Simple Regenerating Codes: Network Coding for Cloud Storage

Dimitris S. Papailiopoulos†, Jianqiang Luo‡, Alexandros G. Dimakis†, Cheng Huang*, and Jin Li*
†University of Southern California, Los Angeles, CA 90089, Email:{papailio, dimakis}@usc.edu
‡ EMC Data Domain[1], Email: jianqiang@wayne.edu
*Microsoft Research, Redmond, WA 98052, Email: {cheng.huang, jinl}@microsoft.com

***Abstract*—Network codes designed specifically for distributed storage systems have the potential to provide dramatically higher storage efficiency for the same availability. One main challenge in the design of such codes is the exact repair problem: if a node storing encoded information fails, in order to maintain the same level of reliability we need to create encoded information at a new node. One of the main open problems in this emerging area has been the design of simple coding schemes that allow exact and low cost repair of failed nodes and have high data rates. In particular, all prior known explicit constructions have data rates bounded by $1/2$.**

In this paper we introduce the first family of distributed storage codes that have simple look-up repair and can achieve rates up to $2/3$. Our constructions are very simple to implement and perform exact repair by simple XORing of packets. We experimentally evaluate the proposed codes in a realistic cloud storage simulator and show significant benefits in both performance and reliability compared to replication and standard Reed-Solomon codes.

I. Introduction

Distributed storage systems have reached such a massive scale that recovery from failures is now part of regular operation rather than a rare exception [19]. Large scale deployments typically need to tolerate multiple failures, both for high availability and to prevent data loss. Erasure coded storage achieves high failure tolerance without requiring a large number of replicas that increase the storage cost [7]. Three application contexts where erasure coding techniques are being currently deployed or under investigation are Cloud storage systems, archival storage, and peer-to-peer storage systems like Cleversafe and Wuala (see e.g., [1], [3])

One central problem in coded distributed storage systems is that of maintaining an encoded representation when failures occur. To maintain the same redundancy when a storage node leaves the system, a *newcomer* node has to join the array, access some existing nodes, and exactly reproduce the contents of the departed node. Repairing a node failure in an erasure coded system requires in-network combinations of coded packets, a concept called network coding, which has been investigated for numerous other applications (see e.g., [4], [5]).

[1]This work was performed while the third author was with Microsoft Research, Redmond, and Wayne State University.

This work was supported, in part, by NSF Career Grant 1055099 and a Research Gift by Microsoft Research.

In this paper we focus on network coding techniques for exact repair of a node failure in an erasure coded storage system [1], [2]. There are several metrics that can be optimized during repair: the total information read from existing disks during repair [9], the total information communicated in the network [11]–[18] (called repair bandwidth [2]), or the total number of disks required for each repair [6], [10].

Currently, the most well-understood metric is that of repair bandwidth. For designing (n, k) erasure codes that have n storage nodes and can tolerate any $n - k$ failures, an information theoretic tradeoff between the repair bandwidth γ and the storage per node α was established in [2], using cut-set bounds on an information flow graph and various constructions exist for the the two extreme points on this tradeoff [11]–[18]. However, different performance metrics might be of interest in different applications. It seems that for cloud storage applications the main performance bottleneck is the disk I/O overhead for repair, which is proportional to the number of nodes d involved in rebuilding a failed node. Despite the substantial amount of prior work, there are no practical code constructions of efficiently repairable codes with data rates above $1/2$.

Our Contribution: We introduce the first family of distributed storage codes that have simple look-up repair and can achieve rates above $1/2$. Our constructions are simple to implement and perform exact repair by simple packet combinations. Specifically, the (n, k)-SRC is a code for n storage nodes that can tolerate $n - k$ erasures and has rate $\frac{2}{3} \cdot \frac{k}{n}$, which can be made arbitrarily close to $\frac{2}{3}$, for fixed erasure resiliency. To repair a single node we need to access $d = 4$ other disks and the repair bandwidth cost is 2 times the size of the lost contents.

We experimentally evaluate the proposed codes in a realistic cloud storage simulator that models node rebuilds in Hadoop [20]. Our simulator was initially validated on a real system of 16 machines connected by a 1GB/s network. Our subsequent experiment involves 100 machines and compares the performance of SRC to replication and standard Reed-Solomon codes. We find that SRCs add a new attractive point in the design space of redundancy mechanisms for cloud storage.

II. Simple Regenerating Codes

The first requirement from our storage code is the (n, k) property: a code will be storing information in n storage nodes

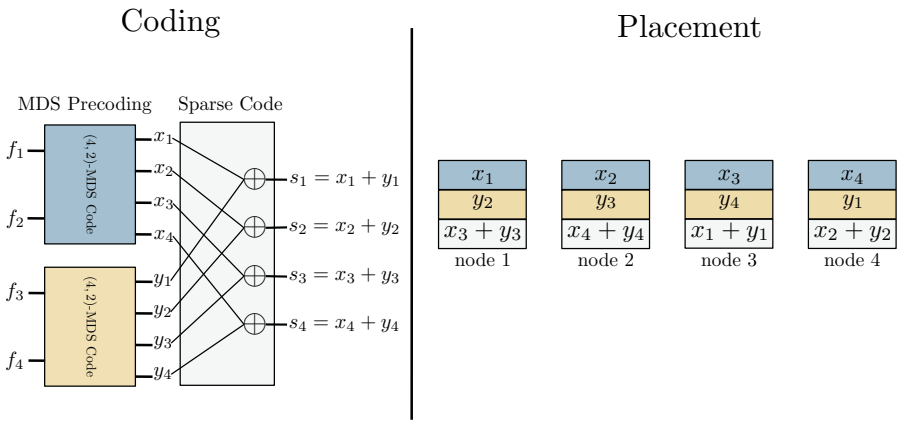

Fig. 1. Example of a $(4, 2)$-SRC. $n = 4$ storage nodes, any 2 disks recover the data and XORs of degree 2 provide simple repair.

and should be able *to tolerate any combination of* $n - k$ *failures* without data loss. We refer to codes that have this reliability as "(n, k) *erasure codes*," or codes that have "*the* (n, k) *property*."

One well-known class of erasure codes that have this property is the family of maximum distance separable (MDS) codes [3], [8]. In short, an MDS code is a way to take a data object of size M, split it into chunks of size M/k and create n coded chunks *of the same size* that have the (n, k) property. It can be seen that MDS codes achieve the (n, k) property with the minimum storage overhead possible: any k storage nodes jointly store M bits of useful information, which is the minimum possible to guarantee recovery.

Our second requirement is efficient exact repair [3]. When one node fails or becomes unavailable, the stored information should be easily reconstructable using other surviving nodes. Simple regenerating codes achieve the (n, k) property and simple repair simultaneously by separating the two problems. Two MDS pre-codes are used to provide reliability against any $n-k$ failures, while very simple XORs applied *over the MDS coded packets* provide efficient exact repair when single node failures happen.

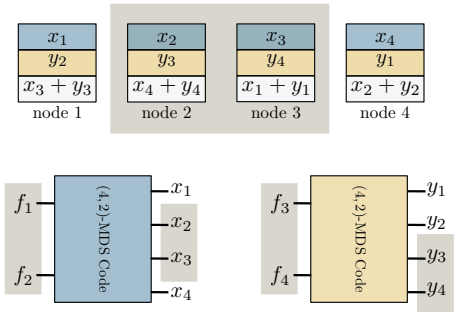

Fig. 2. File reconstruction of a $(4, 2)$-SRC.

We give a first overview of our construction through a simple example in Fig. 1, which shows an $(n = 4, k = 2)$-SRC. The original data object is split in 4 chunks f_1, f_2, f_3, f_4. We first encode $[f_1 \;\; f_2]$ in $[x_1 \;\; x_2 \;\; x_3 \;\; x_4]$ and $[f_3 \;\; f_4]$ in $[y_1 \;\; y_2 \;\; y_3 \;\; y_4]$ using any standard $(4, 2)$ MDS code. This can be easily done by multiplication of the data with the 2×4 generator matrix $\mathbf{G}$ of the MDS code to form $[x_1 \;\; x_2 \;\; x_3 \;\; x_4] = [f_1 \;\; f_2]\mathbf{G}$ and $[y_1 \;\; y_2 \;\; y_3 \;\; y_4] = [f_3 \;\; f_4]\mathbf{G}$. Then we generate a parity out of each "level" of coded chunks, i.e., $s_i = x_i + y_i$, which results in an aggregate of 12 coded chunks. We circularly place these coded chunks in 4 nodes, each storing 3, as shown in Fig. 1.

It is easy to check that this code has the (n, k) property and in Fig. 2 we show an example by failing nodes 1 and 4. Any two nodes contain two x_i and two y_i coded chunks which through the outer MDS codes can be used to recover the original data object. We note that the parity chunks are not used in this process, which shows the sub-optimality of our construction.

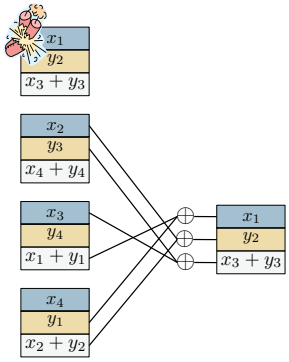

Fig. 3. The repair of node 1 in a $(4, 2)$-SRC

In Fig. 3, we give an example of a single node repair of the $(4, 2)$-SRC. We assume that node 1 is lost and a newcomer joins the system. To reconstruct x_1, the newcomer has to download y_1 and s_1 from nodes 3 and 4. This simple repair scheme is possible due to the way that we placed the coded chunks in the 4 storage nodes: each node stores 3 chunks with different index. The newcomer reconstructs each lost chunk by downloading, accessing, and XORing 2 other coded chunks. In this process the outer MDS codes are not used. In the following subsection, we present the general (n, k) construction.

A. Code Construction

Let a file $\mathbf{f}$, of size $M = 2k$, that we cut into 2 parts $\mathbf{f} = \left[\mathbf{f}^{(1)}\ \mathbf{f}^{(2)}\right]$ where $\mathbf{f}^{(i)} \in \mathbb{F}^{1\times k}$, for $i \in [2]$, $[N] = \{1, \ldots, N\}$, and $\mathbb{F}$ is the finite field over which all operations are performed. Our coding process, is a two-step one: first we independently encode each of the file parts using an outer MDS code and generate simple XORs out of them. Then, we place in a specific way the coded chunks and the simple XORs in n storage components. This encode and place scheme enables easy repair of lost coded chunks and arbitrary erasure tolerance.

We use an (n, k) MDS code to encode *independently* each of the 2 file parts $\mathbf{f}^{(1)}$ and $\mathbf{f}^{(2)}$, into two coded vectors, $\mathbf{x}$ and $\mathbf{y}$, of length n

$$\mathbf{x} = \mathbf{f}^{(1)}\mathbf{G} \text{ and } \mathbf{y} = \mathbf{f}^{(2)}\mathbf{G}, \tag{1}$$

where $\mathbf{G} \in \mathbb{F}^{k\times n}$ is the generator matrix of the (n, k) MDS code. The maximum distance of the code ensures that any k encoded chunks of $\mathbf{x}$ or $\mathbf{y}$ can reconstruct $\mathbf{f}^{(1)}$ or $\mathbf{f}^{(2)}$, respectively. Then, we generate a simple XOR vector

$$\mathbf{s} = \mathbf{x} + \mathbf{y}. \tag{2}$$

The above process yields $3n$ chunks: $2n$ coded chunks in the vectors $\mathbf{x}$ and $\mathbf{y}$, and n simple parity chunks in the vector $\mathbf{s}$.

We proceed by placing these $3n$ chunks in n storage nodes in the following way: each storage node will store 3 coded chunks, one from $\mathbf{x}$, one from $\mathbf{y}$, and one from the parity vector $\mathbf{s}$. We require that these 3 coded chunks do not share a subscript. The following circular placement of chunks satisfies this requirement for any $n \geq 2$.

node 1	node 2	...	node $n-2$	node $n-1$	node n
x_1	x_2	...	x_{n-2}	x_{n-1}	x_n
y_2	y_3	...	y_{n-1}	y_n	y_1
s_3	s_4	...	s_n	s_1	s_2

B. Erasure Resiliency and Effective Coding Rate

The (n, k)-SRC can tolerate any possible combination $n-k$ erasures and has effective coding rate $\frac{2}{3}\cdot\frac{k}{n}$. The (n, k) property of the SRC is inherited by the underlying MDS outer codes: we can always retrieve the file by connecting to any subset of k nodes of the storage array. Any subset of k nodes contain k coded chunks of each of the two file parts $\mathbf{f}^{(1)}$ and $\mathbf{f}^{(2)}$, which can be retrieved by inverting the corresponding $k \times k$ submatrices of the MDS generator matrix $\mathbf{G}$.

The coding rate (space efficiency) R of the (n, k)-SRC is equal to the ratio of the total amount of useful stored information, to the total amount of data that is stored

$$R = \frac{\text{file size}}{\text{storage spent}} = \frac{2\cdot k}{3\cdot n}, \tag{3}$$

and is upper bounded by $\frac{2}{3}$. If we fix the erasure tolerance, $n - k = m$, an SRC can have R arbitrarily close to $\frac{2}{3}$ since $\frac{2}{3}\frac{k}{k+m} \overset{k\to\infty}{\longrightarrow} \frac{2}{3}$.

C. Repairing Lost Chunks

When a single node is lost in an (n, k)-SRC the repair process is initiated. The SRCs enable easy repair of single lost coded chunks, or single node failures, due to the fact that each chunk that is lost shares an index with 2 more coded chunks stored in 2 nodes. By contacting these 2 remaining nodes, we can repair the lost chunk by a simple XOR operation. The repair of a single node failure costs 6 in repair bandwidth and chunk reads, and 4 in disk accesses.

Let for example node $i \in [n]$ fail, that say contains the chunks x_i, y_{i+1}, and s_{i+2}.[1] Then, to reconstruct x_i, we need to connect to node $i-2$ that contains the parity s_i and to node $i-1$ that contains y_i and simply XOR the two chunks. We can similarity repair y_{i+1} by accessing disk $i+1$ and downloading x_{i+1}, then accessing disk $i-1$ and downloading s_{i+1}, and then by XORing the downloaded chunks. Again, the simple parity s_{i+2} is repaired in a similar manner by accessing nodes $i+2$ and $i+1$ from which we download, read, and XOR chunks x_{i+2} and $\mathbf{y}_{i+2}$, respectively. Hence, the repair of a single chunk costs 2 disk accesses, 2 chunk reads, and 2 downloads. To repair a single node failure an aggregate of 6 chunk downloads and reads is required. The set of disks that are accessed to repair all chunks of node i is $\{i-2, i-1, i+1, i+2\}$, for $i \in [n]$, Hence, the number of disk accesses is $d = \min(n-1, 4)$.

III. Simulations

A. Simulator Introduction

We first present the architecture of the cloud storage system that our simulator is modeling. The architecture contains one master server and a number of data storage servers, similar to that of GFS [19] and Hadoop [20]. As a cloud storage system may store up to tens of petabytes of data, we expect numerous failures and hence fault tolerance and high availability are critical. To offer high data reliability, the master server needs to monitor the health status of each storage server and detect failures promptly.

In the systems of interest, data is partitioned and stored as a number of fixed-size chunks, which in Hadoop can be 64MB or 128MB. Chunks form the smallest accessible data units and in our system are set to be 64MB. To tolerate storage server failures, replication or erasure codes are employed to generate redundant coded chunks. Then, several coded chunks are grouped and form a redundancy set [21]. If one chunk is lost, it can be reconstructed from other surviving coded chunks. To repair the chunks due to a failure event, the master server will initiate the repair process and schedule repair jobs.

We implemented a discrete-event simulator of a cloud storage system using a similar architecture and data repair mechanism as Hadoop. To provide accurate simulation results, our simulator models most entities of the involved components such as machines and chunks. When performing repair jobs,

[1]The index subtractions an additions are performed over the ring $\{1, \ldots, n\}$ (for example $1 - 1 = n$).

the simulator keeps track of the details of each repair process which gives us a detailed performance analysis.

B. Simulator Validation

We first calibrated our simulator to accurately model the data repair behavior of Hadoop. During the validation, we ran one experiment on a real Hadoop system. This system contains 16 machines, which are connected by a 1Gb/s network. Each machine has about 410GB data, namely approximately 6400 coded chunks. Then, we manually failed one machine, and let Hadoop repair the lost data. After the repair was completed, we analyzed the log file of Hadoop and derived the repair time of each chunk. Next, we ran a similar experiment in our simulator. We also collected the repair time of each chunk from the simulation. In Fig. 4, we present the CDF of the repair time of both experiments.

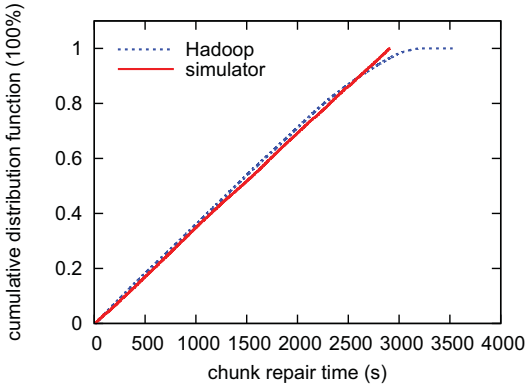

Fig. 4. CDF of repair time

Fig. 4 shows that the repair result of the simulation matches the results of the real Hadoop system very well, particularly when the percentile is below 95. This indicates that the simulator can precisely simulate the data repair process of Hadoop.

C. Storage Cost Analysis

Now we observe how storage overhead varies when we grow (n, k). We compare three codes: 3-way replication, Reed-Solomon (RS) codes, and SRC. To make the storage overhead easily understood, we define the cost of storing one byte as the metric of how many bytes are stored for each useful byte. Obviously, high cost results in high storage overhead. As 3-way replication is a popularly used approach, we use it as the base line for comparison. The results are presented in Fig. 5.

Fig. 5 shows that when $n - k$ is fixed, the normalized cost of both the RS-code and the SRC decreases as n grows. When (n, k) grows to $(50, 46)$, the normalized cost of the SRC is 0.54, and that of the RS-code is 0.36. In other words, $(50, 46, 2)$ SRCs need approximately half the storage of 3-way replication.

D. Repair Performance

In this experiment, we measure the throughput of repairing one failed data server. The experiment involves a total of 100

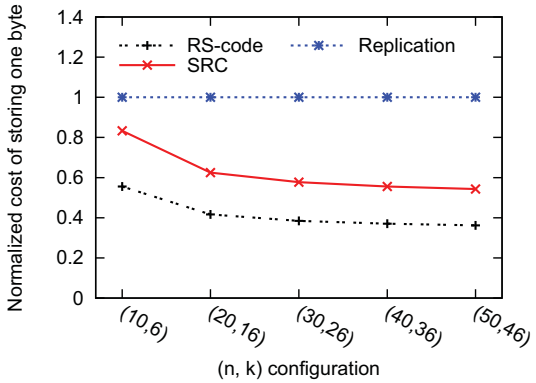

Fig. 5. Storage cost comparison

machines, each storing 410GB of data. We fail at random one machine and start the data repair process. After the repair is finished, we measure the elapsed time and calculate the repair throughput. The results are shown in Fig. 6. Note that the throughput of using 3-way replication is constant across different (n, k) since there is no such dependency on these parameters. From Fig. 6 we can make two observations. First,

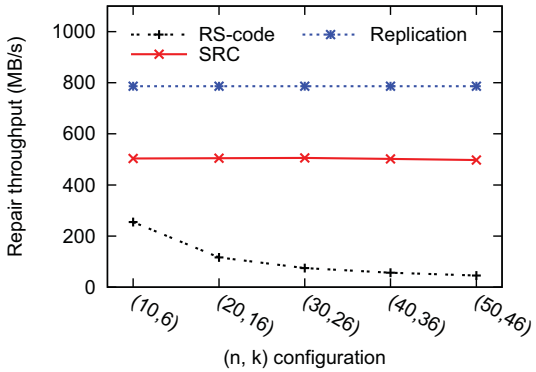

Fig. 6. Repair performance comparison

3-way replication has the best repair performance followed by SRC, while the RS-code offers the worst performance. This is not surprising due to the amount of data that has to be accessed for the repair. Second, the repair performance of the SRC remains constant on various (n, k), but the performance of the RS-code becomes much worse as n grows. This is one of the major benefits of SRCs, i.e., the repair performance can be independent from (n, k). Furthermore, the repair throughput of the SRC is about 500MB/s, approximately 64% of the 3-way replication's performance.

E. Data Reliability Analysis

Now we analyze the data reliability of an SRC cloud storage system. We use a simple Markov model [22] to estimate the reliability. For simplicity, failures happen only to disks and we assume no failure correlations.

We assume that the mean time to failure (MTTF) of a disk is 5 years and the system stores 1PB data. To be conservative,

the repair time is 15 minutes when using 3-way replication and 30 minutes for the SRC, which is in accordance to Fig. 6. In the case of the RS-code, the repair time depends on k of (n, k). With these parameters, we first measure the reliability of one redundancy set, and then use it to derive the reliability of the entire system. The estimated MTTF of the entire storage system is presented in Fig. 7. Fig. 7 shows that the data

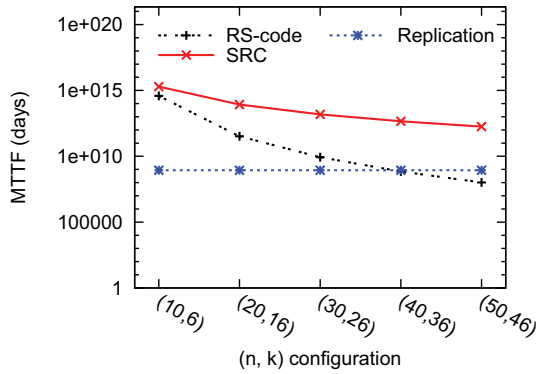

Fig. 7. MTTF comparison

reliability of the 3-way replication is in the order of 10^9. This is consistent with the results in [22]. We can observe that the reliability of SRCs is much higher than 3-way replication. Even for the high rate (low storage overhead) $(50, 46)$ case, SRCs are several orders of magnitude more reliable than 3-way replication. This is benefited from the high repair speed of SRCs. RS codes show a significantly different trend. Although the reliability of $(10, 6)$ and $(20, 16)$ are higher than 3-way replication, the reliability of the RS-code reduces greatly when (n, k) grows. This happens because their repair performance rapidly decreases as k grows.

IV. Conclusions

We introduced a novel family of distributed storage codes that are formed by combining MDS codes and simple locally repairable parities for efficient repair and high fault tolerance. One very significant benefit is that the number of nodes that need to be contacted for repair is always 4, that is independent of n, k. Further, SRCs can be easily implemented by combining any prior MDS code implementation with XORing of coded chunks and the appropriate chunk placement into nodes. We presented a comparison of the proposed codes with replication and Reed-Solomon codes using a cloud storage simulator. The main strength of the SRC in this comparison is that it provides approximately four more zeros of data reliability compared to replication, for approximately half the storage. The comparison with Reed-Solomon leads almost certainly to a win of SRCs in terms of repair performance and data availability when more storage is allowed. Our preliminary investigation therefore suggests that SRCs should be attractive for real cloud storage systems. In conclusion, we think that SRCs add new feasible points in the tradeoff space of distributed storage codes.

References

[1] The Coding for Distributed Storage wiki `http://tinyurl.com/storagecoding`

[2] A. G. Dimakis, P. G. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," in *IEEE Trans. on Inform. Theory*, vol. 56, pp. 4539 – 4551, Sep. 2010.

[3] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," in *IEEE Proceedings*, vol. 99, pp. 476 – 489, Mar. 2011.

[4] A. Asterjadhi, E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. "Towards network coding-based protocols for data broadcasting in wireless ad hoc networks." in *IEEE Transactions on Wireless Commun.*, vol. 9, pp. 662 – 673, Feb. 2010

[5] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive view of a live network coding P2P system," in IMC, Association for Computing Machinery, Inc., Oct 2006

[6] O. Khan, R. Burns, J. Plank, and C. Huang, "In search of I/O-optimal recovery from disk failures," to appear in *Hot Storage 2011, 3rd Workshop on Hot Topics in Storage and File Systems*, Portland, OR, Jun., 2011.

[7] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: a quantitative comparison," *in Proc. IPTPS*, 2002.

[8] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in raid architectures," in *IEEE Transactions on Computers*, 1995.

[9] L. Xiang, Y. Xu, J.C.S. Lui, and Q. Chang, "Optimal recovery of single disk failure in RDP code storage systems" in *Proc. ACM SIGMETRICS (2010) international conference on Measurement and modeling of computer systems*

[10] F. Oggier and A. Datta, "Self-repairing homomorphic codes for distributed storage systems," in *Proc. IEEE Infocom 2011*, Shanghai, China, Apr. 2011.

[11] K.V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran "Explicit construction of optimal exact regenerating codes for distributed storage," In *Allerton Conf. on Control, Comp., and Comm.*, Urbana-Champaign, IL, September 2009.

[12] C. Suh and K. Ramchandran, "Exact regeneration codes for distributed storage repair using interference alignment," in *Proc. 2010 IEEE Int. Symp. on Inform. Theory (ISIT)*, Seoul, Korea, Jun. 2010.

[13] K. 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," submitted to IEEE Transactions on Information Theory. Preprint available at at http://arxiv.org/pdf/1005.4178.

[14] S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in Proc. of *48th Allerton Conf. on Commun., Control and Comp.*, Monticello, IL, September 2010.,"

[15] I. Tamo, Z. Wang, and J. Bruck, "MDS array codes with optimal rebuilding," to appear in *2011 IEEE Symposium on Information Theory (ISIT)*. Preprint available at http://arxiv.org/abs/1103.3737

[16] V. R. Cadambe, C. Huang, S. A. Jafar, and J. Li, "Optimal repair of MDS codes in distributed storage via subspace interference alignment," *arxiv pre-print 2011*. Preprint available at http://arxiv.org/abs/1106.1250.

[17] K. W. Shum and Y. Hu, "Exact minimum-repair-bandwidth cooperative regenerating codes for distributed storage systems," to appear in *2011 IEEE Symposium on Information Theory (ISIT)*. Preprint available at http://arxiv.org/abs/1102.1609.

[18] D. S. Papailiopoulos, A. G. Dimakis, and V. R. Cadambe, "Repair optimal erasure codes through Hadamard designs," Preprint available at http://arxiv.org/abs/1106.1634v1.

[19] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *SOSP '03: Proc. of the 19th ACM Symposium on Operating Systems Principles*, 2003.

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *MSST '10: Proc. of the 26th IEEE Symposium on Massive Storage Systems and Technologies*, 2010.

[21] Q. Xin, E. L. Miller, D. D. E. Long, S. A. Brandt, T. Schwarz, and W. Litwin, "Reliability mechanisms for very large storage systems," in *MSST '03: Proc. of the 20th IEEE Symposium on Massive Storage Systems and Technologies*, 2003.

[22] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. T. L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *OSDI '10: Proc. of the 9th Usenix Symposium on Operating Systems Design and Implementation*, 2010.