays the same to a second node, and so on. There is no obvious analogous mechanism to create erasure coded redundancy in a distributed manner. However, the local/self-repairability property can be exploited not only to replace lost redundancy, but also to create the redundancy to start with by pursuing an 'in-network' decentralized encoding process.

Fig. 4. In-network redundancy generation for a locally (self) repairable code of length 7.



A. In-network encoding for fast data insertion

Let us consider the locally repairable code of Fig 3. In Fig 4 we show an in-network redundancy generation example for this code. The encoded pieces at the seven nodes are c_i for $i = 1...7$. Traditionally, a single node (the source) centrally computes all these encoded blocks and transfers them to seven different storage nodes, making the outgoing bandwidth at the central node a bottleneck. Alternatively, the source can upload a subset of this information (namely c_1, c_2, c_3 in this example). Then, nodes 1 and 2 send their respective fragments, c_1 and c_2, to node 4, which computes and stores $c_4 = c_1 + c_2$. The rest of the nodes compute the fragments $c_6 = c_2 + c_3$, $c_5 = c_4 + c_6$ and $c_7 = c_3 + c_4$ in a similar manner.

Note that the creation of c_5 and c_7 depends on the a priori existence of c_4 and c_6. Although at a first glance it might seem that symbols c_5 and c_7 can thus be created only some time in the future after the generation of c_4 and c_6, practical implementations can overcome this restriction by allowing nodes 4 and 6 to start forwarding the first generated bytes of c_4 and c_6 to nodes 5 and 7 in a pipelined way, similarly to the mechanism described above for replication. By doing so, blocks c_4 to c_7 can be generated quasi-simultaneously once nodes 1 to 3 receive their blocks from the source node.

This example demonstrates how local repairability can be exploited to generate the in-network redundancy generation, by identifying encoded block dependencies to generate a suitable flow of information as depicted in Figure 4. However, obtaining such a flow graph, scheduling the flow of information and orchestrating the decentralized coding accordingly for any arbitrary locally repairable code is not trivial. More details on these issues, as well as the resulting trade-offs such as higher usage of network resources for better data insertion throughput can be found in [25].

B. Rapid conversion of replicated data to erasure coded redundancy

Often, when data is first acquired/instered in a system, it is stored using replication, and subsequently it is archived using erasure coding when it becomes 'cold', i.e., it is accessed infrequently. Replication is preferred for the initial phase, since

Fig. 5. An example of migration of replicated data into an erasure coded archive (e.g. as done in HDFS-RAID [1]). The squares represent storage nodes and the arrows across boxes denote data (labels indicate the actual amount) transferred over the network. The "X" symbol denotes the replicas that are discarded once the archival finishes, and the symbol $\otimes$ denotes an encoding operation. We see that this process requires a total of *five* network transfers.

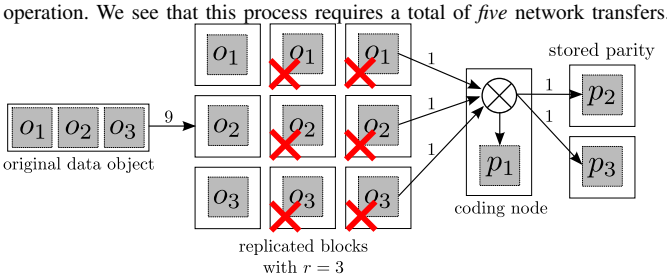
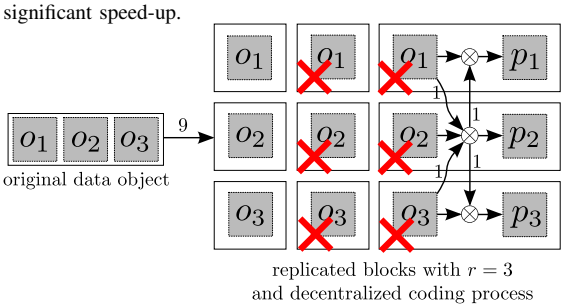


Fig. 6. Decentralized encoding: An example of migration of replicated data into an erasure coded archive (e.g. as done in RapidRAID [26]). The squares represent storage nodes and the arrows across boxes denote data (labels indicate the actual amount) transferred over the network. The "X" symbol denotes the replicas that are discarded once the archival finishes, and the symbol $\otimes$ denotes an encoding operation. We see that this process requires only *four* network transfers (instead of five used in the traditional approach depicted in Fig. 5). If each node has a upload/download bandwidth limit, then the decentralized coding process can, besides saving network traffic, also yield significant speed-up.



the replicas can then be accessed in parallel - yielding higher read throughput, or computational tasks utilizing the data can be assigned to the least loaded of the replicas, etc. Typically, the whole encoding process is treated as an atomic process, and is centrally carried out, as is the case, for instance in HDFS-RAID [1]. This means, one node first needs to collect all the relevant data blocks, and then carry out the coding. This is illustrated in Fig 5.

Given the presence of multiple replicas of the data, an interesting alternative is to decompose the hitherto atomic encoding process, and carry it out in a distributed fashion. This premise was explored recently, resulting in two new code families, namely RapidRAID [26], and a follow-up variant [27] where pieces of the original data are present among the encoded pieces. Fig. 6 provides a high-level sketch of the distributed coding process utilizing the existence of replicas.

V. Representative 'repairable' storage systems

While there was still some dilemma (at least in the public domain, though there were likely several works in progress in the industry) whether to use erasure codes in data-center environments [35], DiskReduce [11] was one of the pioneering works which integrated erasure coding with the popular open-source Hadoop Distributed File System (HDFS), which was followed by a mature version implemented by Facebook, namely HDFS-RAID [1]. HDFS-RAID essentially uses a hybrid strategy for achieving redundancy, namely replication and erasure coding, and furthermore, it supports two kinds of erasure coding - namely XOR based parity and Reed-Solomon coding, which provide one or arbitrary number of parities, respectively. Depending on how actively the data is being read, the degree of replication is reduced, as erasure coded redundancy is introduced in the system to achieve comparable fault-tolerance but as significantly lower storage overhead. In the meanwhile, several other major players including Google and Microsoft have also integrated erasure coding in their proprietary file/storage systems. Other commercial data-storage centric applications and services using erasure codes include products from CleverSafe, or Pivot3. HDFS-RAID separates the data access from repair process. The former is carried out in a fault tolerant manner by carrying out a decoding process if any data block is unavailable or corrupted. The latter is carried out periodically by a background daemon process, or by manual triggers from the system administrator. The repair process requires to carry out decoding/re-encoding.

Barring Microsoft's Azure system [10], all the other systems mentioned above use traditional erasure code, and the focus is only on achieving fault-tolerance while incurring low storage overhead. As discussed previously, Azure deploys an optimized variation of Pyramid codes, named Local Reconstruction Code [18], which allows reconstruction of unavailable data blocks using a small number of other data and parity blocks, providing better degraded read performance with respect to what could be achieved if MDS codes were to be used, where a complete decoding becomes necessary even if a single data block is missing. Though not specifically meant for repairs, the reconstruction property of Local Reconstruction Codes can be exploited to achieve better repairs in some fault-scenarios involving data blocks. A very recent work, XORbas [34] essentially does so. Another recent work explores the use of a collaborative regenerating code to provide recovery of multiple failures [20] in HDFS.

Though this article primarily explores novel erasure codes, it is worth noting that engineering alternatives to the repair problem using traditional codes is also possible. An example is our HDFS-RAID compliant implementation called CORE (Cross-object redundancy based) storage system [9], where the idea (as shown in Fig 7) is to use HDFS-RAID's Reed-Solomon coding to create multiple parities for individual data objects - which primarily provides redundancy for fault-tolerance, while using HDFS-RAID's XOR parity to create another parity block for (both data and parity) blocks from different objects in order to achieve very good repairability even in presence of multiple failures.[2]

While there has been a flurry of systems research (though a lot of ground still remains to be covered), particularly many of them integrating the novel codes with HDFS, and

[2][20] has incidentally an identical name and addresses the same problem of multiple failures repair, but the approach is fundamentally different, and inherits both the advantages and the drawbacks of regenerating codes.

Fig. 7. In this example of cross-object coding, a horizontal (9,6) Reed-Solomon code is used per object, and XOR parity is computed on data as well as parity blocks across three objects.

aimed at data-center environments, there have also been a few prototypes which look at a higher level of abstraction, namely distributing data across multiple cloud services. DepSky [2] was an early system which, analogous to the data-center related works (e.g. DiskReduce and HDFS-RAID), uses traditional erasure codes, while NCCloud [16] has incorporated regenerating coding for repairs in such multi-cloud settings.

VI. Concluding remarks

Codes achieving low degree repairs naturally realize reconstruction of data blocks, which can be leveraged for good degraded reads, and to a certain extent the converse is also true. In that sense, these two functionalities can be simultaneously achieved by many of the novel codes - particularly, local/self-repairing codes (but not regenerating codes) and cross-object coding. Likewise, the in-network coding technique to create erasure coded data on the fly when data is being introduced in the system leveraged on the local repairability property.

Thus, in order to realize a system achieving all the processes outlined in Fig 1, the prominent outstanding issue is a code which realizes good repairs and degraded reads, but simultaneously achieves RapidRAID like mechanism to benefit from the existence of replicas in order to distribute and accelerate the encoding process.

A code with all these desirable properties is the foremost open issue in our opinion. For applicability in wide range of environments (e.g. decentralized and P2P settings), codes with inherent security properties would also be desirable.

Additionally, even if only a small fraction of the stored data likely needs to be manipulated and mutated, mechanisms to support efficient updates to mutable content is another interesting aspect to be further explored. Likewise, depending on the kind of data and applications, e.g., multimedia data, marrying ideas from multi-resolution coding with those of repairable codes poses another obvious yet interesting frontier.

References

[1] Apache.org. HDFS-RAID. http://wiki.apache.org/hadoop/HDFS-RAID.

[2] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. DepSky: Dependable and secure storage in a cloud-of-clouds. In *EuroSys*, 2011.

[3] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. M. Voelker. Total recall: System support for automated availability management. In *NSDI*, 2004.

[4] Cleversafe. www.cleversafe.com/.

[5] A. Datta and K. Aberer. Internet-scale storage systems under churn – a study of the steady-state using markov models. In *P2P*, 2006.

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

[7] A. Duminuco, E. Biersack, and T. En-Najjary. Proactive replication in distributed storage systems using machine availability estimation. In *CoNEXT*, 2007.

[8] A. Duminuco and E. W. Biersack. Hierarchical codes: How to make erasure codes attractive for peer-to-peer storage systems. In *P2P*, 2008.

[9] K.S. Esmaili, L. Pamies-Juarez, and A. Datta. The core storage primitive: Cross-object redundancy for efficient data repair & access in erasure coded storage. In *arXiv:1302.5192*, 2013.

[10] B. Calder et al. Windows azure storage: A highly available cloud storage service with strong consistency. In *SOSP*, 2011.

[11] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson. Diskreduce: Raid for data-intensive scalable computing. In *PDSW*, 2009.

[12] S. Ghemawat, H. Gobioff, and S.T. Leung. The google file system. In *SOSP*, 2003.

[13] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin. On the locality of codewords symbols. In *ECCC*, 2011.

[14] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: highly durable, decentralized storage despite massive correlated failures. In *NSDI*, 2005.

[15] Y. S. Han, R. Zheng, and W. Mow. Exact regenerating codes for byzantine fault tolerance in distributed storage. In *INFOCOM*, 2012.

[16] Y. Hu, H. Chen, P. Lee, and Y. Tang. NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds. In *FAST*, 2012.

[17] C. Huang, M. Chen, and J. Li. Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems. *Trans. Storage*, 9(1), 2013.

[18] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in windows azure storage. In *USENIX ATC*, 2012.

[19] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *ASPLOS*, 2000.

[20] R. Li, J. Lin, and P. Lee. Core: Augmenting regenerating-coding-based recovery for single and concurrent failures in distributed storage systems. In *29th IEEE Conference on Massive Data Storage*, 2013.

[21] X. Liu and A. Datta. Redundancy maintenance and garbage collection strategies in peer-to-peer storage systems. In *SSS*, 2009.

[22] F. Oggier and A. Datta. Self-repairing codes for distributed storage – a projective geometric construction. In *ITW*, 2011.

[23] F. Oggier and A. Datta. Self-repairing homomorphic codes for distributed storage systems. In *Infocom*, 2011.

[24] F. E. Oggier and A. Datta. Byzantine fault tolerance of regenerating codes. In *P2P*, 2011.

[25] L. Pamies-Juarez, A. Datta, and F. Oggier. In-network redundancy generation for opportunistic speedup of backup. *Future Generation Computer Systems*, 29, 2013.

[26] L. Pamies-Juarez, A. Datta, and F. Oggier. RapidRAID: Pipelined Erasure Codes for Fast Data Archival in Distributed Storage Systems. *Infocom*, 2013.

[27] L. Pamies-Juarez, F. Oggier, and A. Datta. Decentralized erasure coding for efficient data archival in distributed storage systems. In *ICDCN*, 2013.

[28] D.S. Papailiopoulos and A.G. Dimakis. Locally repairable codes. In *ISIT*, 2012.

[29] D. A. Patterson, G. Gibson, and R.H. Katz. A case for redundant arrays of inexpensive disks (raid). *SIGMOD Rec.*, 17(3), 1988.

[30] Pivot3. Pivot3: Converged storage and compute appliances. http://pivot3.com/.

[31] A.S. Rawat, O. O. Koyluoglu, N. Silberstein, and S. Vishwanath. Optimal locally repairable and secure codes for distributed storage systems. *arXiv/1210.6954*, 2012.

[32] A.S. Rawat and S.Vishwanath. On locality in distributed storage systems. In *ITW*, 2012.

[33] I.S. Reed and G. Solomon. Polynomial codes over certain finite fields. *SIAM*, 8(2), 1960.

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

[35] Z. Zhang, A. Deshpande, X. Ma, E. Thereska, and D. Narayanan. Does erasure coding have a role to play in my data center? Technical Report MSR-TR-2010-52, Microsoft Research, 2010.