Redundantly grouped cross-object coding for repairable storage

Datta, Anwitaman; Oggier, Frederique

2012

Datta, A., & Oggier, F. (2012). Redundantly grouped cross-object coding for repairable storage. Proceedings of the Asia-Pacific Workshop on Systems - APSYS '12, 1-6.

https://hdl.handle.net/10356/95628

https://doi.org/10.1145/2349896.2349898

Redundantly Grouped Cross-object Coding for Repairable Storage

Anwitaman Datta, Frédérique Oggier
Nanyang Technological University, Singapore
E-mail: {anwitaman, frederique}@ntu.edu.sg

Abstract

The problem of replenishing redundancy in erasure code based fault-tolerant storage has received a great deal of attention recently, leading to the design of several new coding techniques [3], aiming at a better repairability. In this paper, we adopt a different point of view, by proposing to code across different already encoded objects to alleviate the repair problem. We show that the addition of parity pieces - the simplest form of coding - significantly boosts repairability without sacrificing fault-tolerance for equivalent storage overhead. The simplicity of our approach as well as its reliance on time-tested techniques makes it readily deployable.
Keywords: storage, fault-tolerance, repair

1 Introduction

Given the scale of data-centers in terms of the number of machines and network components, failures, might they be transient or permanent, are inevitable. In order to deal with transient failures, adequate redundancy is introduced to keep services available, while permanent failures need to be handled through redundancy replenishment over time. Erasure codes have been increasingly embraced as the preferred redundancy mechanism, because of their low storage overhead and high fault tolerance. The shift from replication to erasure coding has been prompted by the continuously increasing amount of data that data-centers need to store, together with the associated infrastructural and operational costs.

While erasure codes have long been studied for their fault tolerance, the problem of restoration of lost redundancy for long term resilience has gained much attention recently [4, 7, 9, 14, 10]. The notion of better repairability encompasses several aspects, most prominently: the amount of data transfer and I/O per repair, number of nodes involved for a repair and the time to carry out a repair. Individual existing works often addresses only a subset of these issues - and the presented work is no exception in that respect. However, instead of addressing the question of designing erasure codes with better inherent repair capabilities, this work explores additional coding across already erasure encoded objects. This results in disentangling high fault-tolerance realized through erasure coding from efficient repairability achieved by another layer of coding.

More precisely, our proposal is to carry out erasure coding on individual data objects being stored, as is currently the case, and then create parity groups, formed by including one encoded piece from each object, together with a parity piece - the simplest form of coding. This is similar to a RAID-4 type of coding on multiple encoded objects (Fig. 1). This very simple strategy provides performance benefits in terms

of repairability, while still retaining comparable fault-tolerance as could be achieved with the use of only erasure coding and incurring equivalent storage overhead. We explore the implications of our design along the various performance criterion enumerated above.

2 Background

Erasure codes have long been studied in the peer-to-peer (P2P) system literature as a way to achieve low overhead fault-tolerant storage [1, 4, 13], and have more recently been also embraced in data-centers such as Microsoft Azure [2], Hadoop File System[1] and the new version of Google File System. The essential idea of erasure coding is to split an object O of size $|O|$ in k pieces each of size $|O|/k$, and transform these into $n > k$ encoded pieces. One can reconstruct the original object using $k' \geq k$ of the encoded pieces, where k' is a code parameter. In the following, the notation (n, k)-code will imply the optimal choice $k' = k$, with *rate* defined as k/n. A popular example of such optimal codes are the Reed-Solomon codes [12].

In the context of distributed storage systems, n different encoded pieces for an object O encoded with an (n, k)-code are stored in n different nodes, with *storage overhead* $\rho = n/k$. For instance, a $(15, 10)$-code will have a storage overhead of $\rho = 1.5$, which is also referred to as a 1.5x code. Let f be the failure probability of a storage node (unless stated otherwise, we will assume that each storage node fails independently of the others). The chance $P_{obj}(n, k, f)$ that the stored object survives is then

$$P_{obj}(n, k, f) = \sum_{i=k}^{n} \binom{n}{i} (1-f)^i f^{n-i}. \quad (1)$$

Systems designers often use the notion of numbers of nines, denoted by π, to describe the data availability. If $P_{obj}(n, k, f) \geq 0.9999$, then at least four 9s of availability is guaranteed, which can be calculated by

$$\pi = -lg(1 - P_{obj}(n, k, f)). \quad (2)$$

While erasure codes provide very good fault-tolerance, they have been criticized for poor performance in terms of repairability [13]: in order to create a single encoded piece, data equivalent to the size of the object is needed, though deferring repairs is known to reduce the per-repair cost [1]. This is particularly relevant in P2P settings, which (1) start with a relatively high amount of redundancy to deal with the network dynamics, and (2) experience temporary churn, i.e., nodes frequently go offline but come back online, thus repair is sometimes not even necessary. If r repairs are carried out together, a node downloads k available encoded pieces to recreate the r missing encoded pieces, and then redistribute them among $r - 1$ other nodes, incurring a total traffic (per repair) of

$$\frac{k + r - 1}{r}. \quad (3)$$

The case of data-center environments is different. Deferring repairs is not a feasible option: given that the number of faults is relatively small (nevertheless non-negligeable) due to the use of a dedicated infrastructure and of the stability of the environment, as well as cost considerations, the initial amount of redundancy is quite low. While procrastinating repairs is not preferred, the system still needs to be able to recover from multiple (correlated) failures.

As opposed to the heuristic of deferring repairs to amortize repair costs, there has also been a significant interest recently in designing new codes with better repair properties. These include efforts to use traditional erasure codes in multiple layers such as Pyramid [7] and Hierarchical codes [4], applying network coding on top of erasure codes [9, 14] or sacrificing the MDS property to introduce dependencies among the encoded pieces [10] (see [3] for a more detailed exploration of such existing works). These approaches try to address the repair problem by considering the encoded

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

pieces of individual objects in isolation. In contrast, we next propose and study a RAID-4 like additional layer of coding among the encoded pieces of multiple objects. DiskReduce [5] also applies RAID for Hadoop File System, but in that work RAID is utilized analogous to the use of erasure codes on individual objects.

3 Parity over encoded objects

Consider m objects $O_1, \ldots, O_m$ to be stored. For $j = 1, \ldots, m$, object O_j is erasure encoded into n encoded pieces $e_{j1}, \ldots, e_{jn}$, all of them to be stored in $m \cdot n$ distinct storage nodes. Additionally, we create *parity groups*, formed by m encoded pieces (with one encoded piece chosen from each of the m objects), together with a parity piece (or xor). W.l.o.g, a parity group is of the form $e_{1l}, \ldots, e_{ml}$ for $l = 1, \ldots, n$, and the parity piece p_l is $p_l = e_{1l} \oplus \ldots \oplus e_{ml}$. The parity pieces are then stored in other n distinct storage nodes. Such additional redundancy is akin to RAID-4. The process is depicted in Fig. 1. While the storage nodes may be placed arbitrarily within the data-center, allocating one parity group per rack as displayed has advantages in terms of repair localization, discussed in Section 3.3.

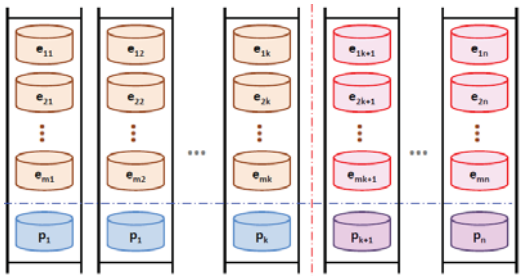

Figure 1: Redundantly grouped coding

The above proposal is the simplest form of coding that can be performed across several objects. We now discuss how it performs with respect to repairability and fault-tolerance.

3.1 Repairability

In the event of a single node failure affecting e_{jl}, repair consists of xoring the remaining e_{kl} ($k \neq j$) and p_l. The data transfer cost is m, as opposed to k. A suitable choice of $m < k$ can then achieve significant savings in terms of **data transfer overhead**, which is furthermore disentangled from the code parameter k, allowing more flexibility in choosing larger values of k. This may be desirable for various reasons such as faster data access in parallel from many nodes, since a larger k yields lower amount of data per encoded piece stored at individual nodes, whose I/O may be a bottleneck.

The **repair time** is decreased by pipelining the data-flow through the live nodes, which xor their local data with the incoming one on the fly, and forward the result to the next node. In contrast, with erasure coding, the node performing the repair needs to receive k encoded pieces and re-encode. The repair is thus bottlenecked by the bandwidth, I/O and computation capacities of that node.

The probability that a repair can be performed within a parity group is $(1 - f)^m$. Parity groups being mutually exclusive, failures occurring in different groups do not affect each other. Unlike in P2P systems, the typical chance of random failures f is very low in data-center environments[2], and our design correspondingly disentangles the long term fault-tolerance achieved via erasure coding, from the short-term maintenance process, which predominantly relies on parity.

In case the parity cannot be exploited (for instance, due to two or more failures within a group), repair is handled through the erasure code, by gathering k encoded pieces. Ignoring mixed strategies where the last repair within one group is done using the parity, and assuming repair can actually be handled (as determined by (1)), the expected data (no. of pieces) transfer per repair is approximated by

$$\Delta = m \cdot (1 - f)^m + k \cdot (1 - (1 - f)^m). \quad (4)$$

Fig. 2 shows Δ (measured by the size of one encoded piece) for one repair using our strat-

[2]Correlated failures are discussed in Section 3.3.

egy with a $\rho = 1.5$x $(15, 10)$-code, for various choices of m. The baseline is when no parity is used, that is 10 pieces need to be transferred.

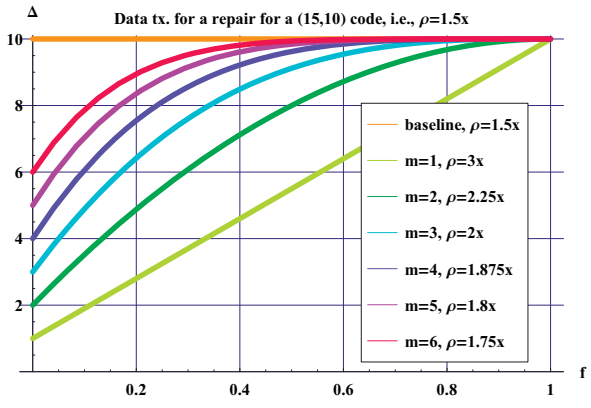

Figure 2: Data transfer for one repair

A small m provides a better chance that the parity can be used for repair, and reduces the repair cost, though it also means an increase by a factor of $1 + 1/m$ (see Section 3.4 for the derivation) in the effective storage overhead ρ.

3.2 Parity group size tradeoffs

We saw above that only $m < k$ gives data transfer savings. Consider the optimistic scenario where r encoded pieces of the same object have to be repaired, but there are no other failures in the respective r parity groups (large number of pieces of one object missing is a general indicator of massive failures, where likely other pieces from the groups have also failed). Repairs then require xoring the available data with the parity piece, amounting to the transfer of $m \cdot r$ pieces. However, r pieces of the same object can also be repaired using the (n, k) code, by transferring $k + r - 1$ pieces. Thus meaningful choices of m should satisfy $m \cdot r \leq k + r - 1$. For a $(15, 10)$ code, if $r = 3$, then $m \leq 4$.

On the other hand, $m = 1$ means replicating/mirroring each encoded piece (as in RAID-1). Thus, for very low values of f, repairs are performed by recreating a new mirror, though at the price of doubling the original storage overhead $\rho = n/k$. For a $(15, 10)$ code, this means 3x. More prudent choices of m are 3 or 4, where ρ becomes 2x and 1.875x respectively, while the savings in data transfers (see Fig. 2) are 40 to 50% even when f is as large as 0.1. Lower values of f imply more savings.

3.3 Potential repair localization

If the data placement is one parity group per rack (as in Fig. 1), the data transfer does not incur any bandwidth within the interconnect, and the pipelined repair time is slightly larger than essentially the I/O time to read or write one encoded piece. Also, even if the whole rack fails or get disconnected from the rest of the data-center - leading to a 'correlated failure' of all the nodes within the rack, for each object, only one encoded piece becomes unavailable. In contrast, if similar localizations of repair traffic were achieved by confining multiple encoded pieces of the same object in the same rack, then all these pieces would simultaneously become unavailable, affecting data availability.

3.4 Storage overhead vs. resilience

While creating the RAID-4 like parity on the encoded pieces of different objects improves repairability, it obviously also incurs more storage overhead than just the erasure code. It is thus necessary to quantify and compare the resilience of the proposed strategy, with respect to erasure codes utilizing 'equivalent' storage.

A particular encoded piece is not available with probability $f \cdot (1 - (1 - f)^m)$, taking into account that it could be calculated from the xor of the remaining pieces of the parity group instead. Thus, the probability $P_{RonEC}(n, k, f, m)$ of data availability while using RAID-like parity on m erasure coded objects (RonEC) is

$$P_{RonEC}(n, k, f, m) = P_{obj}(n, k, f \cdot (1-(1-f)^m)) \quad (5)$$

where $P_{obj}(n, k, f)$ is as defined in (1).

Consider as data placement a parity group per rack, and assume that a rack fails with probability f_r, while individual node failures

happen independently with probability f_s,[3] then the object availability probability becomes

$$\begin{aligned} & P_{RonEC_{Rack}}(n,k,f_s,f_r,m) \qquad (6) \\ = \; & P_{obj}(n,k,f_r+(1-f_r)f_s(1-(1-f_s)^m)). \end{aligned}$$

One interpretation of the proposed scheme is that $m \cdot k$ pieces are stored in $(m+1) \cdot n$ encoded pieces, for a storage overhead of $\frac{(m+1)n}{mk}$. We should compare the proposed strategy to an (n_{eff}, k) code with effective value $n_{\text{eff}} = \frac{m+1}{m}n$ corresponding to the same overhead. However, n_{eff} may not be an integer (e.g., for $n = 15$ and $m = 2$, $n_{\text{eff}} = 22.5$). For a fair comparison, we thus consider that some objects are stored with an $n_{\text{eff}'} = \lfloor \frac{m+1}{m}n \rfloor$ while others are stored with $n_{\text{eff}''} = \lceil \frac{m+1}{m}n \rceil$, where the fractions are chosen such that the weighted average storage overhead is equivalent. In fact, this is achieved when the two n-values are used in a $1 : \zeta$ mix, where $n_{\text{eff}'} + \zeta n_{\text{eff}''} = n_{\text{eff}}$, i.e., $\zeta = \frac{n_{\text{eff}} - n_{\text{eff}'}}{n_{\text{eff}''}}$.

The expected object availability is then given by the weighted average of data availabilities:

$$\begin{aligned} & P_{EC-mix}(n,k,f,m) \\ = \; & \frac{P_{obj}(n_{\text{eff}'},k,f) + \zeta P_{obj}(n_{\text{eff}''},k,f)}{1+\zeta}. \qquad (7) \end{aligned}$$

Fig. 3 shows the number of 9s of availability π comparing our strategy to the fault-tolerance that would be achieved via erasure codes only, but incurring equivalent storage overhead. Not surprisingly, when $m = 1$ (replication), an erasure code using such extra storage has significantly better fault tolerance. But for $m = 2, 3, 4$ we notice that our strategy enjoys comparable fault-tolerance, and even marginally outperforms for very small values of f.

3.5 Summary

Let us summarize the practical advantages of coding across m erasure encoded objects using

[3] Elsewhere, since we do not assume a parity group per rack, we do not distinguish rack from node failure, and $f = f_s + f_r$ represents the cumulative effect.

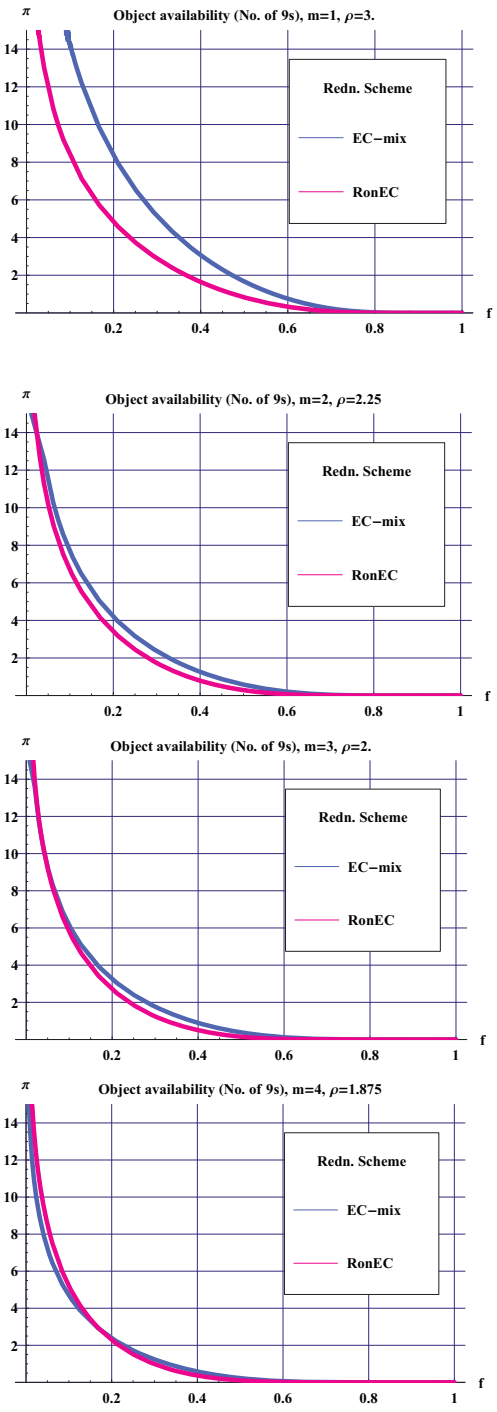

Figure 3: Number of 9s object availability π (y-axis) corresponding to parameters $n = 15$ & $k = 10$. The x-axis shows the probability f that individual storage nodes fail. Determined using (5) & (7) [and also (2)] for our proposal ***RonEC*** RAID-4 on top of erasure coding and ***EC-Mix*** respectively.

a parity piece: a suitable choice of m (i) lowers the cost of data transfer for repair, and (ii) reduces the expected number of storage nodes that need to be contacted during repairs. Furthermore, the possibility to localize repair traffic within a rack (iii) avoids the use of the datacenter's precious bandwidth, (iv) without introducing correlated failure among encoded pieces of the same object. While not quantitatively explored, (v) pipelining data through the live nodes of a parity group decreases the repair time, since the repairing node's I/O or bandwidth does not become a bottleneck. These nice repairability properties are achieved while (vi) retaining comparable fault-tolerance for equivalent storage overhead if only erasure coding per object were to be used.

4 Conclusions

We study the problem of replenishing lost redundancy efficiently in erasure coding based storage systems, and advocate the use of two kinds of redundancy - one primarily achieving fault-tolerance by erasure coding individual objects, the other achieving cheap repairs by creating RAID-4 like parity of the erasure encoded pieces from different objects. Our approach is very simple, and based on mature techniques (standard erasure coding and RAID-4), and hence easy to apply in practice. While the current study is based on simple analytical models, more rigorous experiments based on traces of fault-patterns observed in practice (such as [8]), as well as the implications of the redundancy model and placement in terms of data update, deletion & corruption are outstanding issues to be further studied.

Acknowledgement

A. Datta's work was partially supported by NTU/MoE Tier-1 grant number RG 29/09. F. Oggier's work was supported by the Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03. The authors will like to thank Michael Militzer and the anonymous reviewers for their constructive criticisms of this work, particularly identifying the problems of object deletion or updates and soft-errors (such as flipped bits) that the current paper does not investigate, but are issues that need to be studied comprehensively to determine the practicality of the proposed scheme.

References

[1] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, G. M. Voelker, "TotalRecall: System Support for Automated Availability Management", *NSDI* 2004.

[2] B. Calder, et al., "Windows Azure Storage: a highly available cloud storage service with strong consistency", *ACM SOSP*, 2011.

[3] A. Datta, F. Oggier, "An Overview of Codes Tailor-made for Networked Distributed Data Storage", arXiv 2011.

[4] A. Duminuco, E. Biersack, "Hierarchical Codes: How to Make Erasure Codes Attractive for Peer-to-Peer Storage Systems", *P2P* 2008.

[5] B. Fan, W. Tantisiriroj, L. Xiao, G. Gibson, "DiskReduce: RAID for Data-Intensive Scalable Computing", *PDSW* 2009.

[6] S. Ghemawat, H. Gobioff, S-T. Leung, "The Google file system", *ACM SOSP* 2003.

[7] C. Huang, M. Chen, and J. Li, "Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems", *IEEE NCA* 2007.

[8] W. Jiang, C. Hu, and Y. Zhou and A. Kanevsky, "Are Disks the Dominant Contributor for Storage Failures? A Comprehensive Study of Storage Subsystem Failure Characteristics", *ACM Transactions on Storage (TOS)*, Vol. 4, Issue 3, 2008.

[9] A.-M. Kermarrec, N. Le Scouarnec, G. Straub, "Repairing Multiple Failures with Coordinated and Adaptive Regenerating Codes", *NetCod* 2011.

[10] F. Oggier, A. Datta, "Self-repairing Homomorphic Codes for Distributed Storage Systems", Infocom 2011. Extended version at `http://arxiv.org/abs/1107.3129`.

[11] D. A. Patterson, G. Gibson, R. H. Katz "A case for redundant arrays of inexpensive disks (RAID)" *ACM SIGMOD*, 1988.

[12] I. S. Reed, G. Solomon, "Polynomial Codes Over Certain Finite Fields", *Jrnl. of the Society for Industrial and Appl. Maths.*, no 2, vol 8, SIAM, 1960.

[13] R. Rodrigues and B. Liskov, "High Availability in DHTs: Erasure Coding vs. Replication", *IPTPS* 2005.

[14] K. W. Shum, "Cooperative Regenerating Codes for Distributed Storage Systems", *ICC* 2011.