HAL
open science

Regenerating Codes: A System Perspective

Steve Jiekak, Anne-Marie Kermarrec, Nicolas Le Scouarnec, Gilles Straub, Alexandre van Kempen

To cite this version:

Steve Jiekak, Anne-Marie Kermarrec, Nicolas Le Scouarnec, Gilles Straub, Alexandre van Kempen. Regenerating Codes: A System Perspective. Dependability Issues in Cloud Computing (DISCCO 2012), Oct 2012, Irvine, California, United States. pp.436-441, 10.1109/SRDS.2012.58 . hal-00764262

HAL Id: hal-00764262

https://hal.science/hal-00764262v1

Submitted on 12 Dec 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Regenerating Codes: A System Perspective

Steve Jiekak*†, Anne-Marie Kermarrec‡, Nicolas Le Scouarnec*, Gilles Straub* and Alexandre Van Kempen*
*Technicolor, Rennes, France
†EPFL, Lausanne, Suisse
‡INRIA Rennes - Bretagne Atlantique, France

arXiv:1204.5028v2 [cs.DC] 16 Jul 2012

***Abstract*—The explosion of the amount of data stored in cloud systems calls for more efficient paradigms for redundancy. While replication is widely used to ensure data availability, erasure correcting codes provide a much better trade-off between storage and availability. Regenerating codes are good candidates for they also offer low repair costs in term of network bandwidth. While they have been proven optimal, they are difficult to understand and parameterize. In this paper we provide an analysis of regenerating codes for practitioners to grasp the various trade-offs. More specifically we make two contributions: *(i)* we study the impact of the parameters by conducting an analysis at the level of the system, rather than at the level of a single device; *(ii)* we compare the computational costs of various implementations of codes and highlight the most efficient ones. Our goal is to provide system designers with concrete information to help them choose the best parameters and design for regenerating codes.**

I. Introduction

As cloud-based solutions for backup and sharing are being offered to users, the amount of storage needed for cloud services keeps increasing. In order to lower the costs (*e.g.*, hardware, energy) for operating such systems, it is important to rely on efficient paradigms. Currently, many systems still rely on well-proven replication [1] to provide high availability from non-reliable devices. While easy to understand and implement, replication is far from being optimal with respect to the trade-off between storage and availability [2]–[4]. Arguably, erasure correcting codes can significantly lower the amount of storage needed in data-centers. However, with classical erasure codes (*e.g.*, Reed-Solomon), repairing after a device failure generates many I/Os and requires transferring a large amount of information over the network. In large-scale multi-site data-centers, the background network traffic due to repairs can become prohibitive for large amounts of data stored. In this paper, we focus on an attractive alternative, namely regenerating codes [5], to lower such network costs.

Regenerating codes offer the same properties as erasure correcting codes with respect to storage and availability. Yet, as opposed to erasure correcting codes, regenerating codes significantly lower the network traffic upon repairs. The seminal paper [5] on regenerating codes applies network coding to storage systems and defines the optimal trade-off between the amounts of data stored and transferred. Regenerating codes, designed to be as generic as possible, rely on many parameters, difficult to grasp in practice where device availability vary from a system to another; let alone the fact that many variants of regenerating codes exist (*e.g.*, [6]–[10]).

In order to help choose the right parameters and coding scheme, we make the following contributions:

- We study the influence of the various parameters at the system level, depending on storage device availability. We show that the optimum at device level does not always apply at system level. (Section III)
- We compare the computational costs of various coding schemes for regenerating codes (random codes [6], product-matrix codes [8], and exact linear codes [7]) to the costs of classical erasure correcting codes (Reed-Solomon codes). (Section IV)

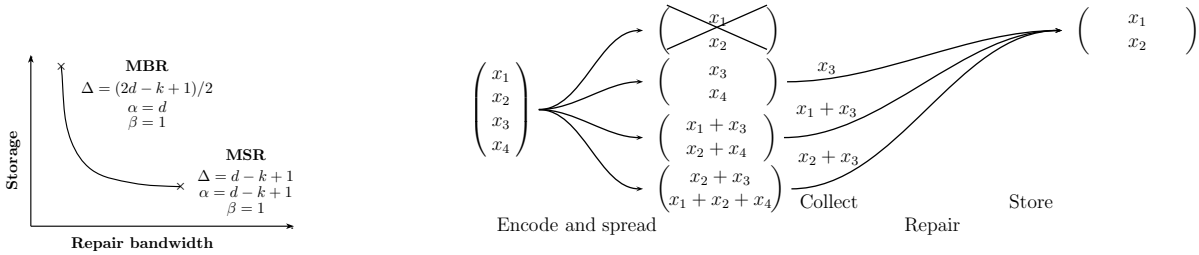
Previous practical work on regenerating codes focused either only on random codes [11] while we consider several other codes; or on a specific system with a specific code [12], [13] while we study several codes and give conclusions that can be applied broadly.

II. Model and Background

We consider a system of n devices connected by a network. The system stores files of size $\mathcal{M}$ that are immutable (*i.e.*, data is appended to the system and once written cannot be modified, as in [14]). Devices are available with a probability p due to temporary disconnections (*e.g.,* reboot, software upgrade, short-term network disruption). Devices fail permanently with a rate of λ failures per month per device (*e.g.*, disk crash, device replacement, long-term network disruption).

When using erasure correcting codes, the file is divided into k blocks and n encoded blocks are produced so that any k encoded blocks allow recovering the file: the file remains available as long as at least k devices are available. Hence, the resulting system availability is $A = \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{n-i}$. Whenever a block is permanently lost, a repair mechanism is used to regenerate it. The repair procedure in erasure correcting codes consists in contacting k live devices, recovering the file and encoding it again to produce a new block. Since the whole data is read from disks and transferred over the network, this procedure has both high I/O costs, which can be reduced using specific codes [15], and high network cost, which can be reduced using regenerating codes [5]. In this paper, we focus on the latter codes reducing network costs.

Regenerating codes apply network coding to storage systems to offer the best trade-off between network bandwidth repair cost γ and storage cost α. The file is divided into $k\Delta$ sub-blocks. These $k\Delta$ original sub-blocks are encoded into $n\alpha$ encoded sub-blocks which are then spread on the n devices



(a) Tradeoff curve

(b) General scheme (MSR, $k = 2$, $n = 4$, $d = 3$, $\alpha = \Delta = 2$)

Figure 1. Regenerating codes are optimal with respect to storage and repair cost thanks to efficient repair methods.

(*i.e.*, one group of α sub-blocks is stored on each device) so that contacting any k devices allows recovering the file. Regenerating codes rely on an additional parameter d, which is the number of devices involved in a repair. Whenever a failure occurs, if the number of available devices is at least d, the following optimal repair method, shown in Figure 1b, can be used: *(i)* the device being repaired fetches β sub-blocks[1] from each of d available devices (thus $\gamma = d\beta$), *(ii)* the device stores α sub-blocks computed from the $d\beta$ sub-blocks it received.

Regenerating codes, as explained in [5], can be parameterized by the values Δ and α to minimize either the storage (MSR, Minimum Storage Regenerating) or the bandwidth (*i.e.*, network repair cost) (MBR, Minimum Bandwidth). This trade-off is illustrated on Figure 1a.

The seminal paper about regenerating codes relies on randomized code constructions, namely random linear network codes [6]. With such codes, the repaired data is not strictly equal to the lost data. As a first consequence, such codes cannot be maintained in a systematic form. Systematic codes suppresses decoding costs and allows direct access to small parts of the file since the file can be read directly from the k first blocks (or $k\alpha$ first sub-blocks) without decoding. As a second consequence, checking the integrity of such randomized codes requires to use complex techniques. In order to solve these two issues, it has been proposed to rely on exact regenerating codes [16]. Various code constructions have been proposed with two of the most advanced ones being by Suh *et al.* [7] and by Rashmi *et al.* [8]. In Section IV, we will compare the computational costs of these schemes with the randomized code constructions [6] and regular Reed-Solomon erasure correcting codes.

Finally, in regenerating codes [5], the number d of devices to contact during repairs is chosen once for all and cannot be changed. Adaptive regenerating codes [17] relax this constraint and allow d to adapt to each repair. However, adaptive regenerating codes currently can only be implemented with random linear network codes, which have a high complexity and require costly schemes for integrity checking. In the following section, adaptive regenerating codes are included for they show the best achievable theoretical bound, yet further research is needed before they can be used in practice. To repair "static" regenerating codes when less than d devices are available, k available devices must be chosen and the repair must be carried by decoding the file before encoding it again leading to a cost $\gamma' = k\Delta$ as with regular erasure correcting codes. Hence, finding the right value for d is important to avoid using this expensive repair by decoding method.

In the sequel of the paper, we study regenerating codes and focus on MSR codes, which offer the same trade-off between storage and availability as Reed-Solomon codes.

[1]In the rest of the paper, we will focus on scalar codes ($\beta = 1$) for the sake of clarity and for they have lower computational costs. Vector codes ($\beta > 1$) have higher computational costs but are useful for reducing I/Os.

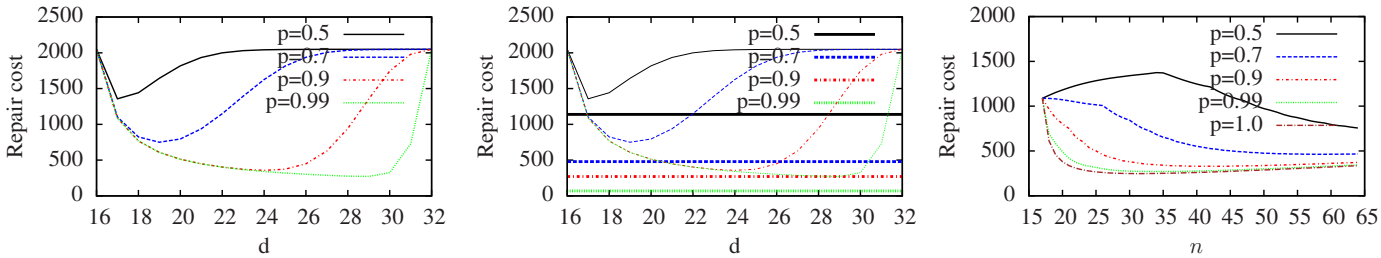
III. System level analysis

In this section, we consider several levels of device availability and study regenerating codes by performing an analysis at the system level for it matches the real costs observed. At the device level, the storage cost is α and the repair cost (*i.e.*, network bandwidth) is γ. Hence, at the system level, the storage cost is $n\alpha$ and the repair cost is $\Gamma = n\lambda\gamma$. The cost at the device level is known for decreasing as n increases. However, this conclusion does not apply at the system level. This section shows some interesting interactions between parameters. In the settings shown on the plots in the rest of the paper, we consider one file of size $\mathcal{M} = 64\text{MB}$ and a failure rate of $\lambda = 1$ failure per month. The repair cost is given in MB per month per file stored. This cost scales linearly with the number of files, the file size $\mathcal{M}$, and the failure rate λ (*i.e.*, if 10 files are stored or if $\lambda = 10$ failures per month, the cost is 10 times higher) thus allowing extrapolating results.

A. How many devices to repair from?

Let us consider that k and n are chosen to reach a given system availability [2]–[4]. Regenerating codes require choosing an additional parameter d, which is the number of live devices contacted during a repair. Theoretical papers suggest that the best value for d is $d_{\text{opt}} = n - 1$. However, it turns out that this choice is not the best as soon as the device availability is $p < 1$ as we explain in this section.

Let us define the probability that exactly i devices are available as $P(X = i) = \binom{n}{i} p^i (1-p)^{n-i}$. We define $G = g(X)$ as the cost of the repair when X devices are available. If $i \geq d$, we repair using the optimal method (*i.e.*, $g(i) = \frac{\mathcal{M}}{k}\frac{d}{\Delta}$), otherwise we repair by decoding the whole file (*i.e.*, $g(i) = \mathcal{M}$). The expected cost at the system level is $E(G) = \sum_{i=k}^{n-1} P(X = i) g(i)$. When less than k devices are



(a) $k = 16$, $n = 32$ (MSR) (b) $k = 16, n = 32$ (ARC in bold, MSR in thin) (c) $k = 16$

Figure 2. System level repair cost as a function of d and n. The repair cost admits a minimum that is not $d_{\text{opt}} = n - 1$.

available[2], the repair is simply delayed. To this end, we plot the system repair cost as $n\lambda \frac{E(G)}{P(k \leq X \leq n-1)}$.

According to theoretical studies [5], a high value for d helps reducing the cost of repairs at the device level. Yet, this also increases the probability that less than d devices are available thus leading to more frequent repairs by decoding. Hence, there should be an optimal value for d. For always available devices ($p = 1$), it appears that $d_{\text{opt}} = n - 1$, as stated in [5].

On Figure 2a, we consider a system relying on an MSR code with $n = 32$ and $k = 16$. We consider various device availabilities p and plot the system level cost as a function of d. We observe that the cost function admits an optimal value for d. For low to medium availabilities ($p = 0.5$ to $p = 0.9$), the optimal value for d is rather low (much lower than $n - 1$ value suggested by the literature [5]). For high availability $p = 0.99$, the optimal value is close to $n - 1$ but is still $d_{\text{opt}} = 29$ ($d_{\text{opt}} = n - 3$). Moreover, choosing $d = 31$ instead of $d = 29$ when $p = 0.99$ more than doubles the repair cost. Hence, as soon as devices are not highly available ($p = 1$), the designer must choose d according to the device availability observed in the system to best leverage regenerating codes.

The repair cost for erasure correcting codes is the same as the cost for regenerating codes with $d = k = 16$. For high device availability ($p = 0.99$), regenerating codes with $d = d_{\text{opt}}$ offer a 10 time improvement over erasure correcting codes. For medium availability ($p = 0.7$), with relatively low $d_{\text{opt}} = 19$, regenerating codes still offer a 3 time improvement over erasure correcting codes.

As explained, the system is rather sensitive to the choice of value d. This calls for codes where d can be changed on the fly, namely adaptive regenerating codes [17], which are similar to MSR codes. These codes may seem more practical since they can self-adapt to the system, yet they currently lack practical code designs.

Figure 2b plots repair costs for optimal adaptive regenerating codes (ARC) in bold lines alongside regular codes in thin lines[3]. An interesting observation is that adaptive regenerating codes, initially designed for highly dynamic systems, perform particularly well in rather stable systems. Indeed, when compared to MSR with optimal d, they provide an improvement of 80% in rather stable systems ($p = 0.99$), whereas in highly dynamic systems ($p = 0.5$) the improvement is only 10%.

B. How to choose the redundancy level?

As with classical codes, the amount of redundancy must be chosen so that the resulting availability $P(X \geq k) = \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{n-i}$ is at least the desired availability A. This guides the values of the parameters k and n as studied in [2]–[4]. Yet, with regenerating codes, it might be interesting to sacrifice storage efficiency in favor of repair cost by either increasing n or relying on MBR codes instead of MSR codes. The first approach increases the number of devices redundant data is spread on, while the latter increases the amount of redundant data on each device without changing n. We study MSR codes and set n to get the lowest possible repair cost without constraints on the system availability A.

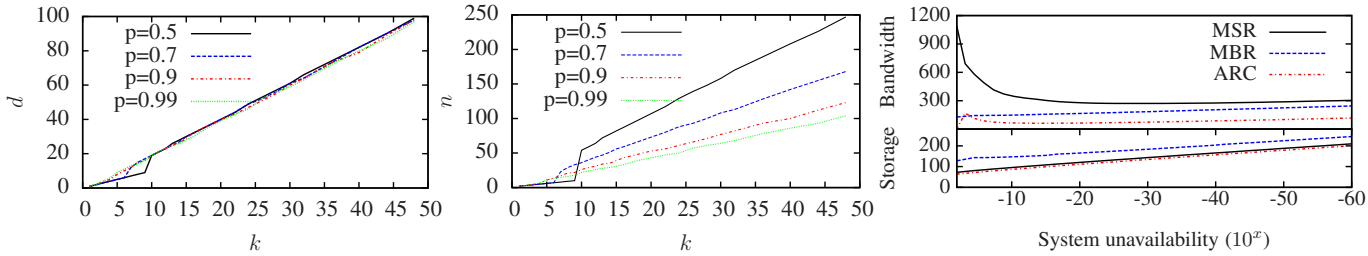
Let us consider that $p \approx 1$ and hence $d = n-1$. As initially observed by Dimakis *et al.* [5], very large n, thus allowing very large $d = n - 1$ does not necessarily helps in reducing costs. Indeed, the system level repair cost $n\lambda\gamma$ admits a minimum at $n_{\text{opt}} = k + \sqrt{k^2 - k}$. When k is large enough, $n_{\text{opt}} \approx 2k$.

We perform a similar study when $p < 1$. For each possible value n, we choose the corresponding value d that offers the lowest cost. This effect is shown on Figure 2c. When applying this procedure for multiple values of p and k, we observe that $d_{\text{opt}} \approx 2k$ and $n_{\text{opt}} \approx k \times c(p)$ where $c(p)$ is a constant that depends only on p as shown on Figure 3a. n_{opt} is chosen so that on average, the number of devices available for repair is approximately d_{opt}. These curves also show that for low values (p, k), regenerating codes cannot operate efficiently for any value (n, d) and erasure-correcting codes ($n = k + 1, d = k$) offer the lowest repair cost (but also a low reliability) for such (p, k). However, slightly increasing k is sufficient to leverage regenerating codes in spite of low device availabilities p.

We now study MBR as an alternative to MSR. Indeed, increasing n beyond the value needed to ensure the required availability for the file, consists in globally increasing the redundancy above the minimal level. Using MBR also increases the redundancy but avoids increasing n (and hence the system failure rate $n\lambda$). To this end, we compare storage and repair bandwidth of MSR, MBR and Adaptive codes (ARC) for various system unavailabilities (Fig. 3b) given a device availability of $p = 0.99$. Overall, MBR consumes twice the storage space needed for MSR or ARC. MBR

[2]This case remains rare as the system availability $P(X \geq k)$ is high.

[3]When using adaptive regenerating codes [17], repairs are performed independently to compare to MSR and MBR on a fair basis (*i.e.*, no coordinated multiple repairs).



(a) optimal (n, d) (b) $p = 0.99, k = 16$

Figure 3. The lowest repair cost is for $d_{\text{opt}} \approx 2k$ and $n_{\text{opt}} \approx k \times c(p)$ (3a). Optimal storage and bandwidth (3b).

significantly reduce the bandwidth consumption. The system designer should choose MBR or MSR depending on resources she wishes to save. ARC[4] approach and even outperform MBR with respect to bandwidth without sacrificing the storage efficiency (as efficient as MSR) thus significantly improving high availability systems. Yet, currently, no practical code designs for ARC are known. This represents a challenging research agenda for theoretical work.

IV. Computational Performance

Various code designs exist to implement regenerating codes. These codes rely on various algorithms and data structures, leading to different costs when implemented. In this section, we consider both CPU processing and memory costs. We compare the following most significant codes, which are Reed-Solomon codes for erasure correcting codes, and random linear network codes [6], exact linear codes [7] and product-matrix codes [8] for MSR regenerating codes.

A. Memory costs

We briefly study the costs associated with the data structures needed to be loaded in memory for encoding and decoding.

Reed-Solomon codes are linear codes which rely on an $n \times k$ matrix containing elements of q bits. For $k = 16, n = 32, q = 16$, the resulting encoding matrice is of size 1 Kbytes.

Linear network codes (random linear network codes [6] or exact linear codes [7]) rely on an $k\alpha \times n\Delta$ (approximately $k^2 \times nk$ for MSR codes). For $k = 16, n = 32, q = 16$, the resulting encoding matrice is of size 256 Kbytes.

The product-matrix codes [8] rely on a more compact scheme and the encoding matrix is of size $n \times 2\alpha$ (approximately $n \times 2k$ for MSR codes). For $k = 16, n = 32, q = 16$, the resulting encoding matrice is only 2 Kbytes.

Overall, when considering 64 MB data blocks (typical in cloud storage systems), the memory requirements for these matrices is negligible. Moreover, apart for random linear network codes, which are non-deterministic codes, the encoding matrices, which are the same for all files, are created using a deterministic process and need not be stored since they can be re-created on the fly when needed. As a consequence, memory costs, even if higher for regenerating codes than for erasure correcting codes, are only a minor issue and are not relevant for choosing one particular code design over another. However, as we will explain hereafter, the processing costs are a true limitation and vary greatly from one code design to another.

B. CPU costs

To compare the processing costs, we implemented in Java several MSR codes. All codes implementations have similar levels of optimization and rely on a log-table based finite field implementation[5]. We ran these mono-threaded implementations on a Pentium E2200.

We implemented *(i)* random linear network codes (RL) [6], *(ii)* exact linear codes (EL) [7][6], *(iii)* product-matrix codes that use a compact representation of codes with efficient encoding and decoding algorithms (PM) [8], and *(iv)* Reed-Solomon erasure correcting codes (RS) [18].

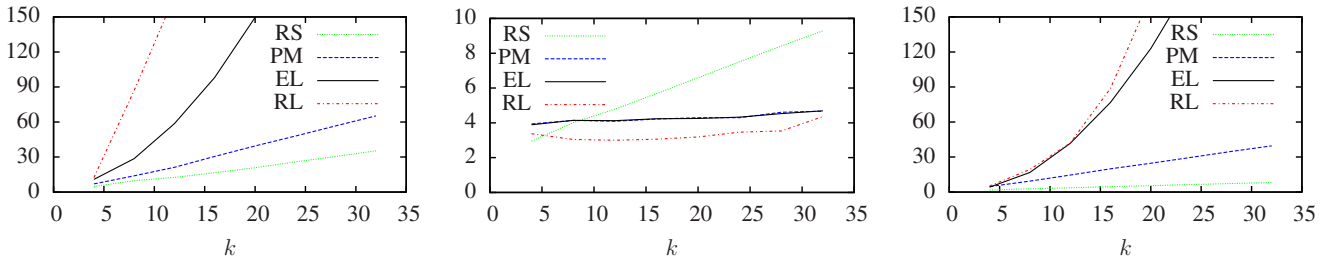
Regular erasure correcting codes (*e.g.*, Reed-Solomon) involve linear operations on matrices of size $k \times k$. Such operations have a reasonable complexity $\Omega(k^2)$. However, regenerating codes involve the same linear operations but on matrices of size $k\alpha \times n\Delta$ (approximately $2k^2 \times k^2$ when $n = 2k$ and $d = n-1$). Consequently, naive linear implementations (EL, RL) [6], [7], [11] suffer from a complexity of $\Omega(k^4)$ and have high computational costs even for low values of k. It can be observed that Product-Matrix codes [8] (PM) rely on efficient algorithms departing from classical linear approaches thus lowering costs as shown here.

Figure 4 shows the time needed to process a file of size $\mathcal{M} = 16\text{MB}$ depending on the parameter k. When considering the encoding (Figure 4a), all regenerating codes (PM, EL, RL) perform worse than regular erasure correcting codes (RS). However, it is interesting to notice that product-matrix codes (PM) clearly outperform regular linear regenerating codes codes (EL, RL). The two linear regenerating codes rely on the same encoding and decoding algorithm, yet, exact linear codes (EL) clearly outperform random linear network codes (RL). Indeed, the encoding matrix of exact linear codes (EL) is much sparser than the one of random linear network codes (RL). Overall, these results are consistent with the asymptotic complexities discussed in the previous paragraph.

[4]Again, for the comparison to be fair, repairs are performed independently without relying on coordinated multiple repairs capability of ARC, which would reduce bandwidth consumption even more.

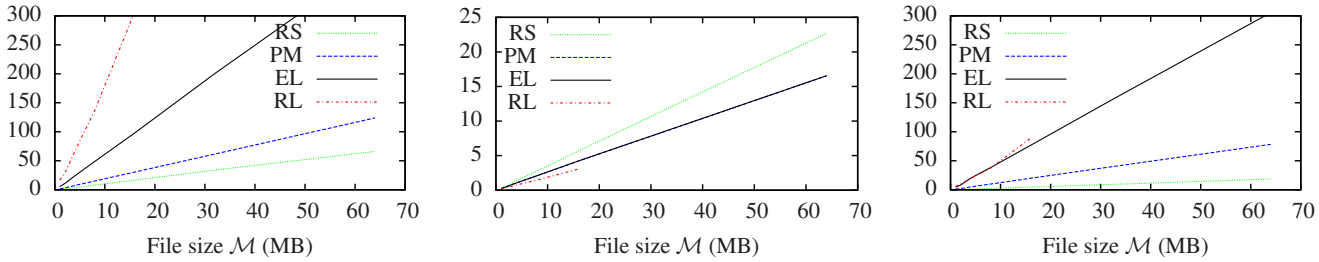
[5]Computational costs using multiplication-table based finite field implementations are 30% lower but with such finite field implementation k is limited because of constraints on field size. These plots, as well as plots showing that all computational costs scale linearly with the file size, were omitted due to limited space.

[6]We use a linear implementation of product-matrix codes [8], leading to constructions similar to [7].



(a) Time for encoding a file in seconds (b) Time for repairing a lost device in seconds (c) Time for decoding a file in seconds

Figure 4. Performance for $\mathcal{M} = 16\text{MB}$. Reed-Solomon decode the file and encode the lost block at each repair.



(a) Time for encoding a file in seconds (b) Time for repairing a lost device in seconds (c) Time for decoding a file in seconds

Figure 5. Performance for $k = 16$. Reed-Solomon decode the file and encode the lost block at each repair.

When considering the repair time (Figure 4b), all regenerating codes (PM, EL and RL) exhibit similar costs, with a slight advantage for random linear network codes (RL) because their randomized repair procedure is simpler. Reed-Solomon codes, whose repair rely on a costly decoding followed by an encoding suffer a cost that increase with k and that is higher than the cost of all regenerating codes schemes. As a consequence, when repair are frequent, it can be more interesting from a computational point of view, to use regenerating codes even if encoding and decoding are more costly.

When considering the decoding time, Reed-Solomon (RS) codes have a low cost thanks to a specific decoding algorithm and small encoding matrices. Product-matrix codes (PM) have a reasonable cost that is much lower than the other regenerating codes (EL, RL). Indeed, product-matrix codes use a specific decoding algorithm much more efficient than the algorithm used for other regenerating codes (EL, RL).

Figure 5 shows that the encoding time, the repair time and the decoding time scale linearly with the file size $\mathcal{M}$, and confirms the relatively good performance of product matrix codes (PM), and the poor performance of linear codes (EL, RL) when compared to classical erasure correcting codes (RS).

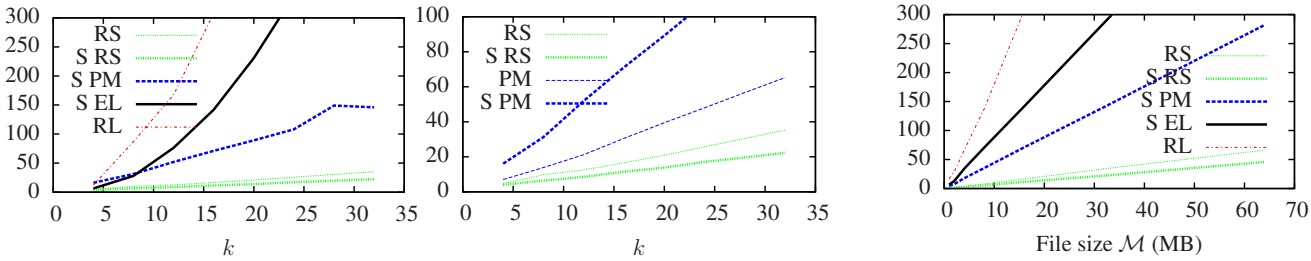
Figure 6 studies the additional cost of relying on systematic codes. Systematic codes are interesting because the k first devices store non-encoded data. As a result, when performing a read on such codes, no decoding is needed (*i.e.*, the decoding cost is null). Reed-Solomon codes use a specific encoding matrix that encodes directly to a systematic form. As this matrix is sparser than the matrix for the non-systematic version, the systematic Reed-Solomon (S RS) is faster than the non-systematic version (RS). On the contrary exact linear regenerating codes (S EL) use a matrix that is costly to create and that is denser than the original one leading to higher costs. Finally, product matrix codes rely on a pre-coding step that is performed before the encoding step. This precoding step uses an algorithm similar to the decoding algorithm and as such increases the costs: the systematic product matrix codes (S PM) are more costly than the non-systematic ones (PM).

Hence, product-matrix codes are good candidates for replacing Reed-Solomon codes in practical systems. Their impact on memory and CPU remains limited. As shown on Figure 5b, their non-systematic form only doubles the encoding costs and quadruples the decoding costs when compared to non-systematic Reed-Solomon codes. Their systematic form multiplies by 7 the encoding costs when compared to systematic Reed-Solomon codes. Systematic product-matrix codes should be preferred when data is read more frequently than it is written, otherwise non-systematic codes are more efficient.

For MBR codes, which are not the focus of this paper, Fractional Repetition Codes [10], not implemented in this benchmark, perform very well since they rely on an efficient systematic pre-code (*e.g.* Reed-Solomon) to produce encoded sub-blocks that are then replicated on several devices. Repairs and reads are performed using simple transfers without any computation. The two other implementations of MBR codes are random linear network codes based [6] or product-matrix codes based [8], and since they use the same algorithms at the MBR and the MSR point, they will behave similarly to their MSR implementation (PM and RL).

V. Conclusion and Discussion

We study the impact of various parameters of regenerating codes since they can have significant impact at the system scale. First, despite common belief, d_{opt} is not necessarily $d_{\text{opt}} = n - 1$ and instead should be carefully tuned due to its high impact on the repair method efficiency at the



(a) Time for encoding a file in seconds ($\mathcal{M} = 16$MB) (all systematic codes on the left, comparison of the best codes with their non systematic version on the right)

(b) Time for encoding a file in seconds ($k = 16$)

Figure 6. Performance for systematic codes. The repair procedure remains unchanged (see Figure 4). No decoding is needed when accessing the file.

system level. Even for high device availability $p = 0.99$ where $d = n - 1$ seems reasonable, we get a two time improvement by choosing $d = n - 3$. Second, the lowest repair cost for MSR codes is obtained with $d = 2k$ and $n = k \times c(p)$ with $c(p)$ a constant depending on the device availability. Third, if sufficient system availability is achieved with a rather low n, the designer needing to further lower repair cost should favor MBR over using MSR with artificially increased n because MBR allows achieving lower repair cost at the system scale. Finally, since adaptive regenerating codes theoretically outperform both MSR and MBR at the system level when looking at the amount of data transferred; providing exact code designs for ARC can be a promising theoretical research area that would allow implementing ARC in practical systems.

We also study the computational costs associated with the various coding schemes available. We show that the additional cost of using regenerating codes (product-matrix codes [8]) is reasonable. We have shown that non-systematic product-matrix codes outperform other linear code designs and only double (resp. quadruple) the cost of encoding (resp. decoding) when compared to Reed-Solomon codes. Systematic product-matrix codes multiply by seven the cost of encoding when compared to systematic Reed-Solomon codes. Hence product-matrix codes keep computational costs within reasonable values given the savings in term of network bandwidth they allow to achieve. As a perspective, to lower systematic codes computational costs, it would be interesting to design product-matrix like codes that encode directly to a systematic form without requiring a pre-processing step.

Interesting perspectives and ongoing theoretical work for applying regenerating codes to practical system concern the minimization of I/Os jointly with network repair cost and storage cost. Indeed, we focused on regular regenerating codes minimizing the network cost. In some systems, minimizing the I/O (disk reads) is equally important. Regenerating codes are not incompatible with this consideration and this research subject is active. Recent work has optimized I/O cost by relying either on specific coding schemes [9], [10] or considering variations of regenerating codes [19], [20]. Also, reducing I/O is an interesting application for coordinated regenerating codes [17] that support repairing multiple devices at once. Indeed, if instead of performing t successive repairs, t repairs are slightly delayed and performed at once in a coordinated way, the I/O for repairs are factored thus reducing the overall I/O by a factor t without jeopardizing the optimality with respect to network.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *SOSP*, 2003.

[2] H. Weatherspoon and J. Kubiatowicz, "Erasure Coding Vs. Replication: A Quantitative Comparison," in *IPTPS*, 2002.

[3] W. K. Lin, D. M. Chiu, and Y. B. Lee, "Erasure Code Replication Revisited," in *P2P*, 2004.

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

[5] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. O. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems," *IEEE Transactions On Information Theory*, 2010.

[6] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," *IEEE Transaction on Information Theory*, vol. 52, pp. 4413–4430, 2006.

[7] C. Suh and K. Ramchandran, "Exact-Repair MDS code construction using interference alignment," *IEEE Transactions On Information Theory*, 2011.

[8] 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," *IEEE Transaction on Information Theory*, 2011.

[9] V. R. Cadambe, S. A. Jafar, C. Huang, and J. Li, "Optimal Repair of MDS Codes in Distributed Storage via Subspace Interference Alignement," in *ISIT*, 2011.

[10] S. El Rouayheb and K. Ramchandran, "Fractional Repetition Codes for Repair in Distributed Storage Systems," in *Allerton Conference*, 2010.

[11] A. Duminuco and E. Biersack, "A Pratical Study of Regenerating Codes for Peer-to-Peer Backup Systems," in *ICDCS*, 2009.

[12] Y. Hu, C.-M. Yu, Y. K. Li, P. P. C. Lee, and J. C. S. Lui, "NCFS: On the Practicality and Extensibility of a Network-Coding-Based Distributed File System," in *NetCod*, 2011.

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

[14] B. Calder *et al.*, "Windows Azure storage: A highly available cloud storage service with strong consistency," in *SOSP*, 2011.

[15] 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 *FAST*, 2012.

[16] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A Survey on Network Codes for Distributed Storage," *The Proceedings of the IEEE*, vol. 99, pp. 476–489, 2010.

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

[18] S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the SIAM*, 1960.

[19] A. Kiani and S. Akhlagi, "Selective Regenerating Codes," *IEEE Communications Letters*, vol. 15, pp. 854–856, 2011.

[20] D. S. Papailiopoulos, J. Luo, A. G. Dimakis, C. Huang, and J. Li, "Simple Regenerating Codes: Network Coding for Cloud Storage," in *INFOCOM*, 2012.