# On repairing erasure coded data in an active-passive mixed storage network

Datta, Anwitaman; Oggier, Frédérique

2015

Oggier, F., & Datta, A. (2015). On repairing erasure coded data in an active-passive mixed storage network. International journal on information and coding theory, 3(1).

https://hdl.handle.net/10356/79416

https://doi.org/10.1504/IJICOT.2015.068697

# On Repairing Erasure Coded Data in an Active-Passive Mixed Storage Network


## Frédérique Oggier

School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
E-mail: frederique@ntu.edu.sg

## Anwitaman Datta

School of Computer Engineering
Nanyang Technological University, Singapore
E-mail: anwitaman@ntu.edu.sg



**Abstract:** A major change has been recently witnessed in networked distributed storage systems (NDSS), with increased use of erasure codes in lieu of replication for realizing data redundancy. Yet, both the industry and academic research communities are still trying to derive most advantage from the trade-offs between storage overhead and reliability that erasure codes provide, while optimizing them to satisfy specific storage needs like repairability and better degraded read performance.

Many newly proposed coding strategies for NDSS exploit the storage nodes' computational ability to improve on repairability by applying network coding techniques. However, not every storage node in a NDSS is necessarily endowed with computing capability.

This paper studies the effects of passive nodes, i.e., without computational ability, on the repairability of erasure coded data, and how they may impair the promised performance of novel coding techniques. In particular, we provide a lower bound on the minimum storage overhead needed, which could be used as a guideline to evaluate the price to pay in storage overhead to achieve bandwidth efficient repairability when combining passive nodes with erasure coding techniques.



**Keywords:** Erasure Codes, Networked Distributed Storage Systems, Repairability.


## 1 Introduction

The problem of repairing erasure-coded data in networked distributed storage systems, i.e., replenishing data redundancy lost due to failure of storage nodes, has been extensively researched in the recent years. When some nodes fail, the data that they contained needs to be reproduced at some live nodes, to maintain the level of redundancy in the system,

hence ensuring reliability over time. Different code designs have been proposed for a better repairability (see [10] for a survey): (i) presence of local parities ([6] originally for improved degraded reads, [7, 5, 3] also for degraded reads), (ii) reduction of repair bandwidth [2, 14, 8], or (iii) minimization of the number of live nodes contacted for repair [9, 13, 11, 4]. One of the earliest and prominent approach for bandwidth efficient repairs is based on network-coding techniques first proposed in [2], where it is assumed that all the storage nodes can actively carry out necessary computational tasks. The cornucopia of follow-up works on the topic are based on the same assumption that all storage nodes (can) play an active role in the repair process, while some code instances are also known to perform repair on the contrary by simply transferring pieces of data (see e.g. [12] for an earlier construction of so-called repair by transfer). The assumption that all nodes are active fits certain environments, e.g. peer-to-peer storage systems or compute-centric storage clusters such as HDFS [1], where all the nodes typically do have computing resources. However, in other storage intensive application scenarios, such as 'cold storage' or backup of data seldom used, investing on computationally powerful hardware attached with each storage unit may not make sense. Instead, one often encounters systems comprising of a mix of nodes - some of which provide only storage - i.e., data can be written into or read from them, but these nodes do/can not participate in any computational tasks, while the others can carry out both storage as well as computations. Another potential application is that of a storage system upgrade, where active nodes are added to a cluster composed of only active nodes, to improve the system repairability, and/or reliability.

It is thus important to revisit the problem of repairability of erasure coded distributed storage systems taking into account the possible disparity of storage nodes in terms of their ability to compute. To that end, we make the following main contributions in this paper:

(i) We generalize the original min-cut bound analysis [2] to take into account passive and active nodes, yielding a lower bound on the storage overhead as a function of the repair bandwidth of active and passive nodes. The original storage overhead - repair bandwidth trade-off is obtained as the particular case where all storage nodes are active. Likewise, the traditional approach of repairing erasure coded data, where one node would download enough encoded blocks and re-encode and re-insert the lost data into the system is obtained as another extreme case where all storage nodes are passive. We further focus on two particular regimes comprising a mixture of passive and active nodes, when both active and passive nodes have the same download bandwidth, and when they have the same repair degree.

(ii) We illustrate how some known code instances fit different regimes exhibited by our analysis.

(iii) We draw conclusions from our study, in terms of impairment provoked by the presence of passive nodes, in particular in terms of increase of storage overhead. This is particularly useful for the two regimes that include a mixture of passive and active nodes.

This paper is organized as follows. In Section 2, we formalize erasure coding for networked distributed storage systems, and propose a model for active and passive nodes, together with a motivating example. Section 3 contains a generic bound, where two extreme cases are identified (the case of all nodes being active, respectively passive), together with fitting known code instances. Section 4 and Section 5 are each dedicated to a special regime of mixed active-passive nodes, respectively when both active and passive nodes have the same repair degree (Section 4), and when they have the same download bandwidth (Section 5), where a refinement of the lower bound on the storage overhead is computed, and illustrated. Insights on the results are elaborated.

## 2 System Model

Consider a network of storage nodes, where data objects are stored using erasure codes, or more precisely linear erasure codes. Given a data object $\mathbf{o}$, represented as a row vector of length $B$ over some alphabet, typically the finite $\mathbb{F}_q$, with $q$ a prime power, a linear erasure code is a family of $l$ row vectors $v_i$ of length $B$, $l \geq B$. Encoding consists of computing the inner products $\mathbf{o}v_i^T$, $i = 1, \ldots, l$ which are referred to as encoded pieces, or encoded fragments. This may be written in matrix form as

$$\mathbf{o}\left[v_1^T\ v_2^T \ldots v_l^T\right] = [\mathbf{o}v_1^T, \ldots, \mathbf{o}v_l^T].$$

As such, an $(l, B)$ linear erasure code is a linear map that transforms the data object $\mathbf{o} \in \mathbb{F}_q^B$ to $(\mathbf{o}v_1^T, \ldots, \mathbf{o}v_l^T) \in \mathbb{F}_q^l$, and the $l$ encoded fragments are allocated to $N$ storage nodes across the network. Typically $l$ divides $N$, so that every node stores the same number of encoded fragments from $\mathbf{o}$. Replication is, for instance, a particular case of erasure coding. Take $l = 3B$, and the vectors $v_i = v_{B+i} = v_{2B+i}, i = 1, \ldots, B$ to be the whole zero vector, but for a 1 in the $i$th position, that is:

$$\mathbf{o}\left[\mathbf{I}_B\ \mathbf{I}_B\ \mathbf{I}_B\right] = [\mathbf{o}, \mathbf{o}, \mathbf{o}],$$

where $\mathbf{I}_B$ is the $B$-dimensional identity matrix. Then $\mathbf{o} \in \mathbb{F}_q^B \mapsto (\mathbf{o}, \mathbf{o}, \mathbf{o}) \in \mathbb{F}_q^{3B}$, namely, we get a 3-way replication scheme.

### *2.1 Active versus Passive Storage Nodes*

We are interested in the repair processes, given that the data objects are now encoded using linear erasure codes. Suppose that a node fails. Then, to reconstruct the content lost due to this failure, some storage node will have to contact several live nodes, download data from them, and possibly perform computations, depending on the erasure code used.

The model that we propose to study in this paper assumes that *not all* storage nodes are endowed with computational abilities. Thus we distinguish two kinds of storage nodes, *active* and *passive* nodes. We call a node to be active if it has (uses) computational capabilities for carrying out repairs, and passive otherwise. Among the $N$ nodes storing a data object $\mathbf{o}$, we assume that $N_1$ are active storage nodes, while the other $N_2$ nodes are passive ($N = N_1 + N_2$). Note that the total number of storage nodes in an overall system $N_{tot}$ could be much larger than the number of nodes used to store a single object, i.e., $N_{tot} >> N$.

To understand the effect of passive nodes in the storage network, we adopt the point of view of trade-offs between the amount of stored data, versus the repair bandwidth needed to perform one node repair.

In this context, we make the assumption that passive nodes are able to up/download data without loss of generality. Indeed, to be useful, the data contained in the storage nodes need to be accessible, thus either (1) the nodes are capable of transferring the data by themselves, or (2) there must be some helper nodes in the system, which can access the storage nodes, to either read or write some content. Now saying that the helper node(s) are only accessing data (with no computational ability) is equivalent, as far as bandwidth is concerned, to say that the storage nodes are transferring data themselves. If the helper node(s) however also have computational power, the scenario reduces to have that of active nodes. Likewise, in this model, if such helper nodes are used, it is assumed that they have no storage capacity.

Otherwise, it is just the same as passive nodes which can transfer data on their own. Helper nodes, if any, are thus not counted among the $N$ storage nodes, but justify our assumption.

We provide a small example to motivate our approach. We will also use this example to illustrate several of the different features that are desired for a storage code. Even though this paper focuses on the storage overhead and repair bandwidth, other properties - reliability and degraded reads- should be kept in mind.

**Example 1:** Consider $N = 5$ nodes, one of which is passive, meaning that it can only repair a node failure by downloading existing data, but cannot compute. These $N = 5$ nodes store a data object $\mathbf{o}$ represented as twelve pieces of data denoted by $A_1, \ldots, A_8$, $P_1, \ldots, P_4$, where the notation is chosen to emphasize that the passive node stores only the four $P_i$s, while the active nodes store the eight first pieces, as well as some other linear combinations (or copy) of the twelve pieces. The erasure code used (1) keeps the original data $A_1, \ldots, A_8$, $P_1, \ldots, P_4$, (2) replicates $P_1, \ldots, P_4$, and (3) computes four encoded blocks from $A_1, \ldots, A_8$, which are $A_3 + A_6, A_5 + A_8, A_7 + A_2, A_1 + A_4$. Using the formalism of the system model, this erasure code may alternatively be described using the vectors $v_1, \ldots, v_{20}$: the vectors $v_1, \ldots, v_{12}$ form a 12-dimensional identity matrix, the vectors $v_{13}, \ldots, v_{16}$ are all-zero vectors but for a 1, in the 13th, 14th, 15th and 16th position, respectively. Finally $v_{17}, \ldots, v_{20}$ are all-zero vectors but for two 1's, in positions 3,6, for $v_{17}$, 5,8 for $v_{18}$, 7,2 for $v_{19}$ and 1,4 for $v_{20}$.

*Data Placement.* In the example illustrated in Figure 1, we consider node 5 to be passive (and nodes 1 to 4 to be active), and the data along with redundancy for fault-tolerance is distributed among the five nodes in the following fashion:

node $1 : A_1, A_2, A_3 + A_6, P_1$
node $2 : A_3, A_4, A_5 + A_8, P_2$
node $3 : A_5, A_6, A_7 + A_2, P_3$
node $4 : A_7, A_8, A_1 + A_4, P_4$
node $5 : P_1, P_2, P_3, P_4$

*Object Recovery.* All the twelve original pieces of the data can be accessed by contacting any four available nodes. They can be read directly if nodes 1 through 4 are accessed, since they contain systematic pieces, and by downloading the relevant three pieces from each of these four nodes, thus incurring a total download of precisely 12 pieces of data. The term *systematic piece* refers to a piece of data which appears unchanged in a codeword.

*Degraded Read.* It refers to the need of reading an unavailable systematic piece. Suppose now that node 1 becomes unavailable. In that case, since not all the systematic pieces are directly available, the data accessor will experience a degraded read, but nevertheless be able to access all the pieces, particularly by reconstructing $A_1$ by using $A_4$ from node 2, and $A_1 + A_4$ from node 4, and likewise, $A_2$ can be derived from $A_7 + A_2$ and $A_7$ obtained from nodes 3 and 4 respectively. In this particular example, the degraded read still required the transfer of twelve data pieces, but incurred additional computation on the part of the data accessor to rebuild the missing systematic pieces.

*Repair of the Passive Node.* The 5th node can clearly be repaired passively in the event of a failure, by contacting every other live node, and downloading $P_i$ from node $i$.

*Repair of an Active Node.* If one of the other nodes fail instead, computations during the repair process cannot be avoided. If say node 1 fails, the download of (i) $A_4, A_1 + A_4$, from respectively nodes 2 and 4, (ii) $A_7, A_2 + A_7$ from respectively nodes 4 and 3, (iii)

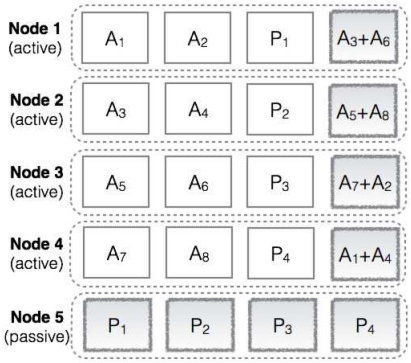


**Figure 1** An example where twelve pieces of a data object $A_1, \ldots, A_8$, $P_1, \ldots, P_4$ are stored across five nodes - four active and one passive (the last row indicates the passive node in this example). The data layout is such that the whole object can be read by contacting any four of the five nodes (tolerating a single fault). Each node is indicated using dotted box, and stores four pieces. Fault-tolerance is achieved through redundancy. The redundant information is indicated using shaded boxes. So, in this example, all in all, twenty pieces are stored in order to ensure that the original twelve pieces can be accessed even if there is one node failure, implying that the storage overhead is $20/12 = 1.667\times$. The pieces stored in any one of the five nodes can also be rebuilt using the pieces at the other nodes, but by downloading data less in volume than the size of the whole object (namely, twelve pieces), and subject to the constraint that node 5 has to be rebuilt at a passive node.

$A_3, A_6$ from respectively nodes 2 and 3, and (iv) $P_1$ from node 5, allows to complete the repair of node 1. It would involve the download of two pieces of data from nodes 2, 3 and 4 and only one piece from node 5, incurring in total, seven pieces of data transfer.

The above example emphasizes the different repair processes, depending on the computational ability of the nodes. Note furthermore that for rebuilding the affected information subsequent to a failure of either of a passive node, or an active node, the amount of data transfer is less than the amount of data corresponding to the size of the whole object. Hence, the repair of a single failure is bandwidth efficient with respect to a naive approach, where the whole data may have to be downloaded in order to re-encode and recreate the missing pieces. We will in fact see next, that there are trade-offs on how much expected bandwidth is consumed for repairs, versus how much storage overhead is necessary in the system in order to facilitate such bandwidth efficient repairs.

### *2.2 Storage Capacity and Repair Bandwidth*

We make the assumption that every storage node, active or passive, has the same storage capacity $\alpha$, that is, they store exactly $\alpha$ amount of encoded fragments of the data object $\mathbf{o}$. The length $l$ of the code is thus $N\alpha$. The $(N\alpha, B)$ linear erasure code used should be such that when a data collector wants to recover $\mathbf{o}$, he should be able to do so by contacting any choice of $k$ storage nodes. Note that there are other classes of erasure codes, e.g., those minimizing the number of live nodes to be contacted per repair [9, 13, 11], which trade this property of data retrievability from an arbitrary $k$ nodes, in order to carry out repairs with

fewer ($< k$) live nodes. The min-cut analysis argument we use in this paper does not apply to such codes.

Given the assumption of data retrievability from any choice of $k$ storage nodes, the amount $\alpha$ of data stored at each node (or number of coefficients in $\mathbb{F}_q$) must be at least $B/k$. Indeed, if there is one node which stores less than $B/k$, then by contacting a set of $k$ nodes, which includes the node that has less than $B/k$ amount of data, the total amount of data gathered would be less than $B$, and recovering the data object will not be feasible. The case when every node exactly stores $\alpha = B/k$ amount of data is sometimes referred to as the *minimum storage repair (MSR) point*. At MSR point, the linear code used is more easily linked to codes as classically described in coding theory. Then the object $\mathbf{o}$ is cut into $k$ pieces of length $B/k$, these $k$ pieces are mapped to $N$ encoded pieces, $N > k$, using an $(N, k)$ code.

The amount of data downloaded at one node per repair depends on the repair degree, which is the number of live nodes contacted (say $d_1$ nodes for active nodes, and $d_2$ nodes for passive nodes), and the download bandwidth, or amount of bandwidth used per live contacted (say $\beta_1$ from a live node to an active node, and $\beta_2$ from a live node to a passive node). Since a node stores $\alpha$ amount of data, a repaired node must contain the same amount $\alpha$, and thus the data downloaded during repair must be at least $\alpha$, that is

$$d_1\beta_1 \geq \alpha, d_2\beta_2 \geq \alpha. \tag{1}$$

For a passive node, we put the additional constraint that

$$\beta_2 d_2 = \alpha. \tag{2}$$

Indeed, if a node has no computational power, then it will store exactly what it downloaded, put otherwise, it has no reason to download more than what it stores, since it cannot compute and store any other kind of derived results instead. However, this is of course not a complete characterization, in that it is also possible for an active node to download only $\alpha$. This constraint is however enough to obtain a bound on the repair bandwidth involving passive nodes. The numbers $d_1, d_2$ of nodes contacted are assumed to be at least $k$. This is needed to deal with the minimum storage point where $\alpha = B/k$. Indeed, if a repair could be done by contacting less than $k$ nodes, then there would be linear dependencies among the vectors $v_1, \ldots, v_l$, and the data collector would not be able to reconstruct the data object by contacting these $k$ nodes with linear dependencies.

For ease of reference, the notations and assumptions explained in this section are summarized in Table 1.

## 3 General Bounds on the Repair Bandwidth

To evaluate the impact of the presence of passive nodes in the system, we make use of a standard technique [2], that of computing a min-cut bound in an information flow graph, as illustrated in Fig. 2.

The data to be stored is represented as a graph: (1) a data owner uploads the object to be stored over the network, (2) whenever there is a repair, the data flows from live nodes to the node performing the repair, and (3) when the data collector wants the object back, he contacts $k$ live nodes, and the data flows from them to him. A min-cut in this graph gives a

| | |
|---|---|
| $N$ | number of storage nodes to store one object (helper nodes are not counted) |
| $N_{tot}$ | total number of storage nodes in the overall system ($N_{tot} >> N$) |
| $\mathbf{o}$ | object to be stored |
| $B$ | size of one object to be stored across $N$ nodes |
| $k$ | number of live nodes contacted for object retrieval |
| $\alpha$ | amount of data stored at each storage node (for one object) ($\alpha \geq B/k$) |
| $v_1, \ldots, v_l$ | family of vectors that define the linear erasure code used ($\mathbf{o}v_1^T, \ldots, \mathbf{o}v_l^T$ are stored encoded fragments) |
| $N_1$ | number of active nodes storing a specific object $\mathbf{o}$ |
| $k_1$ | number of active nodes involved in a min-cut |
| $d_1$ | number of live nodes contacted by an active node to repair or repair degree ($k \leq d_1 \leq N-1$) |
| $\beta_1$ | amount of data downloaded per repair by an active node or download bandwidth ($\beta_1 d_1 \geq \alpha$) |
| $N_2$ | number of passive nodes storing a specific object $\mathbf{o}$ ($N_1 + N_2 = N$) |
| $k_2$ | number of passive nodes involved in a min-cut ($k_1 + k_2 = k$) |
| $d_2$ | number of live nodes contacted by a passive node to repair ($k \leq d_2 \leq N-1$) |
| $\beta_2$ | amount of data downloaded per repair by a passive node ($\beta_2 d_2 = \alpha$) |

**Table 1** Summary of the notations and assumptions.

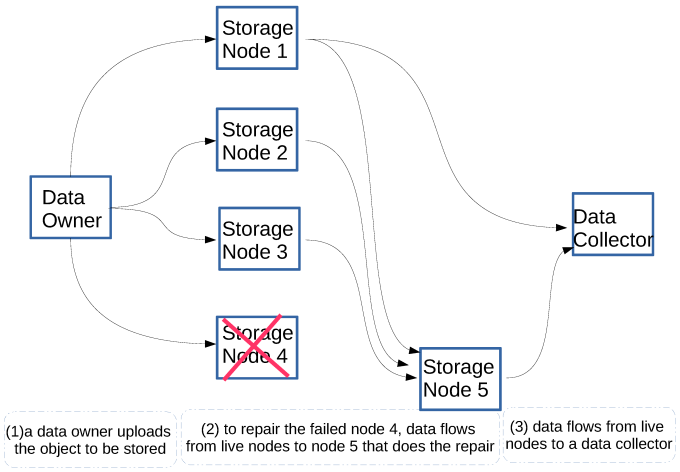


**Figure 2** An information flow graph showing the flow of the stored data from the moment it is uploaded by a data owner until it is recovered by a data collector. In case of failure, the data is repaired at another node.

constraint on the flow of data that must circulate from the data owner to the data collector, in order for the latter to recover the object. We will follow these steps:

1. Compute a min-cut bound in the information flow graph. The size $B$ of the stored object must be less than the bound to indicate that sufficient information has been retained in the system over a period of time (undergoing failures and repair processes), otherwise it would mean that too much data is lost preventing the object recovery.

2. Set equality between the min-cut bound and $B$. Under this constraint, optimize the storage capacity as a function of the repair bandwidth.

To discern the effect of passive nodes in the min-cut bound, we consider that among the $k$ nodes accessed by a data collector arbitrarily out of the $N$ nodes, $k_1$ of them are active nodes, while $k_2$ (where $k = k_1 + k_2$) are passive.

We will see in the subsequent analysis that supporting passive nodes increases the overheads. Note that in the worst case scenario, up to $N_2$ passive nodes may get contacted. Thus, once a code to be used for storing an object is decided based on an assumed choice of $k_2$ (and corresponding overheads), the system needs to ensure that the invariant $N_2 \leq k_2$ is satisfied for it to function properly.

We start with a very generic expression.

**Proposition 1:** *A min-cut bound between the data owner and data collector is*

$$\sum_{i=0}^{k-1} \min\{\alpha, (d_1 - i)\beta_1, (d_2 - i)\beta_2\} \tag{3}$$

*which is further minimized under the constraint that there are $k_1$ terms of the form $(d_1 - i)\beta_1$, and $k_2$ terms of the form $(d_2 - i)\beta_2$, with $k_i \leq N_i$.*

*A*: t the first repair, if an active node is repaired, a contribution to the cut involves the minimum between the storage capacity, and the repair bandwidth, which is of $d_1\beta_1$. It is also the minimum between the storage capacity and the repair bandwidth, of $d_2\beta_2$ this time, if the node is passive. This gives

$$\min\{\alpha, d_1\beta_1, d_2\beta_2\}.$$

At the $i + 1$th repair, the contribution to the cut is similarly

$$\min\{\alpha, (d_1 - i)\beta_1, (d_2 - i)\beta_2\}.$$

Now, there are exactly $k_1$ active nodes which were repaired, thus there are exactly $k_1$ terms of the forms $(d_1 - i)\beta_1$ in the sum, and $k_2$ terms of the forms $(d_2 - i)\beta_2$. Since there is no control on the sequence in which nodes may fail (and be repaired accordingly), and from which live nodes data is downloaded for repairs, but yet $B$ amount of data does need to flow through the network, the minimum of (3) must be considered, over all the possible configurations of $k_1$ active and $k_2$ passive nodes. □

The min-cut bound (3) takes nicer forms when something is known about $d_1, d_2$ and $\beta_1, \beta_2$. We will next study some specific cases accordingly. Before doing so, we will like

to highlight a few crucial aspects on how to interpret the results of the presented min-cut bound analysis.

The min-cut bound provides a necessary condition which needs to be satisfied for an object to be retrievable, but the analysis itself does not guarantee (i.e., the analysis does not provide a sufficient condition for) the existence of a code which may realize the properties. Nor does the analysis itself provide any concrete clues on relevant code constructions.

Furthermore, we only study the information flow, but not how this information is actually represented. Hence, hypothetically speaking, when some pieces of data are lost, the repair process may lead to the creation of pieces which are not bit-wise identical, even though the same amount of information is preserved. In the literature, coding strategies which recreate bit-wise identical pieces are known as *exact repair*, while those that recreate the redundancy for functional correctness, but not identical to what was lost, are known as *functional repair*.

Finally, we will like to note that the study carried out in this paper looks at single repair process at a time in isolation, though over a period of time, many such single repairs may happen in sequence. For the case of all active nodes, a generalization has been studied in the literature, variously known as collaborative/cooperative regenerating codes [14, 8] which investigate the scenario where multiple failures are simultaneously repaired, and the repairing nodes collaborate among each other. Such collaboration can further reduce the repair cost per repair. Our current work may likewise be generalized to the case of multiple simultaneous repairs, an aspect we defer for the future. Here, we focus only on the case of single repair at a time. Note that, multiple, uncoordinated repairs may be performed in parallel, but it will just not have the benefit of further repair bandwidth reduction that collaborative repairs provide.

### *3.1 The Active Node Only Case*

When all the nodes involved in the min-cut are active, then $k = k_1$ and this is the result obtained in [2].

**Corollary 2:** *If all the $k$ nodes involved in the min-cut are active, then*

$$\sum_{i=0}^{k-1} \min\{\alpha, (d_1 - i)\beta_1\}.$$

In this case, $k_2 = N_2 = 0$, and the repair bandwidth is $d_1\beta_1$ for every repair node.

The min-cut tells how much information is necessary to be preserved inside the network for object recovery. Given this constraint, one may minimize the storage overhead, as shown in [2], where it was proven that under the constraint

$$\sum_{i=0}^{k-1} \min\{\alpha, (d_1 - i)\beta_1\} = B,$$

the minimum storage overhead $\alpha = \alpha(d, \beta)$ is a piecewise linear function given by

$$\alpha(d,\beta) = \begin{cases} B/k & d\beta \in [f(0), +\infty) \\ \frac{B - \left(\frac{(2d-2k+i+1)i}{2d}\right)d\beta}{k-i} & d\beta \in [f(i), f(i-1)) \end{cases} \tag{4}$$

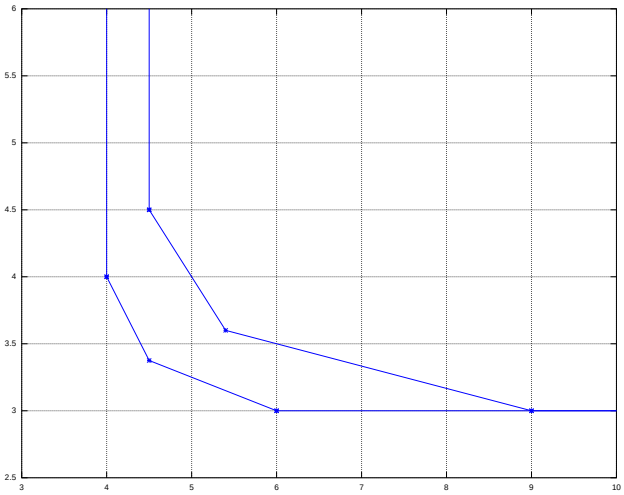


**Figure 3** The storage $\alpha$ on the $y$-axis is shown as a function the repair bandwidth $d\beta$ shown on the $x$-axis. The parameters used are those of Example 2, namely $B = 9$, $k = 3$, $d = 4$ for the lower curve, and $d = 3$ for the upper one.

where

$$f(i) = \frac{2Bd}{(2k-i-1)i + 2k(d-k+1)}. \tag{5}$$

An illustration of the trade-off is found on Figure 3. We observe that the minimum storage $\alpha$ at every node is 3 (for both curves), while the minimum repair bandwidth is 4 (only for the lower curve).

### *3.2 The Passive Node Only Case*

If every node is passive, then $N_1 = k_1 = d_1 = 0$, and $k = k_2$. The constraint $d_2\beta_2 = \alpha$ then gives the following simplified min-cut bound.

**Corollary 3:** *If all the $k$ nodes involved in the min-cut are passive, then a min-cut bound is*

$$k\left(\alpha - \beta_2\frac{k-1}{2}\right).$$

*U*: sing (3), the min-cut is given by

$$\sum_{i=0}^{k-1}\min\{\alpha,(d_2-i)\beta_2\} = \sum_{i=0}^{k-1}\min\{\alpha,\alpha-i\beta_2\}$$

$$= k\alpha - \sum_{i=0}^{k-1} i\beta_2$$
$$= k\alpha - \beta_2 \frac{k(k-1)}{2}$$

since $\alpha = d_2\beta_2$ □

From

$$k\left(\alpha - \beta_2 \frac{k-1}{2}\right) = B,$$

we get a repair bandwidth per node of $d_2\beta_2$ and

$$\boxed{\alpha - \beta_2 \frac{k-1}{2} = B/k.}$$

*The minimum storage repair (MSR) point*

Clearly if $\alpha = B/k$, this forces

$$\beta_2 \frac{k-1}{2} = 0$$

that is $k = 1$, corresponding to replication, and conversely, $k = 1$ implies that $\alpha = B/k$.

*Beyond the MSR point*

Suppose now that $\alpha = a(B/k)$ for some $k > a > 1$. Then

$$\beta_2 = \frac{B}{k}\frac{2(a-1)}{k-1}. \tag{6}$$

For example, when $a = 2$ and $k = 3$, then

$$\beta_2 = \frac{B}{k}\frac{2}{k-1}.$$

and thus:

$$\alpha = 2B/3,\ \beta_2 = B/3.$$

The following family of codes proposed in [12] carries out repair by transfer, and is suitable for the all passive nodes scenario. Consider a complete graph of $N$ vertices, thus with $(N-1)N/2$ edges. Label every edge by $i$, $i = 1, \ldots, (N-1)N/2$, and each of the $N$ nodes stores $N-1$ amount of data, given by $\mathbf{o}v_i^T$, where $i$ corresponds to the edges of

the chosen node. To retrieve the object, the content of $k$ nodes is needed, thus if the object is of length $B$, one gets

$$k\,(N-1)$$

equations. If the $v_i$ are all linearly independent (in particular, $B \geq (N-1)N/2$), then

$$k\,(N-1) - k\frac{k-1}{2}$$

linearly independent equations, which is then the size $B$ of the object.

**Example 2:** Suppose that the network is composed of $N = 5$ passive storage nodes. Then the graph has 10 edges, and the code is made of 10 vectors $v_1, \ldots, v_{10}$ which are linearly independent, e.g., $v_i$ is the whole zero vector with a 1 in the $i$th position, for $i = 1, \ldots, 9$, and $v_{10} = \sum_{i=1}^{9} v_i$. These vectors $v_i$ need to have the same length as the object, that is $B$, and thus $B \geq 9$. Every node stores $\alpha = N - 1 = 4$ amount of data. When contacting any choice of $k = 3$ nodes, we get $3 \cdot 4 - 3 = 9$ linearly independent equations, and thus the size $B$ of the object has to be 9. The explicit data placement is:

node $1 : v_1, v_2, v_3, v_4$
node $2 : v_1, v_5, v_6, v_7$
node $3 : v_2, v_5, v_8, v_9$
node $4 : v_3, v_6, v_8, v_{10}$
node $5 : v_4, v_7, v_9, v_{10}$

One node failure is easily repaired by contacting all the other live nodes and downloading one fragment per edge.

Since $B/k = 9/3 = 3$ and $\alpha = 4$, we have that $\alpha = a(B/k)$ with $a = 4/3$, and from (6)

$$\beta_2 = \frac{B}{k}\frac{2(a-1)}{k-1} = \frac{B}{k}\frac{2}{3(k-1)} = 1.$$

We see from Figure 3 that this code does not provide the minimum storage capacity (which would be $\alpha = 3$), but does provide the minimum repair bandwidth of 4.

Next, we study scenarios where the storage network comprises of a mix of active and passive nodes. While one may arguably consider all manner of combinations of the different parameters, to keep the analysis tractable and yet discern the impact of passive nodes, we next investigate two specific settings, namely when $d_1 = d_2$ and when $\beta_1 = \beta_2$.

## 4 The Case $d_1 = d_2 = d$

If the number of nodes contacted per repair is the same for both passive and actives nodes, that is $d_1 = d_2 = d$, then (3) becomes

$$\sum_{i=0}^{k-1} \min\{\alpha, (d-i)\beta_2, (d-i)\beta_1\},$$

which is minimized under the constraint that it contains exactly $k_1$ terms of the form $(d-i)\beta_1$, and $k_2$ terms of the form $(d-i)\beta_2$.

The case $\beta_1 < \beta_2$ does not make sense. Indeed, from the way (2) a passive node is modeled, $d\beta_2 = \alpha$. If we have $d\beta_1 < \alpha$, then a repaired node would have less data than the failed node, which is a contradiction.

**Corollary 4:** *If $\beta_2 \leq \beta_1$, then $(d-i)\beta_2 \leq (d-i)\beta_1$ for every $i$, and the smallest cut is*

$$\sum_{i=0}^{k_2-1} \min\{\alpha, (d-i)\beta_2\} + \sum_{i=0}^{k_1-1} \min\{\alpha, (d-k_2-i)\beta_1\},$$

*where*

$$\begin{aligned}\sum_{i=0}^{k_2-1} \min\{\alpha, (d-i)\beta_2\} &= \sum_{i=0}^{k_2-1} \min\{\alpha, \alpha - i\beta_2\} \\ &= k_2\alpha - \beta_2 \frac{k_2(k_2-1)}{2}.\end{aligned}$$

*T*: his can be shown by carrying out a term by term comparison: during the first repair, the smallest repair bandwidth, from a live node to a passive node, is $d\beta_2$ since $d\beta_2 \leq d\beta_1$. In fact, since $(d-i)\beta_2 \leq (d-i)\beta_1$, at the $(i+1)$th repair, the smallest cut always comes from $\min\{\alpha, (d-i)\beta_2\}$, and this is repeated till the $k_2$ passive nodes are exhausted. The $(k_2+1)$th node to be considered is an active node, and now the cut involves the repair bandwidth component $d\beta_1$, however the downloaded data may be redundant in terms of information with respect to what has already been accounted for from the $k_2$ live passive nodes, and thus the cut only involves $(d-k_2)\beta_1$ repair bandwidth. The process continues until $k_1$ active nodes are considered. □

This translates into the constraint

$$k_2\alpha - \beta_2 \frac{k_2(k_2-1)}{2} + \sum_{i=0}^{k_1-1} \min\{\alpha, (d-k_2-i)\beta_1\} \geq B, \tag{7}$$

on the min-flow for the repair of an object of size $B$ to be possible, and the best scenario occurs with equality. Since $d_2\beta = \alpha$, all we need to know is where $\alpha$ is located, among the values

$$(d-k_2-k_1+1)\beta_1 \leq \ldots \leq (d-k_2)\beta_1.$$

If $d_2\beta = \alpha \leq (d-k_1-k_2+1)\beta_1$, then (7) becomes

$$k_2\alpha - \beta_2 \frac{k_2(k_2-1)}{2} + k_1\alpha = B.$$

More generally, if $\alpha \in [(d-k_1-k_2+j)\beta_1, (d-k_1-k_2+j+1)\beta_1]$, $1 \leq j \leq k_1-1$, we obtain

$$B = k_2\alpha - \beta_2 \frac{k_2(k_2-1)}{2} + \sum_{i=1}^{j} (d-k_1-k_2+i)\beta_1 + (k-k_2-j)\alpha$$

or equivalently, by noticing that $\beta_2 \frac{k_2(k_2-1)}{2} = \alpha \frac{k_2(k_2-1)}{2d}$

$$\alpha = \frac{(B - j(d-k+\frac{j+1}{2})\beta_1)2d}{2d(k-j) - k_2(k_2-1)}.$$

The case $j = k_1 - 1$ provides the range $\alpha \in [(d-k_2-1)\beta_1, (d-k_2)\beta_1]$ of largest values of $\alpha$, beyond which, $\alpha$ does not affect (7) anymore. Recalling that $k = k_1 + k_2$, the range $(d-k+j)\beta_1 \leq \alpha \leq (d-k+j+1)\beta_1$ is the same as $d\beta_1 \leq \alpha - (-k+j)\beta_1 \leq (d+1)\beta_1$, that is

$$\begin{aligned} d\beta_1 &\leq \frac{B - j(d-k+\frac{j+1}{2})\beta_1}{k-j-\frac{k_2(k_2-1)}{2d}} + (k-j)\beta_1 \\ &\leq (d+1)\beta_1. \end{aligned}$$

By setting

$$g(j) = k - j - \frac{k_2(k_2-1)}{2d},$$

the above range can be expressed in terms of $d_1\beta$:

$$\frac{B}{g(j)} \frac{d}{(d+1) + \frac{j(2d-2k+j+1)}{2g(j)} - (k-j)} \leq d\beta_1$$

and

$$d\beta_1 \leq \frac{B}{g(j)} \frac{d}{d + \frac{j(2d-2k+j+1)}{2g(j)} - (k-j)}.$$

Thus in summary $\alpha = \alpha(d, k, \beta_1)$ is a piecewise linear function given by

$$\boxed{\alpha = \frac{(B - j(d-k+\frac{j+1}{2})\beta_1)2d}{2d(k-j) - k_2(k_2-1)}} \tag{8}$$

where $d\beta_1 \in [f_1(j), f_1(j-1))$, $0 \leq j \leq k_1 - 1$, and

$$f_1(j) = \frac{2Bd}{2g(j)(d+1-k+j) + j(2d-2k+j+1)}.$$

Note that $g(j-1) - g(j) = 1$, thus

$$\begin{aligned} & f_2(j-1) \\ &= \frac{2Bd}{2g(j-1)(d+1-k+j-1) + (j-1)(2d-2k+j)} \\ &= \frac{2Bd}{2(g(j)+1)(d-k+j) + (j-1)(2d-2k+j)} \\ &= \frac{2Bd}{2g(j)d + j(2d-2k+j+1) - 2g(j)(k-j)} \end{aligned}$$

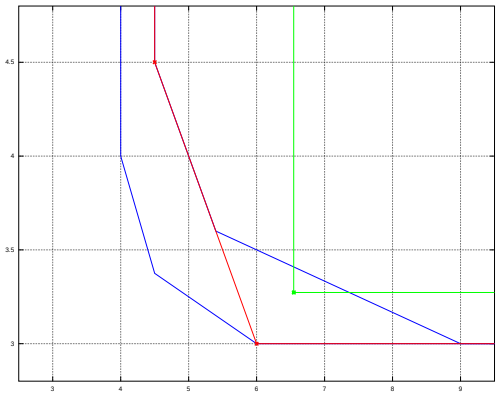


**Figure 4** On the $y$-axis, the storage $\alpha$ is shown as a function of $d\beta_1$ on the $x$-axis. The parameters are $B = 9$, $k = 4$ and $d = 4$. The lower curve (the best scenario) corresponds to the whole active case ($k_1 = 3$) also shown on Figure 3. The next curve corresponds to $k_1 = 2$ and $k_2 = 1$, which partly overlaps the 3rd curve, that of the whole active case curve for $d = 3$. The upper curve is the worst case, when $k_2 = 2$.

which is indeed the desired bound on $d\beta_1$.

As a sanity check, note that if $k_2 = 0$ (all nodes are active), then $g(j) = k - j$, and $\alpha$ matches with (4) while $f_1(j) = f(j)$ (given in (5)).

In Figure 4, we plot on the $y$-axis the storage $\alpha$ as a function of $d\beta_1$. By assumption $d\beta_2 = \alpha$. The parameters are $B = 9$, $k = 4$ and $d = 4$. The lower curve (the best scenario) corresponds to the whole active case ($k_1 = 3$) also shown on Figure 3. In the whole active case, when $d \geq k$ increases, the trade-off improves. The next curve corresponds to $k_1 = 2$ and $k_2 = 1$, which partly overlaps the whole active case curve for $d = 3$. The upper curve is the worst case, when $k_2 = 2$. We observe that the minimum value of $\alpha$ is larger when $k_2 = 2$.

From (8), the minimum value of $\alpha$ is reached with $j = 0$, namely:

$$\alpha = \frac{B2d}{2dk - k_2(k_2 - 1)}.$$

We notice that $\alpha$ takes the same value, namely $\frac{B}{k}$, which is the minimum storage overhead, when $k_2 = 0$ or $k_2 = 1$, otherwise it increases as a function of $k_2$. This is illustrated in Figure 5. This lower bound quantifies the impairment on storage overhead as a function of the number of passive nodes in the system.

## 5 The Case $\beta_1 = \beta_2 = \beta$

Suppose that the repair bandwidth is the same for both active and passive nodes. Since from (1)-(2), $d_1\beta \geq \alpha$, $d_2\beta = \alpha$, we have that $d_1 \geq d_2$, meaning that we let the active nodes

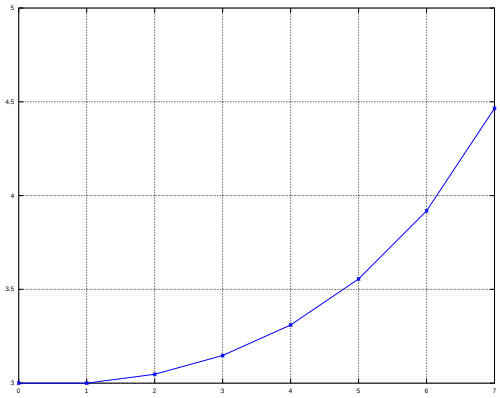

**Figure 5** The $y$-axis shows the minimum storage $\alpha$ as a function of $k_2$ on the $x$-axis. The parameters are $B = 24$, $k = d = 8$. It shows that when $k_2 = 0, 1$, the minimum storage $B/k$ is obtained, but when $k_2 \geq 2$ increases, so does the minimum storage $\alpha$.

contact more live nodes. This makes sense since they can process more information. Recall that by assumption $d_1, d_2 \geq k$.

**Corollary 5:** *If $d_2 \leq d_1$, then $(d_2 - i)\beta \leq (d_1 - i)\beta$ for every $i$, and the smallest cut is*

$$k_2\alpha - \beta\frac{k_2(k_2-1)}{2} + \sum_{i=0}^{k_1-1} \min\{\alpha, (d_1 - k_2 - i)\beta\}.$$

*T*: he key point is that we have a term by term comparison, after which the proof is similar to that of Corollary 4. □

This translates into the following constraint on the min-flow for the repair of an object of size $B$ to be possible:

$$k_2\alpha - \beta\frac{k_2(k_2-1)}{2} + \sum_{i=0}^{k_1-1} \min\{\alpha, (d_1 - k_2 - i)\beta\} \geq B, \tag{9}$$

and the best scenario occurs with equality. Since $d_2\beta = \alpha$, all we need to know is where $\alpha$ is located, among the values

$$(d_1 - k_2 - k_1 + 1)\beta \leq \ldots \leq (d_1 - k_2)\beta.$$

If $d_2\beta = \alpha \leq (d_1 - k_1 - k_2 + 1)\beta$, then (9) becomes

$$k_2\alpha - \beta\frac{k_2(k_2-1)}{2} + k_1\alpha = B.$$

More generally, if $\alpha \in [(d_1 - k_1 - k_2 + j)\beta, (d_1 - k_1 - k_2 + j + 1)\beta]$, $1 \leq j \leq k_1 - 1$, we obtain

$$k_2\alpha - \beta\frac{k_2(k_2-1)}{2} + \sum_{i=1}^{j}(d_1 - k_1 - k_2 + i)\beta + (k - k_2 - j)\alpha = B$$

or equivalently

$$\alpha = \frac{B + [\frac{k_2(k_2-1)}{2} - j(d_1 - k_1 - k_2 + \frac{j+1}{2})]\beta}{k - j}.$$

The case $j = k_1 - 1$ in the above expression also provides the largest possible value of $\alpha$, namely $\alpha = (d_1 - k_2)\beta$. Above this value, $\alpha$ does not appear in (9) anymore. Recalling that $k = k_1 + k_2$, the range $(d_1 - k + j)\beta \leq \alpha \leq (d_1 - k + j + 1)\beta$ is the same as $d_1\beta \leq \alpha - (-k + j)\beta \leq (d_1 + 1)\beta$, that is

$$\begin{aligned} d_1\beta &\leq \frac{B + [\frac{k_2(k_2-1)}{2} - j(d_1 - k + \frac{j+1}{2}) + (k-j)^2]\beta}{k - j} \\ &\leq (d_1 + 1)\beta \end{aligned}$$

which expressed in terms of $d_1\beta$ becomes

$$\frac{Bd_1}{(d_1 + 1 - k + j)(k - j) - \frac{k_2(k_2-1)}{2} + j(d_1 - k + \frac{j+1}{2})} \leq d_1\beta,$$

and

$$d_1\beta \leq \frac{Bd_1}{(d_1 - k + j)(k - j) - \frac{k_2(k_2-1)}{2} + j(d_1 - k + \frac{j+1}{2})}.$$

Thus $\alpha = \alpha(d_1, k_1, k_2, \beta)$ is a piecewise linear function, given by

$$\boxed{\alpha = \frac{B + [\frac{k_2(k_2-1)}{2} - j(d_1 - k + \frac{j+1}{2})]\beta}{k - j}}$$

for $d_1\beta \in [f_2(j), f_2(j-1))$, $0 \leq j \leq k_1 - 1$, and

$$f_2(j) = \tfrac{2Bd_1}{2(d_1+1-k+j)(k-j) - k_2(k_2-1) + j(2d_1 - 2k + j + 1)}. \tag{10}$$

As sanity checks (again), take $k_2 = 0$ (that is $k_1 = k$, $d_1 = d$), in which case all nodes are active. Then $\alpha$ simplifies to

$$\alpha = \frac{B - j(d - k + \frac{j+1}{2})\beta}{k - j},\ d_1\beta \in [f_2(j), f_2(j-1))$$

which indeed matches with (4), and (10) reduces to (5). If $k_1 =$ (that is $k_2 = k$, $d_2 = d$), when all nodes are passive, we have that $j = 0$, and we find that

$$\alpha = \frac{B + [\frac{k(k-1)}{2}]\beta}{k}$$

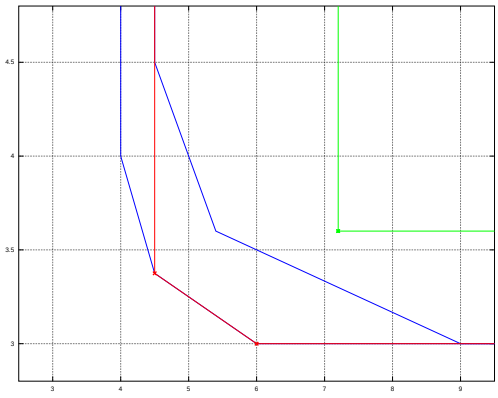


**Figure 6** The storage $\alpha$ is shown on the $y$-axis as a function of $d_1\beta$ on the $x$-axis. The two lower curves are the whole active case with $d = 4$ and the case of only one passive node ($k_1 = 2$). The next curve is the whole active case with $d = 3$, and the worst case is the upper curve, for $k_2 = 2$ passive nodes.

which is indeed the storage for the passive case.

In Figure 6, we display the storage $\alpha$ as a function of $d_1\beta$. We still have that $d_2\beta = \alpha$. The curves for the whole active case for respectively $d = 3$ and $d = 4$ are shown for comparison. The parameters are $B = 9$, $k = 3$ and $d_1 = 4$, $d_2 \leq 3$. The curve for $k_1 = 2$ and $k_2 = 1$ partly overlaps that of $d = 4$ for the whole active case. There is a big penalty for having $k_2 = 2$, especially in terms of minimum storage, as already observed in the previous section.

## 6 Concluding Remarks

In this paper, we proposed a min-cut analysis to study the repair of erasure coded data (using network coding techniques) in a distributed storage system comprising a mix of passive (with computational ability) as well as active nodes. We investigated resulting bounds on repair bandwidth as a function of the storage overhead for different specific configurations of active and passive nodes. This analysis provided a mechanism to quantify the overheads incurred in employing network coding techniques for repairing lost redundancy in the presence of passive nodes. Our results show that starting from a system with only active nodes, the introduction of (very) few passive nodes retains a low storage capacity, but generates an increase in repair bandwidth, while both the storage capacity and the repair bandwidth have to increase when the number of passive nodes increases.

This paper is a first step, and opens several interesting directions of future research. The bounds derived in this paper provide necessary condition, thus one open question is whether the bounds are tight. Generalizing the model for further exploration of the parameter space is another possibility, as also consideration of collaborative repair scenarios. In addition to these fundamental aspects, the immediate practical question is the design of codes which can operate in such mixed network. We provided a toy example code in this paper as a

proof-of-concept in addition to guide the narrative, but more general code constructions are necessary, and constitutes our immediate future work.

## Acknowledgement

The work of F. Oggier and A. Datta is supported by the MoE Tier-2 grant “eCODE: Erasure Codes for Datacenter Environments".

## References

[1] Apache.org. HDFS. `http://hadoop.apache.org/hdfs/`.

[2] 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.

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

[4] 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.

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

[6] 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.

[7] 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.

[8] A-M. Kermarrec, N. LeScouarnec, and G. Straub. Repairing multiple failures with coordinated and adaptive regenerating codes. In *NetCod*, 2011.

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

[10] F. Oggier and A. Datta. Coding techniques for repairability in networked distributed storage systems. *Foundations and Trends in Communications and Information Theory*, 9(4), 2012.

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

[12] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran. Explicit construction of optimal exact regenerating codes for distributed storage. In *Allerton Conference*, September 2009.

[13] 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.

[14] K. W. Shum. Cooperative regenerating codes for distributed storage systems. In *ICC*, 2011.