

Hierarchical codes: A flexible trade-off for erasure codes in peer-to-peer storage systems

Alessandro Duminuco · Ernst W. Biersack

Received: 28 November 2008 / Accepted: 23 March 2009
© Springer Science + Business Media, LLC 2009

Abstract Redundancy is the basic technique to provide reliability in storage systems consisting of multiple components. A redundancy scheme defines how the redundant data are produced and maintained. The simplest redundancy scheme is replication, which however suffers from storage inefficiency. Another approach is erasure coding, which provides the same level of reliability as replication using a significantly smaller amount of storage. When redundant data are lost, they need to be replaced. While replacing replicated data consists in a simple copy, it becomes a complex operation with erasure codes: new data are produced performing a coding over some other available data. The amount of data to be read and coded is d times larger than the amount of data produced, where d , called repair degree, is larger than 1 and depends on the structure of the code. This implies that coding has a larger computational and I/O cost, which, for distributed storage systems, translates into increased network traffic. Participants of Peer-to-Peer systems often have ample storage and CPU power, but their network bandwidth may be limited. For these reasons existing coding techniques are not suitable for P2P storage. This work explores the design space between replication and the existing erasure codes. We propose and evaluate a new class of erasure codes, called Hierarchical Codes, which allows to reduce the network traffic due to maintenance without losing the benefits given by traditional erasure codes.

Keywords Peer-to-peer · Storage · Erasure codes · Reliability · Dependability

1 Introduction

P2P (Peer-to-Peer) systems have received a lot of attention in recent years. In particular, the research community has shown an increasing interest in the use of P2P systems for file storage [2, 4, 8, 14]. This application can be very attractive for two main reasons: (i) centralized solutions are expensive (ii) common PCs are equipped with high-capacity local disks, that are often underutilized.

The main challenge in designing storage systems is to guarantee the persistence of the stored data. This is non-trivial because storage devices are not totally reliable: they may face failures, data corruption or accidental data losses. The proposed solutions move in two complementary directions: increasing the device reliability and adding redundancy to data.

In a peer-based approach, the first direction is not feasible since existing hardware is used and a proper redundancy scheme is the only tool in the hands of the system designer.

The simplest approach to redundancy is replication. The drawback of such scheme is its inefficiency in terms of storage. Another approach is erasure coding, which is able to provide the same level of reliability with much lower storage requirements [12, 15]. The price to pay for this is a higher maintenance cost as we explain below.

When data are lost, a maintenance operation, called *repair*, is needed to replace them. The replacement of replicated data is trivial and consists in making a copy.

A. Duminuco (✉) · E. W. Biersack
EURECOM, 2229 Route des Crètes,
Sophia Antipolis, France
e-mail: duminuco@eurecom.fr

E. W. Biersack
e-mail: biersack@eurecom.fr

For erasure codes, instead, every bit of new data is the result of a coding operation over several other bits of data. This introduces additional computational cost, to perform the coding, and additional I/O cost needed to retrieve the *bits* to be coded, which in distributed systems translates into *network traffic*.

Traditional storage systems can easily handle this additional costs. RAID systems, for example, need a very small amount of repair operations and in any case they are equipped with dedicated processing and network resources, which are dimensioned according to their needs.

In a P2P storage system, the number of repairs can be very high and, while such a system can rely on large storage and processing resources provided by participating peers, the system must cope with limited network resources. This makes coding unattractive for P2P storage, since it has been conceived with a different cost model in mind.

The coding schemes proposed in literature strive to increase the storage efficiency while ignoring the other costs. We propose a new class of codes, called *Hierarchical Codes*, which introduce a flexible trade-off between replication and traditional erasure codes, reducing the maintenance cost without sacrificing storage efficiency.

In Section 3.1 we formalize the efficiency metrics used and in Section 3.2 we perform a cost analysis for the main existing redundancy schemes and in particular for linear erasure codes. In Section 4 we present Hierarchical Codes. Finally, in Section 5 we evaluate Hierarchical Codes by means of experiments.

2 Related works

Various redundancy schemes for P2P storage systems have been studied. Many papers focus on the the comparison between replication and coding. In [15] it is shown how, given the same amount of storage space, erasure codes can give big improvements in terms of data durability as compared to replication. Rodriguez [12] also compares erasure codes and replication taking into account the peer behavior and the maintenance process and arrives to the conclusion that in some cases the advantage of coding may be not worth its disadvantages. Rodriguez is also the first one to point out that the repair cost required by coding may be prohibitive in peer-to-peer environments. He then proposes a *hybrid* scheme that uses both, coding and replication. However, this scheme increases the complexity and loses most of the storage efficiency offered by coding. This point has been well explained by Dimakis et al. [5],

who proposes a new class of codes, called *Regenerating Codes*, which are able to reduce the repair traffic while consuming slightly more storage space. Also, the cost of computation may be significant [6]. Our work is inspired by the existing solutions and proposes an alternative solution, called *Hierarchical Codes*, which does not sacrifice the storage efficiency and does not increase the computation requirements with respect of traditional erasure codes. Both codes, Regenerating codes and Hierarchical Codes, adopt some of the concepts and the tools presented in the early literature about Linear Network Coding [3, 9].

As we will show in details in Section 4, the core idea of Hierarchical Codes is to allow encoded blocks to be a linear combination of a *limited* number of original fragments. This is, to some extent, similar to LDPC (Low Density Parity Checks) family codes.¹ In this family of codes each encoded symbol is produced performing the *XOR* operation on a number of original symbols, which in average is much smaller than their total number. In spite of this similarity, there are profound differences between LDPC and Hierarchical Codes. The most important difference is that LDPC codes are conceived for data transmission on lossy channels and for this reason they do not contemplate the concept of repair, which is essential in distributed storage systems. This means that they do not provide a way to produce a new encoded block using other encoded blocks preserving the same level of reliability of the code, while this is the main concern of Hierarchical Codes.

3 Redundancy schemes for P2P storage

3.1 Efficiency metrics

The measure of the efficiency consists in comparing the benefits provided with the costs required.

In the domain of redundant storage, the benefit is the reliability of the data storage in spite of failures of the storage components. In the P2P storage systems, failures are represented by temporary disconnections, abandons, device errors etc. The ability of a redundancy scheme to be resilient to such failures is usually measured as the probability of a correct reconstruction of a stored object. Note that this measure is not absolute, but it is conditioned by the peer behavior: one of the most important factors is the probability of having concurrent failures. For this reason, the *reliability* of

¹There are several codes derived from LDPC codes, such as Tornado-Codes, LT-Codes etc., see [10] for a brief survey.

a redundancy scheme is measured as the number of concurrent losses that it can sustain without compromising the data. More formally, one can express this property as the probability of data loss (*failure*) given that l concurrent losses occurred: $P(\text{failure}|l)$.

The description of the costs is more complex. To perform a complete analysis, we need to consider separately the two main activities involved in a storage system:

Storage The core activity of a storage system consists in the initial insertion of the data along with their redundancy. The cost of the redundancy scheme, in this phase, is merely the absolute amount of storage space consumed. To abstract from the amount of data, it is measured as the ratio between the size of the stored data $|S|$ and the size of the original data $|O|$. This cost can be referred to as *Redundancy Factor* and denoted as $\beta = |S|/|O|$.

Maintenance During the lifetime of a P2P storage system, permanent failures occur. Whenever this happens part of the redundant data is lost and the chances of losing the original data increase. If nothing is done to compensate these losses, sooner or later the durability will be not guaranteed anymore. The maintenance consists in refurbishing the redundant data when they are lost. This operation is performed reading the *available* data blocks and producing a new one. The reading operation has a cost, which in a distributed storage system translates into network traffic, whose volume depends on the redundancy scheme adopted but also on lots of other factors, such as the peer behavior, the repair policy, the coordination algorithms etc. To evaluate only the contribution of the redundancy scheme, we measure the amount of data read with respect of the amount of new redundant data created. In other words, once the system has decided that a new encoded bit needs to be created, we measure how many available bits the scheme has to read. This cost can be referred to as *Repair Degree* and denoted as d .

3.2 Efficiency analysis

In this section we describe some of the most representative examples of redundancy schemes and illustrate their efficiency in terms of the metrics described in the previous section.

Replication Replication is the most straightforward way to add redundancy. Its basic version consists in creating multiple copies of the object to store. The analysis of such a scheme is very simple. Let us assume that R replicas of the original object are stored on

different peers. The number of losses that the system can support is $R - 1$. In a formal way the probability of losing the object conditioned by the probability of having a given number of concurrent losses is:

$$P(\text{failure}|l) = \begin{cases} 0 & l < R \\ 1 & l = R \end{cases}$$

The redundancy factor is $\beta = R$, while the repair degree is $d = 1$, since the reconstruction of a new element corresponds to a simple copy of one replica.

Block Replication In a P2P system, a key strategy to make data survive failures is to spread them across different locations, exploiting the *diversity* of peers. For this reason there is a more complex way to do replication.

Consider an object O and split it in k fragments, then create R replicas for every fragment and place every single replica on a different peer. Now the number of peers involved is $k \times R$, the redundancy factor is still $\beta = R$ and the repair degree is still $d = 1$. The analysis of the reliability of this scheme is more complex, since there is not a single number that says how many peers we can lose without compromising the object as the survival of the object depends on which particular peers fail. In a very fortunate case we can lose all the redundancy, i.e. $k \times (R - 1)$ blocks, and still be able to retrieve the original object, in the opposite case if all the replicas of a single block disappear, the object is lost when as few as R blocks are lost. The probabilistic expression of the reliability is very helpful in this case. Exploring exhaustively all the possible combinations of losses for the case $k = 8$ and $R = 3$, we obtain the solid curve in Fig. 1.

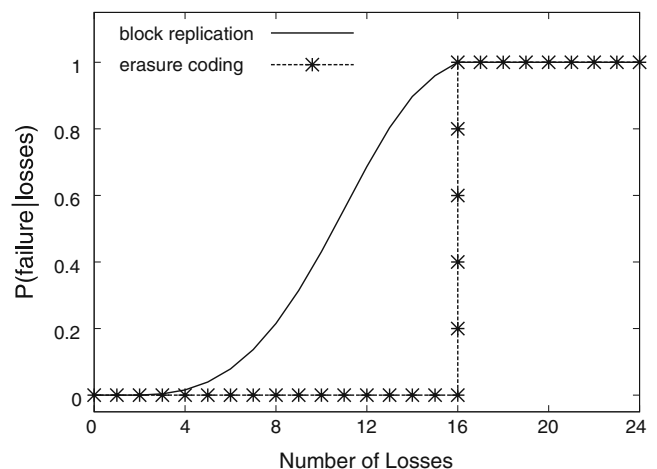


Fig. 1 Block replication scheme compared to erasure codes with $k = 8$ and $R = 3$. $P(\text{failure}|l)$ as function of the number of concurrent losses l

Erasure Codes A generic erasure (k,h)-code can be described as follows. Consider an object O and split it in k fragments, then process these fragments producing $k + h$ encoded blocks such that any k of them are sufficient to reconstruct the original fragments. In a P2P storage system, each of this encoded blocks is stored on a distinct peer. The number of losses that this scheme can sustain is h , while the redundancy factor is $\beta = (k + h)/k$. If, for example, we use $k = 8$ and $h = 16$ we will obtain $\beta = 3$. This configuration is comparable with the example proposed for block replication, since k and β are the same. The starred curve in Fig. 1 shows that the reliability provided by coding is much higher: it can sustain always until 16 losses, while replication is able to do that only in a very small percentage of the cases. The price to pay for this increased storage efficiency resides in the repair cost. As we will show in the next section, the existing coding schemes (with the characteristics described) require a repair degree of $d = k$.

As already mentioned, in P2P storage systems, a low *repair degree* d is crucial. Today, the high repair degree is the reason why most of the P2P storage systems use replication, preferring to pay an increased storage cost rather than an increased communication cost.

We aim at developing a new class of codes, which find a place between erasure codes and replication, offering a flexible trade-off in terms of storage requirements, average repair degree, and reliability.

3.3 Efficiency of linear codes

In this section we detail the basic concepts behind the functioning of *linear codes*, which are a specific implementation of erasure codes. We present also a formal tool to analyze their reliability and repair cost. These concepts will be used when we introduce our Hierarchical Codes.

The parity check can be considered as the simplest implementation of a (k,1)-code: consider k bits and build an additional bit applying the *XOR* operator to all the other bits. The *XOR* operation can be extended to all the bits in a fragment. If we denote as f_i the sequence of bits in the original fragments, and with b_i the sequence of bits in the encoded blocks, we can describe the parity check as follows:

$$b_i = \begin{cases} f_i & i \leq k \\ f_1 \otimes f_2 \otimes \dots \otimes f_k & i = k + 1 \end{cases}$$

It is clear that any k encoded blocks are sufficient to build the missing one, applying again the *XOR* operator.

The *XOR* operation can be interpreted as a linear combination of all the original fragments in the domain of Galois Field² with size 2, denoted as $GF(2)$. In this field there exists only one possible linear combination among all blocks and this is the reason for which only one additional block can be built. In erasure codes, a larger field size is used to be able to build h additional blocks.

Consider a Galois Field $GF(2^q)$, where the elements of such a field can be expressed by q -bit words. This means that every original fragment and every block can be interpreted as a sequence of words in $GF(2^q)$. Let us denote as f_i and b_i the words belonging respectively to the i^{th} fragment and the i^{th} encoded block. A linear code can be built using the following linear operations in $GF(2^q)$:

$$b_i = \begin{cases} f_i & i \leq k \\ \sum_{j=1}^k c_{i,j} f_j & k < i \leq k + h, c_{i,j} \in GF(2^q) \end{cases} \quad (E1)$$

Assuming for simplicity that all the fragments are composed by a single word, we can introduce the following vectors and matrices, all composed by elements in $GF(2^q)$:

$F_{k,1}$	Vector of original fragments.
$B_{k+h,1}$	Vector of encoded blocks.
$I_{k,k}$	Identity matrix.
$C_{h,k}$	Coefficient matrix.

Using these matrices we can give an alternative expression of the code:

$$B = \begin{bmatrix} I \\ C \end{bmatrix} F = C' F$$

If the matrix C is such that any sub-matrix S built using k rows from C' is invertible, then the original fragments can be always reconstructed by $F = S^{-1} B_S$, where B_S is the k -long subvector of B , corresponding to the coefficients chosen in S . If this property is satisfied, the code obtained is a (k,h)-code.

Many choices are possible for the coefficient matrix, and consequently there exist multiple implementations of this class of codes. One of the most prominent are *Reed-Solomon* codes [11], which define the matrix C as a $h \times k$ Vandermonde matrix, i.e. $c_{i,j} = j^{i-1}$. For Reed-Solomon codes as well as for all the codes that fix a specific coefficient matrix, the repair of a lost block requires first the reconstruction of all the original

²A Galois Field or Finite Field is an algebraic structure with a finite number of elements. The main property of a Galois Field is that all the operations applied to its elements results in an element within field itself.

fragments, which are then recombined accordingly to the coefficient row that corresponds to the lost block. This explains why the repair degree is $d = k$.

Another approach is to build the matrix C' choosing randomly the coefficients in the Galois Field.³ This class of codes are called *Random Linear Codes*. It is shown in [1] that a $k \times k$ random matrix S in $\text{GF}(2^q)$ is invertible with a probability that depends only on the field size. By increasing the field size, this probability can be pushed arbitrarily close to 1. A common choice for the parameter q is $q \geq 16$, in which case the probability can be considered to be 1 for all practical purposes. This means that any $k \times k$ sub-matrix of C' is invertible and that the property of a (k,h) -code is given.

The repair of a lost block could be done like in Reed-Solomon codes, i.e. first reconstructing the original fragments and then combining them again. In this case the repair degree would be again $d = k$. However, with random linear codes we can do better: In fact the reconstruction of original fragments is *not necessary* and the result is indeed equivalent as to when the k encoded blocks are combined directly using random coefficients.

One may be tempted, in this case, to use less than k blocks, reducing in this way the repair degree. However, only a repair degree of $d = k$ is able to preserve the properties of the code. In particular for a repair degree of $d < k$, there will be sets of k encoded blocks that are not sufficient to reconstruct the original k fragments. This result can be derived from the literature about network coding [3, 9] and its application to distributed storage systems [5]. We will reformulate here some of the results in a slightly different but equivalent way, which will help us in deriving Hierarchical Codes.

Let us introduce the concept of an *Information Flow Graph*, which represents the evolution of the stored data across time. In particular, each node represents a block of data at a specific point in time t . The time evolves in discrete steps and every step corresponds to one or more losses and repairs. At the time $t = 0$ the graph is populated only by k source nodes representing the k original fragments denoted as $F = f_1, f_2, \dots, f_k$. At time $t = 1$, the graph consists of $k + h$ nodes that represent the $k + h$ encoded blocks initially inserted in the storage system and denoted as $B_1 = b_{1,1}, b_{2,1}, \dots, b_{k+h,1}$. The graph at time $t = 1$ is referred to as the *code graph*. At $t > 1$, the graph consists of additional $k + h$ nodes that represent the $k + h$ encoded blocks present in the system at time t , which are

denoted as $B_t = b_{1,t}, \dots, b_{k+h,t}$. Connections between nodes are only possible among nodes of consecutive time steps and are always oriented from t to $t - 1$. The possible connections and their semantics in the storage system are:

1. A generic node $b_{1,1}$ at time step 1 is connected to one or more original fragments, denoted as $R(b_{1,1})$. These connections are determined by the equations of the code used and for this reason the graph obtained is called *code graph*. In particular $R(b_{1,1})$ is the set of fragments linearly combined to produce $b_{1,1}$.
2. A generic block $b_{i,t-1}$ in B_{t-1} can be connected to the node $b_{i,t}$. In this case node $b_{i,t}$ *must not be connected to any other nodes in B_{t-1}* . This means that the block b_i has survived at time $t - 1$.
3. Alternatively, a generic block $b_{i,t-1}$ is not connected to any node in the following step. In this case node $b_{i,t}$ is connected to d nodes B_{t-1}^d in B_{t-1} , where $b_{i,t-1} \notin B_{t-1}^d$. This means that block b_i has been lost at time $t - 1$ and it has been repaired linearly combining the d blocks in B_{t-1}^d .

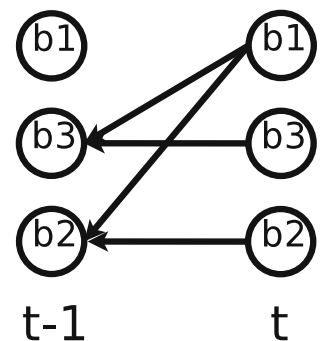
See the example in Fig. 2.

The *Information Flow Graph* we presented is a variant of the one proposed in [5]. This allows us to formulate the following lemma, which derives from Proposition 1 in [5]:

Lemma 1 *A selection of k nodes $B_i^k \subseteq B_t$, is sufficient to reconstruct the original fragments (with a probability that depends only on the size of the Galois Field in which the random coefficients are drawn), only if it is possible to find k disjoint paths from the k nodes in B_i^k to the k source nodes in F .*

The *disjoint paths* condition is obviously related to the choice of the repair degree d . The following proposition holds:

Fig. 2 Example of one step of an *Information Flow Graph*. Blocks b_2 and b_3 have survived at time $t - 1$. Block b_1 has been lost at time $t - 1$ and has been repaired at time t combining the blocks b_2 and b_3



³Note that replacing the identity matrix with random coefficients transforms the code from a systematic one to an unsystematic one.

Proposition 1 At any time t , any possible selection of k nodes B_t^k is sufficient to reconstruct the original fragments only if the disjoint paths condition is provided at time step $t = 1$ (by the code graph) and the repair degree is $d \geq k$.

See the proof A.2 in the Appendix A. The Proposition 1 requires that the *disjoint paths* condition be provided by the *code graph*. In that case this condition can be interpreted as the existence of a *perfect matching* between any selection B_1^k and the k source nodes in F . A *Random linear code* clearly provides this condition, since by design any node in B_1 is connected to all the source nodes in F .

4 Hierarchical codes

The previous section showed that for *traditional* linear erasure codes the repair degree d cannot be smaller than k . Indeed, if $d < k$, there will be selections of k encoded blocks that are *not* sufficient to reconstruct the original fragments. From this point of view, the block replication scheme presented in Section 3.2 can be considered as a limit case of a $(k, (R-1)k)$ -code in which the repair degree is chosen to be $d = 1$. In this case, only a small subset of all possible choices of k encoded blocks (replicated fragments) is able to reconstruct the original object, which may result in a lower reconstruction probability for a given number of losses as we saw in Fig. 1.

Our intuition is that $d = 1$, which corresponds to block replication, and $d = k$, which corresponds to a traditional erasure code, are two limit cases. We believe that there is an interesting design space between these two limits that can be explored to find a better trade-off between storage efficiency and repair degree.

The naïve approach of using $d < k$ in random linear codes poses two main difficulties: (i) there is no easy way to analyze the final reliability of the code, as we did in block replication; (ii) there is no trivial policy for choosing the d blocks (to be combined) that are able to prevent a degradation of the reliability of the code through the maintenance process. Note that in block replication there is such a way: replace a lost replica with a copy of an identical one.

We propose a new coding scheme to overcome these difficulties, which we call *Hierarchical Codes*. A general instance of such a code can be generated through its *code graph* built according to the following procedure:

1. Choose two parameters k_0 and h_0 and build a (k_0, h_0) -code using the Eq. E1 with the coefficients

$c_{i,j}$ chosen randomly in $GF(2^q)$. If we set $k_0 = 2$ and $h_0 = 1$ we obtain the *code graph* in Fig. 3a.

The generated blocks constitute a group denoted as $G_{d_0,1}$, where $d_0 = k_0$ is the degree used to generate the blocks and it is called *combination degree*. In Fig. 3a, $d_0 = 2$.

2. Choose two parameters g_1 and h_1 . Replicate the group structure $G_{d_0,1}$ for g_1 times to obtain g_1 groups denoted as $G_{d_0,1} \dots G_{d_0,g_1}$. Then add other h_1 encoded blocks, obtained combining (with random coefficients) all the existing $g_1 k_0$ original fragments F . This corresponds to a combination degree $d_1 = g_1 k_0 = g_1 d_0$. If we set $g_1 = 2$ and $h_1 = 1$ we obtain the *code graph* in Fig. 3b.

All the blocks constitute a group denoted as $G_{d_1,1}$, which corresponds to a hierarchical (d_1, H_1) -code, where $H_1 = g_1 h_0 + h_1$. The example in Fig. 3b is a hierarchical $(4, 3)$ -code.

3. The previous step can be repeated several times, adding levels to the code. In the generic step s , choose two parameters g_s and h_s . Replicate the structure of the group $G_{d_{s-1},1}$ for g_s times. Then add other h_s encoded blocks, obtained combining all the existing original fragments, which corresponds to a degree $d_s = g_s d_{s-1}$. All the blocks constitute a group denoted as $G_{d_s,1}$, which corresponds to a hierarchical (d_s, H_s) -code, where $H_s = g_s H_{s-1} + h_s$.

The redundancy factor β of a generic hierarchical (k, h) -code does not change with respect of a traditional erasure code: $\beta = (k + h)/(k)$. The other metrics are more complex.

Reliability The analysis of the reliability consists, as usual, in computing the probabilities $P(\text{failure}|l)$. These

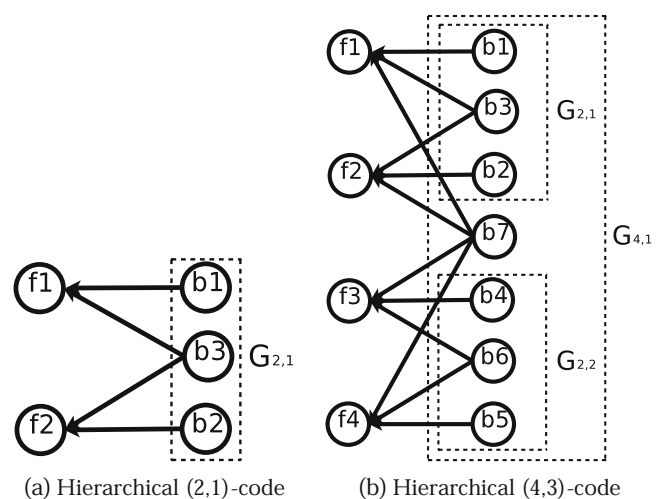


Fig. 3 Samples of code graphs for hierarchical codes (a, b)

probabilities can be computed if we know what sets of k encoded blocks are able to reconstruct the original fragments. Using Lemma 1 applied to the code graph we can state the following:

Proposition 2 Consider B^k , a set of k blocks in the code graph of a hierarchical (k, h) -code.

If the nodes in B^k are chosen fulfilling the following condition:

$$|G_{d,i} \cap B^k| \leq d \quad \forall G_{d,i} \text{ belonging to the code} \quad (\text{C2})$$

which means that in B^k there can be a maximum of d blocks chosen from any group $G_{d,i}$,

Then the nodes in B^k are sufficient to reconstruct the original fragments.

See the proof A.3 in the Appendix A. In the hierarchical $(4,3)$ -code in Fig. 3b, the condition (C2) means that no more than 2 blocks can be chosen from $G_{2,1}$, no more than 2 blocks can be chosen from $G_{2,2}$ and no more than 4 blocks can be chosen from $G_{4,1}$.⁴

Using Proposition 2 we can compute the generic probability $P(\text{failure}|l)$, exploring all the possible configurations of the losses and check in each case if there is still a possible choice of blocks that allows reconstruction, as explained in Appendix B.

Repair Degree In the case of Hierarchical Codes, as in the block replication, there does not exist a single number that expresses the repair degree required. In particular, the repair degree required changes accordingly to which block needs to be repaired and which blocks are still alive. For each situation we would like to know which is the right choice to prevent the code from degrading, i.e. to preserve the guarantees provided by the code before maintenance, as described in the previous paragraph.

We can use again the Lemma 1 to formulate:

Proposition 3 Consider an Information Flow Graph of a hierarchical code at time step t . Consider a node b repaired at time step t . Denote as $G(b)$ the hierarchy of groups that contains b and as $R(b)$ the set of nodes in B_{t-1} that have been combined to repair b .

If $\forall t$ and $\forall b$, $R(b)$ fulfills the following conditions:

$$|G_{d,i} \cap R(b)| \leq d \quad \forall G_{d,i} \text{ belonging to the code} \quad (\text{C3})$$

and

$$\exists G_{d,i} \in G(b) : R(b) \subseteq G_{d,i}, |R(b)| = d \quad (\text{C4})$$

where, (i) condition (C3) means that in the set of blocks combined $R(b)$ there can be a maximum of d blocks chosen from any group $G_{d,i}$ and (ii) condition (C4) means that there must exist a group in the hierarchy $G(b)$ that contains all the combined blocks and that their quantity has to be equal to the combination degree used in that group.

Then the code does not degrade, i.e. preserves the properties of the code graph expressed in Proposition 2.

See the proof A.4 in the Appendix A. For the hierarchical $(4,3)$ -code in Fig. 3b, this means that block b_1 can be repaired in one of the following two ways:

1. Using other 2 blocks belonging to the same group $G_{2,1}$, i.e. blocks b_2 and b_3 .
2. Using other 4 blocks belonging to the whole group $G_{4,2}$, paying attention not to pick more than two blocks from the group $G_{2,2}$, for example blocks b_3 , b_7 , b_4 , and b_6 .

When a repair is performed, according to the block that needs to be repaired, multiple repair degrees are allowed. The repair degree that is actually used will depend on the blocks that are available on the moment of the repair.⁵

Using the Proposition 3 and exploring all the possible combination of losses, we can compute the probability $P(d|l)$. The procedure to compute this probability closely resembles the procedure to compute the failure probability $P(\text{failure}|l)$ explained in Appendix B. $P(d|l)$ indicates what is the probability that, if we have l concurrent losses, the repair of a block, in the *worst case*, requires a degree d . Note that *worst case* means that among the l blocks that we could repair, we decided to repair the one that requires the highest repair degree.

Note that the *worst case formulation* of $P(d|l)$ is quite pessimistic. In the reality, the particular repair performed depends on the repair policy and its repair degree can be lower than d .

We collected the results obtained for the hierarchical $(4,3)$ -code in Fig. 3b in the following table:

	l (losses)		
	1	2	3
$P(d = 2 l)$	0.86	0.42	0
$P(d = 4 l)$	0.14	0.58	0.77
$P(\text{failure} l)$	0	0	0.23

⁴This last constraint is unnecessary, since $G_{4,1}$ represents in this case the whole code.

⁵In the example of Fig. 3b, if b_1 needs to be repaired and either b_2 or b_3 is not available, the repair degree must be $d = 4$.

The first two rows show the repair degree probability, while the last row shows $P(\text{failure}|l)$. This last row represents the cases in which the original fragments cannot be reconstructed. Note that these cases correspond also to the cases in which there is at least one block that cannot be repaired. For these last cases, thus, the failure probability replaces the probability $P(d|l)$, since the repair in the *worst case* cannot be performed. This is also the reason for which the values in each column sum up to 1. The table covers up to 3 losses, because for a higher number of losses it is clear that repairs are never possible and the failure probability is 1.

In Fig. 4a, the same probabilities are graphically represented for a hierarchical code (64,64)-code, built using 6 levels and setting $k_0 = 2$, $g_s = 2$ and $h_s = 1$ for all the levels, except for the last level where $h_5 = 2$. Every bar in the plot corresponds to a column in the

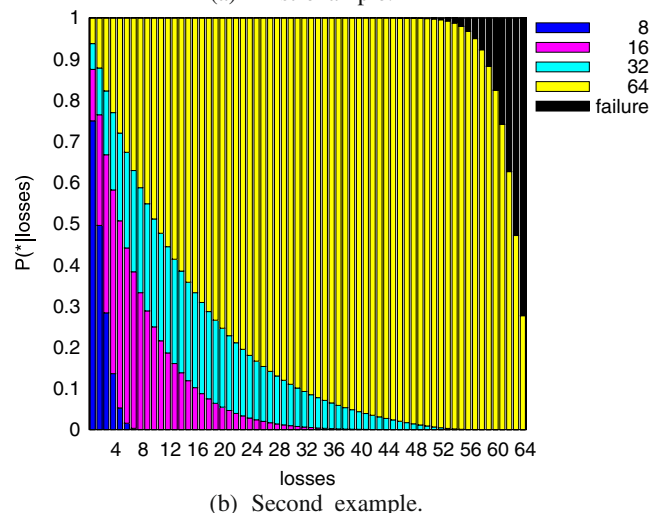
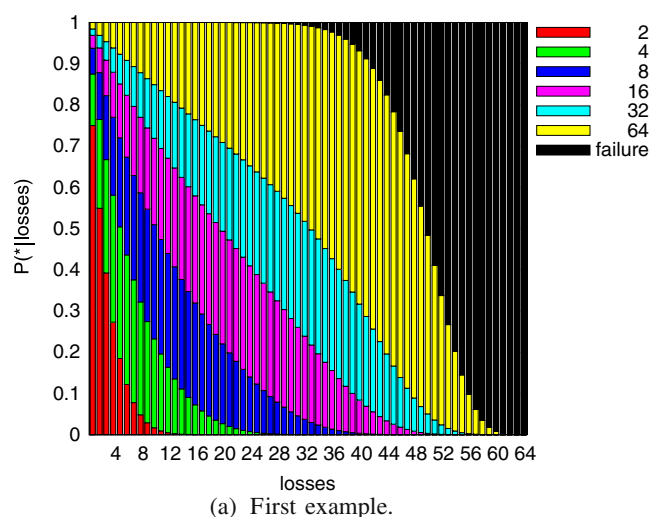


Fig. 4 Examples of Hierarchical (64,64)-codes. $P(d|l)$ and $P(\text{failure}|l)$ as function of the number of concurrent losses l (a, b)

table, while the heights of the sections in a bar represent the probabilities of repair degree or failure given the corresponding number of losses.

This figure nicely shows the properties of *Hierarchical Codes*. They are able to reduce the repair cost significantly: in a traditional (64,64)-code, the repair degree is always 64, while in this hierarchical (64,64)-code, it varies from 2 to 64. At a first look, the price to pay for this advantage seems to be reduced reliability; indeed a traditional (64,64)-code does never fail for fewer than 64 losses, while the hierarchical code may fail even for as low as 32 losses. However, by adjusting the repair policy, as we will explain in next section, one can achieve the same reliability.

We believe that Hierarchical Codes give a new possibility to system designers to determine the right trade-off between costs and benefits with respect to the characteristics of the environment in which the system is going to operate. Note that different choices of the parameters $\{k_0, g_s, h_s\}$ produce different codes with the same level of redundancy, but with a different trade-off between reliability and repair degree.

In this sense, the configuration we proposed in Fig. 4a is just one of the many instances of Hierarchical (64,64)-codes. For some environments other choices of the parameters might be better. For example, if it is not acceptable to have a non-zero failure probability for 32 losses, one can choose the alternative configuration depicted in Fig. 4b. This configuration is built in 4 levels, setting $k_0 = 8$, $g_s = 2$ and $h_s = 4$ for all the levels, except for the last level where $h_3 = 8$. The histogram in Fig. 4b shows how failures occur for higher values of losses as compared to Fig. 4a. However, there is a price to pay in terms of a higher repair cost: the repair degree varies from 8 to 64 and the region corresponding to a repair degree of $d = 64$ is significantly larger.

In this paper we do not explore the tuning of the parameters, which we leave as future work. However, we will show through experiments how different parameter choices can impact the final cost.

5 Experiments

The experiments are carried out via an event-driven simulator, which simulates a storage system for a set of peers whose behavior is described by availability traces that are provided as input.

Our objective is to compare the reliability, the storage, and network communication costs of traditional erasure codes and Hierarchical Codes.

In both cases we chose a (64,64)-code. In particular, the traditional code is a Reed-Solomon code, while the

Hierarchical Codes correspond to the two configurations presented in Fig. 4. This choice assures that the storage consumption is the same in all the scenarios.⁶

The reliability provided depends on the repair policy adopted. We consider a hybrid timer/threshold policy. It assumes the presence of an entity able to monitor the availability of the participating peers and trigger a repair operation according to the following rules:

1. When a peer A disconnects and the number of available peers n is smaller or equal to TH : $n \leq TH \rightarrow$ perform immediately the repair of the block stored on peer A .
2. When a peer A disconnects and $n > TH \rightarrow$ wait for a time T and then if A is still unavailable perform a repair of the block stored on A .

The timer T is used to distinguish between transient and permanent failures. In the ideal case in which T is chosen as the maximum possible disconnection time of a peer, whenever a disconnected peer does not reconnect within T , we are sure that it has abandoned the system for ever. In the real world, disconnection times may be bigger than T . In such a case, the blocks stored on a reconnecting peer are discarded, because they have already been repaired.⁷ This is a waste of resources that suggests to increase T . However, when T is increased, a higher number of peers is allowed to stay offline, in which case the set of online peers is not able to reconstruct the original fragments or is not able to perform repairs. To be quite insensitive to the choice of T , we introduced also the threshold, which has to be such that *availability* is provided, i.e. reconstruction is always possible.

In the case of Reed-Solomon codes, availability is provided if $n \geq k$, which requires that $TH > k$. We fix $TH = k + a$, which means that in the moment of minimum availability, i.e. in the moment of *maximum risk*, the system can still support a more losses. In the case of Hierarchical Codes, there is no fixed threshold that guarantees availability. As shown in Fig. 4, the minimum number of online fragments that provides availability depends on the particular losses that occur in the system and varies, in the case of Fig. 4a, from 64 to about 96. To be comparable with Reed-Solomon codes, our approach is the following: whenever a loss occurs

we recompute the probabilities $P(failure|l)$, which indicate the probability of failure if additional l losses would occur, taking into account the specific losses that already have occurred. If we want that, in the moment of maximum risk, the system can still support additional a losses, we apply the following rule: a repair is performed whenever $P(failure|a) > 0$.

Two notable facts are that (i) the repair policy for Hierarchical Codes needs to maintain a larger number of available blocks and tends to perform more repairs; (ii) the guarantees in terms of availability in the case of Hierarchical Codes **are stronger**: for Reed-Solomon codes, at the moment of maximum risk, if additional a losses would occur, the object will be unavailable with probability 1, while in Hierarchical Codes, the object will be unavailable with a probability that can be much smaller than 1.

In the experiments, we test different environments changing the stability of peers, and we measure the number of block transfers needed to maintain the code. We chose $a = 10$ and T to be three times bigger than the average disconnection time. In any case, we performed other experiments that showed that changing these parameters does not influence the results significantly.

5.1 Experiments with synthetic traces

In this set of experiments, the peer behavior is synthetically generated. In particular, every peer behaves according to a very simple Markovian model: a peer is available for an exponentially distributed time t_{on} , then upon disconnection it can abandon the system with probability p or can stay temporarily offline with probability $1 - p$ for an exponentially distributed time t_{off} , after which it comes back online.

We tested our hierarchical (64,64)-codes and Reed-Solomon (64,64)-code, using different combinations of the three parameters. The results suggest that, while p does not have a strong influence, t_{on} and t_{off} play an important role. Note that the ratio $up = t_{on}/(t_{on} + t_{off})$ represents the percentage of time that a peer spends online, or alternatively the ratio of peers that on average are online. It is clear that up has an influence on the number of repairs needed. This influence is different in Reed-Solomon codes and in Hierarchical Codes, since Hierarchical Codes need on average more peers to be online.

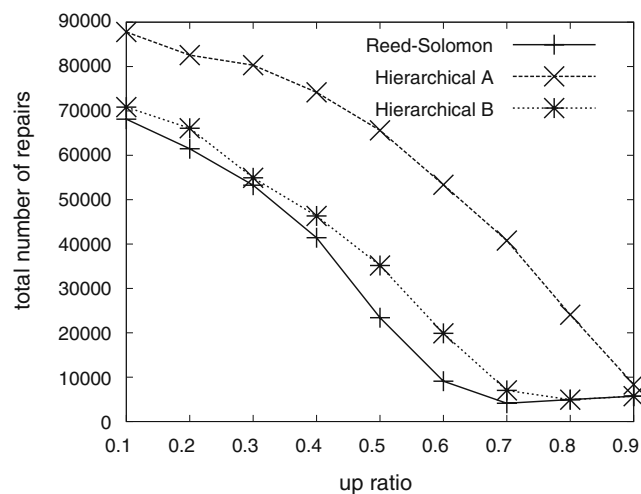
We run the simulation for 10000 time units, setting the disconnection time $t_{on} = 10$, the abandon probability $p = 0.001$ and selecting several values for t_{off} to test different values of the up ratio.

⁶It is $\beta = 2$, which means that every object consumes a space twice its size.

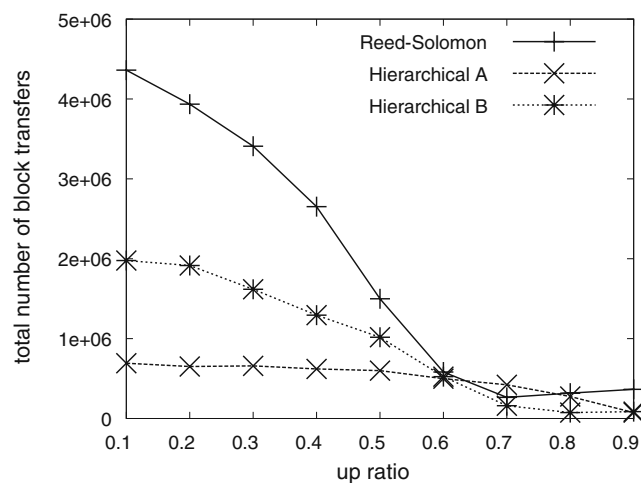
⁷For Hierarchical Codes, reintegration of this block is in some cases possible and would increase significantly our efficiency. However, identifying such cases is not trivial and it is left as future work.

As already mentioned, we evaluate Reed-Solomon codes and the two instances of Hierarchical Codes shown in Fig. 4. Figure 5 presents the results obtained; we label with ‘Hierarchical A’ the results obtained for the hierarchical code of Fig. 4a and with ‘Hierarchical B’ the results for the hierarchical code of Fig. 4b. In particular Fig. 5a shows the number of repairs needed by the three redundancy schemes, while Fig. 5b shows the total amount of block transfers executed for these repairs.

We see in Fig. 5a that, compared to Reed-Solomon codes, the hierarchical code ‘A’ requires a much larger number of repairs, while the hierarchical code ‘B’ requires only a slightly larger number of repairs. This is expected since (1) Hierarchical Codes need to be more reactive as they are more sensitive to losses (2)



(a) Total number of repairs.



(b) Total number of block transfers.

Fig. 5 Cost of maintenance for Reed-Solomon and Hierarchical (64,64)-codes as function of the up ratio $up = t_{on}/(t_{on} + t_{off})$. $a = 10$, $T = 3t_{off}$ (a, b)

The trade-off between reliability and repair cost of the hierarchical code in Fig. 4b is much closer to Reed Solomon with respect to the one in Fig. 4a, (i.e. a non-zero failure probability occurs for a number of losses closer to 64 and the repair degree is equal to 64 in a larger number of cases).

We also see that the number of repairs decreases when the *up ratio* increases. This is due to the fact that a higher *up ratio* corresponds to a higher percentage of peers on line, which in turn means fewer repairs.

The advantages of Hierarchical Codes are shown in Fig. 5b, where the actual communication costs introduced by the repairs is expressed in terms of the number of blocks transferred. Both hierarchical codes, in spite of their significantly higher number of repairs, generate in most of the cases much less block transfers than the Reed-Solomon code. In other words Hierarchical Codes require more repairs, but most of the repairs are quite cheap in terms of block transfers, which reduces the global repair traffic.

To have better understanding of the gain achieved in terms of block transfers, we show in Fig. 6 the ratio between the number of block transfers required by Hierarchical Codes and the one required by Reed-Solomon codes. This picture clearly shows the trade-off offered by the two different Hierarchical Codes. The instance ‘A’ appears to be more suitable for small values of *up ratio*, i.e. when the nodes are more unstable, while for a more stable environment the reduction in repair traffic is smaller and in some cases (namely for $up = 0.7$) it is even higher than in Reed-Solomon codes. On the other hand, the hierarchical code ‘B’ produces a smaller gain for unstable environments (roughly 40%

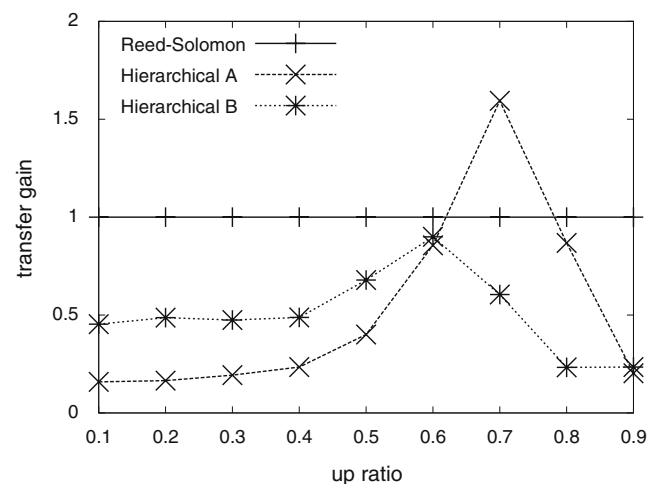


Fig. 6 Gain of Hierarchical (64,64)-codes in terms of number of block transfers with respect of Reed-Solomon (64,64)-codes as function of the up ratio $up = t_{on}/(t_{on} + t_{off})$. $a = 10$, $T = 3t_{off}$

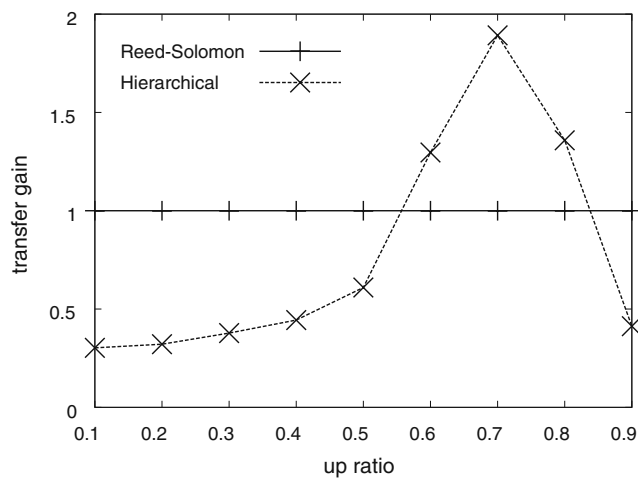


Fig. 7 Gain of Hierarchical (32,32)-codes in terms of number of block transfers with respect of Reed-Solomon (32,32)-codes as function of the up ratio $up = t_{on}/(t_{on} + t_{off})$. $a = 5$, $T = 3t_{off}$

vs 20%), while it reduces the repair traffic much more in stable environments. Also this code always achieves a lower repair traffic than the Reed Solomon code (the curve is always below 1).

In this section we presented in details the results obtained with Reed-Solomon codes and Hierarchical Codes, in the case of $k = 64$ and $h = 64$. We performed also experiments for different values of k and h and the results were consistent with the ones we presented. To give an example we propose in Fig. 7 the gain in terms of block transfers achieved by an instance of Hierarchical Codes⁸ over Reed-Solomon Codes, where $k = 32$ and $h = 32$. The results follow the same trend as in Fig. 6, although the gain of Hierarchical Codes is slightly reduced. This is due to the fact that a repair in Reed-Solomon code with $k = 32$ requires less transfers than in the case of $k = 64$ and consequently Hierarchical Codes have less space to produce a reduction.

5.2 Experiments with real traces

To evaluate the codes for a wider set of operating conditions than the ones given by the synthetic traces, we also use availability data of real distributed systems. We use two different traces:

1. **KAD traces:** obtained crawling a *KAD* network. These traces [13] characterize the availability of about 6500 peers in the *KAD* network, sampling their status every 5 minutes for about 5 months.

⁸This instance corresponds to the instance ‘A’ of Fig. 4a, adapted to have $k = 32$ and $h = 32$.

2. **PlanetLab traces:** obtained monitoring the connectivity of PlanetLab nodes. These traces [7] describe the availability status of 669 nodes, which was obtained by means of pings sent every 15 minutes among all pairs of PlanetLab nodes, starting from January 2004 for about 500 days.

The following table depicts the results for the two traces comparing the total number of repairs and the total number of block transfers for the Reed-Solomon and the two instances of Hierarchical Codes.

		Repairs	Transfers
PlanetLab	Reed-Solomon	472	30208
	Hierarchical A	637	4624
	Hierarchical B	487	6920
KAD	Reed-Solomon	765	48960
	Hierarchical A	3888	39710
	Hierarchical B	1072	20992

The results confirm the trend that we saw in the previous subsection: *Hierarchical Codes* require a higher number of repairs but result in a lower amount of blocks to be transferred. Once again we see the impact of the two different configurations on the results.

6 Conclusion

We presented a new class of codes, called Hierarchical Codes, which offer a flexible way of adding redundancy in distributed storage systems. Hierarchical Codes combine the advantage of reduced repair traffic offered by replication with the higher resilience against failures offered by coding. We believe that Hierarchical Codes make coding a *practical* alternative to replication in P2P storage systems.

Experiments validated our claims, showing that for a given level of availability, a higher number of repairs needed by Hierarchical Codes results in most cases in a smaller amount of repair traffic.

Moreover, we saw that Hierarchical Codes with a given redundancy factor, allow to trade-off in multiple ways reliability and repair cost. Future developments will focus on better understanding these trade-offs, which will allow to determine the optimal configuration of the codes for a given environment.

Finally it would be interesting to compare Hierarchical Codes with Regenerating Codes. We already showed in [6] that Regenerating Codes are able to reduce the repair traffic at the price of a higher computational cost, which can become in some cases prohibitive. The next step, which we plan as future work, is to

compare these two classes of codes taking into accounts both communication and computation costs, deploying both of them in a real P2P storage system.

Acknowledgement The first author was partially supported by a PhD Scholarship from Microsoft Research.

Appendix

A Proofs

A.1 Preliminary Proofs

Lemma 2 Consider an Information Flow Graph for a generic (k, h) -code at time step T . Consider a selection of k blocks B_1^k . Assume that there exists a condition C on this selection that guarantees that the original fragments can be reconstructed.

If for any time step $t \leq T$, any selection of B_t^k that fulfills the condition C can be perfectly matched with a selection of k blocks B_{t-1}^k in time step $t - 1$ that in turn fulfills the condition C ,

Then any selection B_T^k that fulfills the condition C allows the reconstruction of the original fragments.

Proof We proceed by steps:

step 1 Consider a selection B_1^k that fulfills the condition C . By assumption we know that the selection allows the reconstruction of the original fragments. This means, thanks to Lemma 1, that nodes in B_1^k have k distinct paths towards the original fragments F .

step 2 Consider a selection B_2^k that fulfills the condition C . By assumption we know that the nodes in this selection can be perfectly matched with a selection B_1^k that in turn fulfills the condition C . Thanks to previous step, we know that nodes in B_1^k have k distinct paths towards the original fragments F . This means that we can concatenate the perfect matching between B_2^k and B_1^k and the k distinct paths between B_1^k and F , obtaining k distinct paths between B_2^k and F .

The last step can be repeated until the time step T , where thanks to Lemma 1, the lemma is proved. \square

Lemma 3 Consider a code graph of a Hierarchical Code. Consider a group $G_{d_s, i}$ and denote as $F_{d_s, i}$ the subset of original fragments that are connected with nodes in this group $G_{d_s, i}$. Consider a selection of nodes B_1^k and consider the subset of this selection that belongs to the group considered: $A_{d_s, i} = B_1^k \cap G_{d_s, i}$.

If $|A_{d_s, i}| \leq d_s$ and $\forall j: G_{d_{s-1}, j} \subseteq G_{d_s, i}$, the nodes in $A_{d_{s-1}, j}$ have already been perfectly matched with $|A_{d_{s-1}, j}|$ nodes in $F_{d_{s-1}, j}$

Then it is possible to find a perfect matching between the nodes in $A_{d_s, i}$ and the nodes in $F_{d_s, i}$.

Proof Consider the nodes in $A_{d_s, i}$ that do not belong to the subgroups $G_{d_{s-1}, j} \subseteq G_{d_s, i}$ and denote them as \hat{A} . Consider the fragments in $F_{d_s, i}$ that have not been matched with the nodes in the subgroups $G_{d_{s-1}, j} \subseteq G_{d_s, i}$ and denote them as \hat{F} . The nodes in \hat{A} are connected with all the nodes in $F_{d_s, i}$ and can be thus all matched with nodes in the subset \hat{F} , as long as $|\hat{A}| \leq |\hat{F}|$. Since nodes in the subgroups have already been matched, then $|A_{d_s, i}| - |\hat{A}| = |F_{d_s, i}| - |\hat{F}|$, where $|F_{d_s, i}| = d_s$. This implies that whenever $|A_{d_s, i}| \leq d_s$, $|\hat{A}| \leq |\hat{F}|$ and the perfect matching is possible. \square

Lemma 4 Consider an Information Flow Graph of a hierarchical code at time step t . Consider a selection B_t^k that fulfills the condition (C2). Assume that a subset of α nodes $B_t^\alpha \subset B_t^k$ has already been perfectly matched with nodes in the previous step B_{t-1}^α that in turn fulfill the condition (C2). Consider a node $b \in B_t^k \setminus B_t^\alpha$, i.e. that belongs to the selection but has not yet been matched.

If all the repairs in the graph are done fulfilling condition (C3) and condition (C4), and all the blocks $b_i \in B_t^\alpha$ are such that $|R(b_i)| \leq |R(b)|$,

Then it is possible to augment B_{t-1}^α with another node that is matched with b , without violating the condition (C2) on the augmented set $B_{t-1}^{\alpha+1}$

Proof Let us use the following notation: $A_{d_s, i} = B_{t-1}^\alpha \cap G_{d_s, i}$ and $R_{d_s, i} = R(b) \cap G_{d_s, i}$. Assume that $G_{d_{s-1}, 1}$ is the group in which condition (C3) is fulfilled. This condition requires that $|R_{d_{s-1}, 1}| = |d_s|$. Note that all the nodes in B_t^α have a repair degree $d \leq d_s$, which implies that all the nodes in $A_{d_s, i}$ are necessary matched with nodes in $B_t^\alpha \cap G_{d_s, 1}$.⁹ Since $b \in G_{d_{s-1}, 1}$, thanks to condition (C2), $|B_t^\alpha \cap G_{d_s, 1}| < d_s$, which in turn implies $|A_{d_s, i}| < |d_s|$.

Consider two alternative cases:

case 1: $\exists j: 1 \leq j \leq g_s, |R_{d_{s-1}, j}| > |A_{d_{s-1}, j}|$: This means that there is a subgroup of the group $G_{d_{s-1}, 1}$ (that belongs to $G(b)$) that has at least one free node that can be matched with the block b . Since $|R_{d_{s-1}, j}| \leq |d_{s-1}|$, this node can be added without violating condition (C2) and the lemma is proved.

⁹To be matched with a node b_o outside $G_{d_{s-1}, 1}$, the repair degree of b_o must be bigger than d_s , which would violate the condition of the lemma.

case 2: $\forall j: 1 \leq j \leq g_s, |R_{d_{s-1},j}| \leq |A_{d_{s-1},j}|$: This means that there are no free nodes in the subgroups. This implies that: $\sum_{j=1}^{g_s} |R_{d_{s-1},j}| \leq \sum_{j=1}^{g_s} |A_{d_{s-1},j}|$. Consider the nodes in $A_{d_s,1}$ that do not belong to the subgroups and denote them as \hat{A} (they are among the h_s additional nodes), then consider the nodes in $R_{d_s,1}$ that do not belong to the subgroups and denote them as \hat{R} . We can write $\sum_{j=1}^{g_s} |A_{d_{s-1},j}| = |A_{d_s,1}| - |\hat{A}|$ and $\sum_{j=1}^{g_s} |R_{d_{s-1},j}| = |R_{d_s,1}| - |\hat{R}|$. Since $|R_{d_s,1}| = |d_s|$ and $|A_{d_s,1}| < |d_s|$, we have that $|\hat{R}| > |\hat{A}|$. This means that there is at least one free node in \hat{R} that can be matched with the blocks b without violating condition (C2) and the lemma is proved. \square

A.2 Proof of Proposition 1

Proof Thanks to Lemma 2, proving Proposition 1 corresponds to prove that in a generic time step t , only if repairs are done with a repair degree $d \geq k$, then any selection of nodes B_t^k can be perfectly matched with a selection B_{t-1}^k .

Consider a repaired node $b \in B_t^k$. All the other $k - 1$ nodes in B_t^k can be matched at most with $k - 1$ nodes in B_{t-1} . If b has been repaired with a degree $d < k$, it is possible that all the nodes in $R(b)$ have already been matched with the $k - 1$ nodes in B_{t-1}^k , preventing the matching of b . If $d \geq k$ there is at least one free node that can be matched with b . This can be repeated for all the repaired blocks proving, thanks to Lemma 2, the proposition. \square

A.3 Proof of Proposition 2

Proof Thanks to Lemma 2, proving Proposition 2 corresponds to prove that if a selection B_1^k is done fulfilling condition (C2), then it is possible to find a perfect matching between the nodes in B_1^k and the original fragments in F . This can be proved using iteratively the Lemma 3 from the innermost group that nodes in B_1^k belong to, to the outermost one.

A.4 Proof of Proposition 3

Proof Thanks to Lemma 2, proving Proposition 3 corresponds to prove that in a generic time step t , where repairs are done fulfilling the condition (C3) and condition (C4), any selection of nodes B_t^k that fulfills the condition (C2) can be perfectly matched with a selection B_{t-1}^k that in turn fulfills the condition (C2).

Thanks to Lemma 4, B_{t-1}^k can be found matching one by one the nodes in B_t^k proceeding from the nodes with the lowest repair degree to the nodes with the highest one. \square

B Computation of failure probability

Let us consider a Hierarchical (k,h) -code and assume that l losses occurred in this code, where $0 \leq l \leq (k + h)$. We first define the probability $P(k'|l)$, which is the probability that, given that l losses occurred, k' is the maximum number of alive fragments in the code, which fulfills the condition (C2). Note that the definition implies that $P(k'|l)$ exists only for $0 \leq k' \leq k$. Given these probabilities, computing the failure probabilities is straightforward:

$$P(\text{failure}|l) = 1 - P(k' = k|l)$$

To compute the failure probability we proceed as follows:

1. We compute the probabilities $P_0(k'|l)$ for the Hierarchical (k_0, h_0) -code, represented by the level 0 (the innermost) in the hierarchy as explained in Appendix B.1.
2. We compute the probabilities $P_s(k'|l)$ for the Hierarchical (d_s, H_s) -code, represented by the generic level s , using the probabilities $P_{s-1}(k'|l)$ computed for the hierarchical (d_{s-1}, H_{s-1}) -code, represented by the level $s - 1$, as explained in Appendix B.2.

B.1 Probabilities for level 0

At the level 0 the probability computation is straightforward:

$$P_0(k'|l) = \begin{cases} 1 & \forall k' = k_0 + h_0 - l, k' < k_0 \\ 1 & \forall k' > k_0 + h_0 - l, k' = k_0 \\ 0 & \text{otherwise} \end{cases}$$

B.2 Probabilities for level s

If we have l losses in a generic hierarchical (d_s, H_s) -code associated with the s -th level of the hierarchy we have many different ways in which these losses can be distributed among the g_s groups $G_{d_{s-1},i}$ this code is made of and the h_s fragments associated with this level s . Let us define the **Loss Configuration** of l losses denoted as LC_l as a vector of $g_s + 1$ elements $LC_l = \{l_0, l_1, \dots, l_{g_s}\}$, where each element l_i indicates how many losses occur in the group $G_{d_{s-1},i}$ except from

l_0 , which indicates how many losses occur among the h_s fragments of the level s . The constraints of LC_l are:

$$\begin{cases} l_0 \leq h_s \\ l_i \leq d_{s-1} + H_{s-1}, \forall i = 1, \dots, g_s \end{cases}$$

We denote as $P(LC_l)$ the probability that this configuration take place given that l losses occurred and we compute it as explained in Appendix B.3.

Every of the last g_s values in the configuration (all of them except l_0) indicates a number of losses l_i in a given subgroup and thus denotes a set of probabilities $P_{s-1}(k'|l_i)$, with $0 \leq k' \leq d_{s-1}$, which express the probability that k' is the maximum number of alive fragments from the subgroup i that fulfill the condition (C2) for the level $(s-1)$.

If, for each of this subgroup we select a specific value k'_i , we define a **Fragment Configuration** of $K' = \sum_{i=1}^{g_s} k'_i$ alive fragments denoted as $FC_{K'}$, whose probability is denoted as $P(FC_{K'})$ and given by:

$$P(FC_{K'}) = \prod_{i=1}^{g_s} P_{s-1}(k'_i|l_i)$$

The probability $P(FC_{K'})$ represents one of the components of the probability that, given the configuration analyzed LC_l , K' is maximum number of alive fragments taken from the subgroups such that the condition (C2) is fulfilled for the level s . To obtain the maximum number of alive fragments from the *whole group* that fulfill the condition (C2), K' must be augmented with $h_s - l_0$ alive fragments of the level s , with the constraint: $K' + h_s - l_0 \leq d_s$.

Putting the pieces together we can finally define the probability $P_s(k'|l)$:

$$P_s(k'|l) = \sum_{\forall LC_l} P(LC_l) f_s(k', LC_l)$$

where the auxiliary function $f_s(k', LC_l)$ is defined as follows

$$f_s(k', LC_l) = \begin{cases} 1 & k' < h_s - l_0 \\ \sum_{\forall FC_{k'-(h_s-l_0)}} P(FC_{k'-(h_s-l_0)}) & k' < k, k' \geq h_s - l_0 \\ \sum_{j=0}^{k_s-l_0} \sum_{\forall FC_{k'-j}} P(FC_{k'-j}) & k' = k, k' \geq h_s - l_0 \end{cases}$$

B.3 Loss configuration probability

We can map the loss configuration problem to the following balls and bin problem. Consider a set of $g_s + 1$ colors, for each color i there are n_i balls, which are inserted in a bin. We extract from the bin a total of l balls, which will form a color configuration described by a vector of $g_s + 1$ elements, where each element l_i

indicates how many balls of color i have been extracted. Considering the original loss configuration problem, our objective is to compute the probability of a given color extraction, where $n_0 = h_s$ and $n_i = k_{s-1} + h_{s-1}$ for $1 \geq i \geq g$. This probability can be computed dividing the number of possible configurations corresponding to the extraction by the total number of possible configurations, which gives:

$$P(LC_l) = \frac{\prod_{i=0}^{g_s} \binom{n_i}{l_i}}{\binom{\sum_{i=0}^{g_s} n_i}{l}}$$

where $LC_l = \{l_0 \dots l_g\}$ and:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

References

- Acedacnski S, Deb S, Medard M, Koetter R (2005) How good is random linear coding based distributed networked storage? In: NETCOD
- Adya A, Bolosky W, Castro M, Cermak G, Chaiken R, Douceur J, Howell J, Lorch J, Theimer M, Wattenhofer R (2002) Farsite: federated, available and reliable storage for an incompletely trusted environment. In: 5th symposium on OSDI. Boston, 9–11 December 2002
- Ahlsweide R, Cai N, Li S-YR, Yeung RW (2000) Network information flow. IEEE Trans Inf Theory 46(4):1204–1216
- Dabek F et al (2001) Wide-area cooperative storage with CFS. In: Proc. SOSP, Banff, 21–24 October 2001
- Dimakis AG, Godfrey B, Wu Y, Wainwright MJ, Ramchandran K (2008) Network coding for distributed storage systems. Computer research repository (CoRR). arXiv:0803.0632v1
- Duminuco A, Biersack E (2009) A practical study of regenerating codes for peer-to-peer backup systems. In: 29th intl conference on distributed computing systems (ICDCS), Montreal, 22–26 June 2009
- Godfrey B (2006) Repository of availability traces. <http://www.cs.berkeley.edu/~pbg/availability/>
- Haerberlen A, Mislove A, Druschel P (2005) Glacier: highly durable, decentralized storage despite massive correlated failures. In: NSDI05
- Li S-YR, Yeung RW, Cai N (2003) Linear network coding. IEEE Trans Inf Theory 49(2):371–381
- Mitzenmacher M (2004) Digital fountains: A survey and look forward. In: IEEE information theory workshop
- Plank JS (1997) A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Softw Pract Exp 27(9):995–1012
- Rodrigues R, Liskov B (2005) High availability in DHTs: erasure coding vs.replication. In: IPTPS05
- Steiner M (2007) Kad traces. <http://www.eurecom.fr/~btrouip/kadtraces/>
- Weatherspoon H (2006) Design and evaluation of distributed Wide-area on-line archival storage systems. PhD thesis, University of California, Berkeley
- Weatherspoon H, Kubiatowicz JD (2002) Erasure coding vs. replication: a quantitative comparison. In: Proceedings of IPTPS'02, Cambridge



Ernst W. Biersack received his M.S. and Ph.D. degrees in Computer Science from the Technische Universitat Munchen, Munich, Germany and his habilitation from the University of Nice, France. Since March 1992 he has been a Professor in Telecommunications at EURECOM, in Sophia Antipolis, France. His current research is on Peer-to-Peer Systems and Network Tomography.

Alessandro Duminuco received a bachelor degree in 2003 and a master degree in 2006 in computer science engineering from Politecnico di Torino (Turin, Italy). In 2004/2005 he spent a year at EURECOM (Sophia Antipolis, France) and he received an engineering diploma in computer science. In January 2006 he joined EURECOM as Ph.D. Student, where he is working with professor E. Biersack on peer-to-peer backup technologies.