Hierarchical Codes: How to Make Erasure Codes Attractive for Peer-to-Peer Storage Systems

Alessandro Duminuco and Ernst Biersack
EURECOM
Sophia Antipolis, France
{duminuco,biersack}@eurecom.fr

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. 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 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 aims at finding a flexible trade-off that allows the reduction of the network traffic due to maintenance without losing the benefits given by traditional codes.

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, 7, 12]. 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, 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 [10, 13]. 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 trivially a copy. 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 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.

While the coding schemes proposed in literature strive to reduce the storage efficiency, ignoring the other costs, we propose a new class of codes, called Hierarchical Codes, which take into account the cost metrics that are important for a P2P storage system. They introduce a flexible trade-off between replication and traditional erasure codes, which reduces the maintenance costs without scarifying storage efficiency.

In section 3.1 we formalize the efficiency metrics we are interested in. In section 3.2 a cost analysis is performed for the main existing redundancy schemes and in particular for linear erasure code. Thanks to this analysis we present our *Hierarchical Codes* in section 4. Finally in section 5 we evaluate our codes by means of experiments, which run a simulated P2P storage system using trace-based peer behaviors obtained by synthetic and real traces.

2 Related Works

Many works in literature discuss about the use of different redundancy schemes in P2P storage systems. The main point of the discussion has been the comparison between replication and coding. In [13] it is shown how erasure codes can give big improvements in terms of data durability as compared to replication, consuming the same amount of storage space. In [10] the authors perform the same comparison and, taking into account the peer behavior and the maintenance process, they conclude that in some cases the advantage of coding may be not worth its disadvantages. The work in [10] is the first one to point out the repair cost required by coding. To cope with this problem, the authors propose to adopt an *hybrid* scheme that uses both, coding and replication. This reduces the cost but, on one hand it increases the complexity and on the other hand, it loses most of the storage efficiency given by coding. This point has been well explained by [5], where authors propose a new class of codes, called *Regenerating Codes*, which are able to reduce the repair cost, consuming more storage space. Our work takes inspiration from that and proposes an alternative solution that does not sacrifice the storage efficiency. Both works, *Regenerating codes* and our proposal, adopt some of the concepts and the tools presented in the early literature about Linear Network Coding [3, 8].

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 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: $P(\mathit{failure}|l)$.

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 its 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 and producing 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 redundant 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:

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

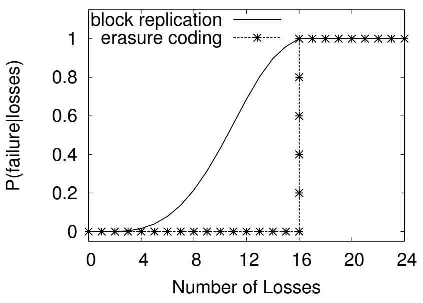


Figure 1. Block replication scheme compared to erasure codes with $k = 8$ and $R = 3$. $P(\mathit{failure}|l)$ as function of the number of concurrent losses l.

The redundancy factor is β=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 replicaion.

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 β=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.

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 $k+h$ redundant blocks such that any k of them are sufficient to reconstruct the original fragments. The number of losses that this scheme can sustain is h, while the redundancy factor is β=$(k+h)/k$. If, for example, we use k=8 and h=16 we will obtain β=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*.

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} redundant block. A linear code can be built using the following linear operations in GF(2^q):

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

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 redundant blocks.
$I_{k,k}$	Identity matrix.
$C_{h,k}$	Coefficient matrix.

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

$$B = \left[\begin{array}{c} I \\ C \end{array} \right] F = C'F$$

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.

Multiple choices of the coefficient matrix are possible, and consequently there exist multiple implementations of this class of codes. One of the most common is the *Reed-Solomon* codes [9], which define the matrix C as a $h \times k$ Vandermonde matrix, i.e. $c_{i,j}$=j^{i-1}. In Reed-Solomon, and in all the codes that fix a specific matrix for the coefficients, the repair of a lost block needs the reconstruction of all the original fragments and possibly their recombination accordingly to the coefficient row that corresponds to the lost block. This explains why the repair degree d=k.

Another approach is to build the matrix C' choosing randomly the coefficient in the Galois Field[1]. This class of codes are called *Random Linear Codes*. In [1] is shown that a $k \times k$ random matrix S in GF(2^q) is invertible with a probability which depends only on the field size and can pushed arbitrarily high increasing this size. A usual choice for the parameter q is $q \geq 16$, in which case the probability can be considered practically 1. This means that any $k \times k$ sub-matrix of C' is invertible and that the property of a (k,h)-code is provided.

The repair of a lost block can be done like in Reed-Solomon codes, i.e. reconstructing the original blocks and combining them again. In this case the repair degree would be again d=k. A property of random linear codes is that the reconstruction of original blocks is *an unnecessary step*, the result is indeed equivalent as to when the k redundant blocks are combined directly with random coefficients. One can be tempted, in this case, to use less than k blocks, reducing in this way the repair degree. However only d=k is able to preserve the properties of the code, in particular if $d < k$, there will be choices of k redundant blocks that are not sufficient to reconstruct the original k fragments.

This result can be derived from the fundamental works about network coding [8, 3] and from their application in distributed storage systems in [5]. We reformulate here some of the results in a slightly different but equivalent way. This formulation will help us in introducing our contribution.

Let us introduce an *Information Flow Graph*, which represents the evolution of the stored data through time. In particular each node represents a block of data in a specific point in time t. The time evolves at discrete steps and every step corresponds to one ore more losses and repairs. At the time $t = 0$ the graph is populated only by k source nodes representing the k original fragments and denoted as $F = f_1, f_2, \dots, f_k$. At $t = 1$ the graph is augmented with additional $k+h$ nodes, which represent the $k+h$ redundant 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 step 1 is referred to as the *code graph*. At $t > 1$ the graph is augmented with additional $k + h$ nodes, which represent the $k + h$ redundant blocks present in the system at time step t and denoted as $B_t = b_{1,t}, \dots, b_{k+h,t}$. Connections between nodes are possible only among nodes of consecutive time steps, oriented from t to $t - 1$. The possible connections and the corresponding semantic in the storage system are listed below:

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 $b_{j,t-1}$ in B_{t-1}, with $j \neq i$. This means that block b_i has been lost at time $t - 1$ and it has been repaired combining linearly the d (survived[2]) blocks b_j.

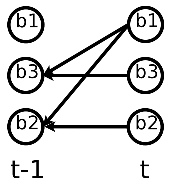


Figure 2. Example of one step of an *Information Flow Graph*. Blocks b_2 and b_3 have survived at time $t - 1$. Block b_1 has been lost at time $t - 1$ and has been repaired at time t combining the blocks b_2 and b_3.

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 $B_t^k \subseteq B_t$, 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_t^k to the k source nodes in F.*

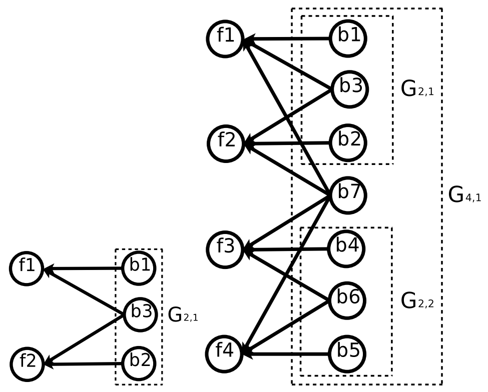
The *disjoint paths* condition is obviously related to the choice of the repair degree d. The following proposition holds:

Proposition 1. *At any time t, any of all the possible selections 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 $d \geq k$.*

[1]Note that replacing the identity matrix with random coefficients transforms the code from a systematic to an unsystematic one

[2]The fact that a node associated with a lost block is not connected to any node in the following step implies that a node in the step t can be connected only to survived blocks.

See the proof A.2 in the appendix . 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.



(a) Hierarchical (2,1)-code (b) Hierarchical (4,3)-code

Figure 3. Samples of Code Graphs for Hierarchical Codes.

4 Hierarchical Codes

The previous section showed as in a *traditional* linear erasure code the repair degree d cannot be lower than k. Indeed, if $d < k$, there will be selections of k redundant blocks *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 part of the possible choices of k redundant blocks (replicated fragments) is able to reconstruct the original object and this property is reflected in the reliability of the scheme, depicted 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 not an easy way to analyze the final reliability of the code, as we did in block replication; (ii) there is not a 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. 3(a).

   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. 3(a), 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 $G_{d_0,1} \ldots G_{d_0,g_1}$. Then add other h_1 redundant 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. 3(b).

   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. 3(b) 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 redundant 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 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 $P(\textit{failure}|l)$. They can be computed if we understand what are the sets of k redundant blocks that are able to reconstruct the original fragments. Using Lemma 1 applied to the *code graph* we can formulate 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:*

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

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. In the hierarchical (4,3)-code in Fig. 3(b), 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 $G_{4,1}$[3] .

Using Proposition 2 we can compute the generic probability $P(\textit{failure}|l)$, 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.

Repair Degree In the case of Hierarchical Codes, as in the block replication, there does not exist a specific 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. 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:*

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

and

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

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. preserve the properties of the code graph expressed in Proposition 2.*

See the proof A.4 in the appendix. In the hierarchical (4,3)-code in Fig. 3(b), this means that block b_1 can be repaired either combining blocks b_2 and b_3 either combining any 4 blocks among all the others; while blocks b_7 can be repaired only combining any 4 blocks among the others.

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[4].

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

[4]In the example of Fig. 3(b), if b_1 needs to be repaired and one between b_2 and b_3 is not available, the repair degree must be $d = 4$

Using the Proposition 3 and exploring all the possible combination of losses, we can compute the probability $P(d|l)$. $P(d|l)$ 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. 3(b) in the following table:

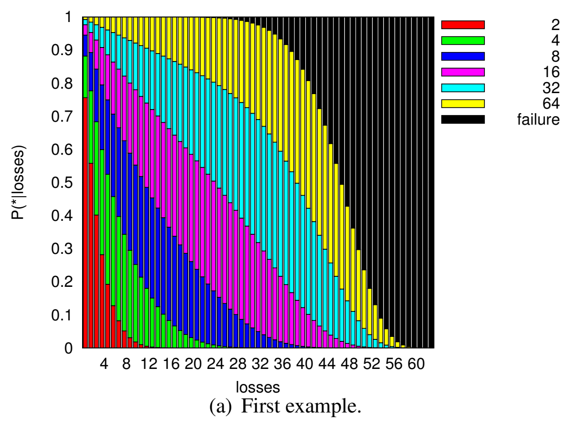
	l (losses)		
	1	2	3
$P(d = 2\|l)$	0.86	0.42	0
$P(d = 4\|l)$	0.14	0.58	0.77
$P(\textit{failure}\|l)$	0	0	0.23

The first two rows show the repair degree probability, while the last row shows $P(\textit{failure}|l)$. 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.

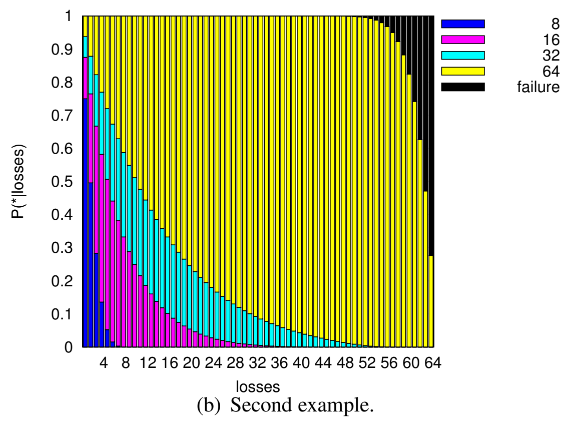
In Fig. 4(a), the same probabilities are graphically represented for a hierarchical code (64,64)-code, built using 6 levels and setting k_0=2, g_s=2 and h_s=1 for all the levels, except for the last level where h_5=2. Every bar in the plot corresponds to a column in the table, while the height of the 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 the reliability, indeed a traditional (64,64)-code does never fail for less than 64 losses, while the hierarchical code has chances of failure even for 32 losses, increasing for higher number of losses. However by adjusting the repair policy, as explained in next section, we can achieve the same reliability.

We believe that Hierarchical Codes give a new possibility to system designers to find the right trade-off between costs and benefits with respect to the characteristics of the environment in which the system is going to work. 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



(a) First example.



(b) Second example.

Figure 4. Examples of Hierarchical (64,64)-codes. $P(d|l)$ and P($failure|l$) as function of the number of concurrent losses l.

different balance between reliability and repair degree. An example is given in Fig. 4(b), which shows the results for an alternative hierarchical (64,64)-code: note how the reliability is increased along with the repair degree. In this paper we do not explore the tuning of the parameters and we leave it as future work.

5 Experiments

The experiments are based on an event-driven simulator, which simulates a storage system on a set of peers whose behavior is described by availability traces it receives as input.

Our objective is to compare the reliability and the costs in terms of storage and network communication when the system adopts alternatively traditional erasure codes and our *Hierarchical Codes*.

In both cases we chose (64,64)-codes, in particular the traditional code is a Reed-Solomon code, while the hierarchical code corresponds to the one presented in Fig. 4. This choice allows us to have the same storage consumption, given by β=2, which means that every object consumes a space that is twice its size.

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 disconnects and the number of available peers n is smaller or equal to *TH*: $n \leq \mathit{TH} \rightarrow$ perform immediately the repair of the block stored on that peer.

2. When a peer disconnects and $n > \mathit{TH} \rightarrow$ wait for a time T and then if the disconnected peer is still unavailable perform a repair of the block stored on that peer.

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[5]. 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 $\mathit{TH} > k$. We fix $\mathit{TH} = k + a$, 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 a 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. 4(a), 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 $P(\mathit{failure}|l)$, which indicate what is the probability of failure if other l losses occur, taking into account the specific losses that already occurred. If we want that, in the moment of maximum risk, the system can still support other a losses, we apply the following rule: a repair is performed whenever $P(\mathit{failure}|a) > 0$.

Two notable facts are that (i) the repair policy, in the case of Hierarchical Codes, needs to keep a bigger number of

[5]For 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.

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, in the moment of maximum risk, if other a losses occur, the object is certainly unavailable, while in Hierarchical Codes, the object is unavailable with a probability that can be very small.

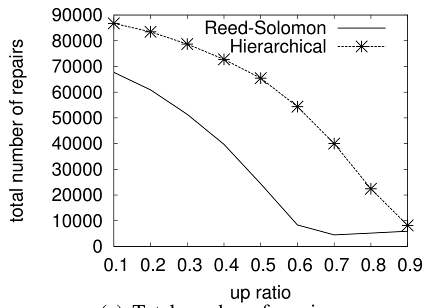
In the experiments we measure the number of block transfers needed to maintain the code with different peer behaviors. We chose $a = 10$ and T three times bigger than the average disconnection time. In any case we performed other experiments that showed that changing these parameters do not influence significantly the results.

5.1 Experiments with synthetic traces

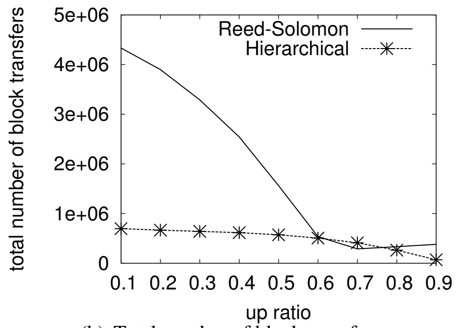
In this set of experiments the peer behavior is synthetically generated. In particular every peer behaves following a very simple Markovian model: a peer is available for an exponentially distributed time t_{on}, then upon disconnection it can abandon the system with probability P or can stay temporarily 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 combination of the three parameters. The results suggest that, while P does not give 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 on line, or alternatively the ratio of peers that in average are on line. It is clear that this 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 in average more peers on line.

We run the simulation for 10000 time units, setting the disconnection time $t_{on} = 10$, the abandon probability $P = 0.001$ and selecting multiple values for t_{off} to test different values of the *up ratio*. In Fig. 5(a) we show the number of repairs needed by the two redundancy schemes, while in Fig. 5(b) we show the total amount of block transfers needed by those repairs. The plots in Fig. 5(a) clearly show, as expected, that Hierarchical Codes require a bigger number of repairs. Moreover we can notice that the number of repairs decrease when the *up ratio* is increased. This is due to the fact that a higher *up ratio* corresponds to a higher percentage of peers on line, which in turn requires less repairs. The advantages of Hierarchical Codes are shown in Fig. 5(b): in most cases the number of block transfers is far less with respect to the one required by Reed-Solomon codes. In other words Hierarchical Codes require more repairs, but most of the repairs are very cheap in terms of block transfers reducing the global communication costs. Note that the advantage of Hierarchical Codes is reduced when the *up ratio* is increased (in some cases, the communication cost for Hierarchical Codes is even higher than the



(a) Total number of repairs.



(b) Total number of block transfers.

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

one for Reed-Solomon). This can be understood considering that in this high availability region, most of the peers are online and the system needs, with both codes, very few repairs. Note that this happens in a very fortunate and unrealistic case, in which the machines used are very stable and the repair policies becomes almost useless. In such cases a proper choice of the redundancy factor β is able to reduce almost to zero the number of repairs.

In real life, peers are less stable, their behavior changes in time and burst losses may occur. For this reason we tested our scheme, using real traces, as illustrated in the next section.

5.2 Experiments with real traces

To test our scheme using real availability data of real distributed systems, we fed our simulator with two different traces:

1. **KAD traces**: obtained crawling a KAD network. These traces [11] give the availability of about 6500 peers in the KAD network, sampling their status every 5 minutes for about 5 months.

2. **PlanetLab traces**: obtained monitoring the connectivity of PlanetLab nodes. These traces [6] consist in

the availability status of 669 nodes and were obtained by means of pings sent every 15 minutes between all pairs of the concerned PlanetLab nodes, starting from January 2004 for about 500 days.

The following table collects the results for the two traces comparing the total number of repairs and the total number of block transfers for both, the Reed-Solomon and the Hierarchical Code.

		Repairs	Transfers
PlanetLab	Reed-Solomon	472	30208
	Hierarchical	637	4624
KAD	Reed-Solomon	765	48960
	Hierarchical	3888	39710

The results confirm the trend shown in the previous subsection: *Hierarchical Codes require a higher number of repairs but a lower communication cost.*

6 Conclusion

We presented a new class of codes, called *Hierarchical Codes*, which represent a new flexible scheme to add redundancy in distributed storage systems. Their flexibility consists in offering a trade-off between the low-communication requirements of replication and the robustness of coding. We believe that this represents a new perspective for system designers, who have the possibility to consider coding as a *practical* alternative to replication in P2P storage systems.

Experiments validated our claims, showing that for a given level of availability, a higher number number of repairs needed by Hierarchical Codes results in most cases in a smaller communication cost. The experiments, obviously, are not able to give an exhaustive analysis of the performance of Hierarchical Codes, but they give a clear intuition of this behavior.

Moreover, this work does not explore the flexibility offered by the different instances of Hierarchical Codes. As mentioned at the end of section 4, a hierarchical code with a given redundancy factor, can mix in many ways reliability and repair cost. It is quite intuitive that different configurations of the code may give different results. Future developments will focus on how to build an optimal configuration of the codes, given a particular environment in which these codes are going to be used.

Acknowledgments The first author is supported by a PhD Scholarship from Microsoft Research.

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. □

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 $\forall j : G_{d_{s-1},j} \subseteq G_{s,i}$, 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. □

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 fullfilling 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 the condition (C2) on the augmented set $B_{t-1}^{\alpha+1}$*

Proof. Let us use the following notation: $A_{d,i} = B_{t-1}^\alpha \cap G_{d,i}$ and $R_{d,i} = R(b) \cap G_{d,i}$. Assume that $G_{d_s,1}$ is the group in which condition (C3) is fulfilled. This condition requires that $|R_{d_s,1}| = |d_s|$. 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 $B_t^\alpha \cap G_{d_s,1}$[6]. Since $b \in G_{d_s,1}$, 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 $G_{d_s,1}$ (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.

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 $\sum_{j=1}^{g_s} |A_{d_{s-1},j}| = |A_{d_s,i}| - |\hat{A}|$ and $\sum_{j=1}^{g_s} |R_{d_{s-1},j}| = |R_{d_s,i}| - |\hat{R}|$. Since $|R_{d_s,1}| = |d_s|$ and $|A_{d_s,i}| < |d_s|$, 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. □

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^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. □

[6]To be matched with a node b_o outside $G_{d_s,1}$, the repair degree of b_o must be bigger than d_s, which would violate the condition of the lemma

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 innest group that nodes in B_1^k belong to, to the outest 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 fullfills 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. □

References

[1] S. Acedacnski, S. Deb, M. Medard, and R. Koetter. How good is random linear coding based distributed networked storage? In *NETCOD*, 2005.

[2] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: Federated, available and reliable storage for an incompletely trusted environment. In *5th Symposium on OSDI 2002*, 2002.

[3] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4), July 2000.

[4] F. Dabek et al. Wide-area cooperative storage with CFS. In *Proc. SOSP 2001*, Oct. 2001.

[5] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. In *Infocom*, 2007.

[6] B. Godfrey. Repository of availability traces. `http://www.cs.berkeley.edu/~pbg/availability/`, 2006.

[7] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *NSDI05*, 2005.

[8] S.-Y. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2), February 2003.

[9] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[10] R. Rodrigues and B. Liskov. High availability in DHTs: Erasure coding vs.replication. In *IPTPS05*, 2005.

[11] M. Steiner. Kad traces. `http://www.eurecom.fr/~btroup/kadtraces/`, 2007.

[12] H. Weatherspoon. *Design and Evaluation of DistributedWide-Area On-line Archival Storage Systems.* PhD thesis, University of California, Berkeley, 2006.

[13] H. Weatherspoon and J. D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proceedings of IPTPS'02*, Cambridge, MA, Mar. 2002.