# Improving reliability and performances in large scale distributed applications with erasure codes and replication

Marco Gribaudo

*DEIB, Politecnico di Milano,*
*via Ponzio 34/5, 20133 Milano (Italy)*

Mauro Iacono

*DSP, Seconda Università degli Studi di Napoli*
*viale Ellittico 31, 81100 Caserta (Italy)*

Daniele Manini

*DI, Università degli Studi di Torino*
*corso Svizzera, 185, 10129 Torino (Italy)*

**Abstract**

Replication of Data Blocks is one of the main technologies on which Storage Systems in Cloud Computing and Big Data Applications are based. With the heterogeneity of nodes, and an always-changing topology, keeping the reliability of the data contained in the common large-scale distributed file system is an important research challenge. Common approaches are based either on replication of data or erasure codes. The former stores each data block several times in different nodes of the considered infrastructures: the drawback is that this can lead to large overhead and non-optimal resources utilization. Erasure coding instead exploits Maximum Distance Separable codes that minimize the information required to restore blocks in case of node failure: this approach can lead to increased complexity and transfer time due to the fact that several blocks, coming from different sources, are required to reconstruct lost information. In this paper we study, by means of discrete

*Email addresses:* `gribaudo@elet.polimi.it` (Marco Gribaudo), `mauro.iacono@unina2.it` (Mauro Iacono), `manini@unito.it` (Daniele Manini)

event simulation, the performances that can be obtained by combining both techniques, with the goal of minimizing the overhead and increase the reliability while keeping the performances. The analysis proves that a careful balance between the application of replication and erasure codes significantly improves reliability and performances avoiding large overheads with respect to the isolated use of replication and redundancy.

*Key words:* Performance modeling, cloud computing and big data infrastructures, storage systems, erasure codes

## 1. Introduction

The request for services that are based on big computing infrastructures is flourishing, and providers with different capability and specializations compete on the market. The abundance of proposals that are founded on the availability of data center technologies and facilities causes lower margins on the revenues of providers, specially because big players can afford more investments with longer strategies. Be the service a simple site hosting, a virtual server, or more complex offers like cloud applications or high performance computing, the main problem of a provider is to be able to supply a given level of performances to a given number of customers, at the minimum cost.

A big part of the investments is devoted to sustain quality of services, that is the result of a number of architectural, scheduling and management issues. Limiting the scope to the problem of providing a dependable storage subsystem, a big part of the investment is devoted to storage hardware and related connectivity. The quality of service specification over the storage subsystem implies the need for a consistent and safe way to ensure data persistence, that is generally based on replication strategies: data chunks are replicated over different storage units on different nodes, so that if a node or a unit fails (temporarily or permanently) the stored data are not lost and are continuously available to the owner. This solution is effective, even if it has some drawbacks: as first, the problem of replication management is not trivial, as replicas can be handled with different strategies, to find the most convenient mapping between data chunks and nodes; as second, the complete replication of data multiplies the number of storage units that are needed and complicate the interconnection infrastructure, significantly increasing the costs. A smart management of the problem, dealing with the need for space so that a lower expense is sufficient to keep data integrity and availability, would make the difference between being in or out of the market.

A possible solution is to move from replication to erasure codes. Erasure code techniques reduce the need for additional space while keeping the integrity level, by substituting replicas with additional redundant data that take less space. When a redundancy technique is applied, in case of node failures lost data are reconstructed by using the additional data (that has to be properly designed to be sufficient), that is distributed over other nodes in the most convenient way; the drawback is that in case of failure lost data have to be rebuilt, causing a momentary computing and communication overhead to the system. Choosing a good redundancy technique and designing the best trade off between additional space and computing needs can lead to significant savings in terms of infrastructural investments, by increasing the overall efficiency of the data storage subsystem. Such a choice is strategical, and must be as much as possible transparent to specific uses of the infrastructure; consequently, the choice should be focused on the lower layers of the system HW/SW stack.

In this paper we deal with optimal storage management. The original contribution of this paper is a simulation technique that supports, in terms of dependability and overall performance evaluation, the design of a strategy that exploits the existing resources by means of a redundancy-based approach that uses erasure codes, to help designers in predicting their best trade off between space and computing resources. We show how it is possible to model the storage subsystem with its management layer by means of an event based simulator that showed to be able to scale up to different architectures. More in details, in this work we focus on techniques that address data availability and durability in distributed storage systems such as Hadoop Distributed File System (HDFS) Borthakur (2008), Google File System (GFS) Ghemawat et al. (2003), and Windows Azure Calder et al. (2011)). In these systems, the occurrence of failures requires the storage of more replicas for each data block on more nodes, to guarantee that a useful number of copies are always available. To improve the tolerance of such systems when data are lost due to a node fault, many replication schemes have been introduced. Taking inspiration from the one presented in Simon et al. (2014) we propose a new idea in order to achieve the best tradeoff among data persistence and overall storage load distribution to avoid non-optimal resources utilization. We implement a simulator able to evaluate this approach and efficiently determine the system parameter settings. We propose a framework that allows tuning the system configuration to achieve the best data availability.

The paper is organized as follows: Section 2 discusses related works; in Section 3 we provide a brief background on replication and erasure coding techniques to improve the fault tolerance of the system. We then describe the proposed model in Section 4 and the simulator in Section 5. Results are then presented in Section 6, followed by conclusions.

## 2. Related works

For a general introduction to performance and dependability modeling of big computing infrastructures the reader can refer to Castiglione et al. (2014b); Barbierato et al. (2014); Castiglione et al. (2014a); Cerotti et al. (2015 (to appear) and Xu et al. (2014); Barbierato et al. (2015 (to appear); Yan et al. (2012), that specifically deal with storage problems; for an introduction to the problem of information dependability in distributed storage systems, we suggest Distefano and Puliafito (2014).

More specific references have been very useful for this work. The impact of replica placement in distributed systems is analyzed in Lian et al. (2005). In Simon et al. (2014) a highly distributed backup storage system is analyzed, based on nano datacenters to save costs, and the reconstruction process of multiple lost chunks is analytically modeled with its correlations, providing a good reference for a characterization in terms of distributions: the main issue is transferring the lost copies on the remaining nodes according to different policies (random, less load and power of choice), and by tuning the window size that determines which neighbors can be selected to minimize the data loss rate. In our work we aim to improve the reliability of the system by grouping blocks in groups composed of different entities, each stored in a different node. For each group, a number of redundancy blocks are created, the redundant information is stored using network coding techniques similar to the one introduced in Section 3.

A first quantitative comparison between erasure coding and replication in self-repairing and fault resilient distributed storage systems can be found in Weatherspoon and Kubiatowicz (2002). The solution has been explored in peer to peer systems: in Kameyama and Sato (2007) a family of erasure codes with a very low overhead factor is applied to distributed storage to show the appropriateness of this approach; in Dandoush et al. (2009) the problem of lost data blocks reconstruction in distributed storage systems is studied in peer to peer architectures, providing an interesting characterization in terms of empirical distributions obtained by means of event based simulations; a further application is in Aguilera et al. (2005); an analysis of the combined application of redundancy and erasure coding in DHT is given in Wu et al. (2005); in another analysis Rodrigues and Liskov (2005) the authors evaluate the performances of DHT by an analytical approach based on traces of 3 different applications, and conclude that results are actually confirming the expected advantages, but cost an excess of complexity in the design of the overall system because of the implementation of erasure coding.

More recently, the same solution has been considered for application in Big Data applications: in Xiang et al. (2014) an application of erasure codes to distributed file system is presented, providing an analytical upper bound of the average service delay due to the network and the needed computations: the paper specially points out the role of the factors that generate latency as a consequence of erasure coding, and their optimal management; in Sathiamoorthy et al. (2013) another family of erasure codes is proved to be applicable to Big Data systems.

The work presented in Friedman et al. (2014) is the most similar proposal to the the solution analyzed in this paper. The authors propose a two-phase technique, namely Replicated Erasure Codes, that combines the storage space efficiency of erasure codes and the repair traffic efficiency of replication. The work is supported by an analytical evaluation of its basic characteristics, that are studied in a range of situations by means of a model, based on some simplification assumptions, and a simulation based on a peer to peer node availability trace. With respect to Friedman et al. (2014), this paper is more focused to analyze the impact of node failures rather than chunk or file failures, and to provide an optimal selection strategy for the allocation of redundancy and coding data; consequently, the repair mechanism privileges a reactive approach, considering that bandwidth is a secondary problem in data centers with respect to data integrity and optimal storage management, specially when the target is Big Data. Moreover, our analysis is based on a simulation that focuses on situations that are more likely to arise in real data centers, and also includes transients.

## 3. Data protection strategies

Data protection strategies can be used to increase the availability of data. To simplify the presentation, we will consider a simple scenario and we will informally describe some of the possibilities. Let us focus on a system where 8 data chunks, named from $A$ to $H$, are distributed over four nodes $n_1$ to $n_4$.

### *3.1. Chunks duplication*

The simplest replication strategy corresponds to storing several different copies of the same chunk, as shown in Figure 1. In this case chunks $A$ and $E$ are available on nodes $n_1$ and $n_2$, chunks $C$ and $F$ on $n_2$ and $n_3$ and so on. In this case, a total of 16 blocks in the system is required to store the two copies of the 8 chunks of data. Let us now suppose that node $n_4$ fails, and all its content is lost, as shown in Figure 2. In this case no chunk is lost because $C$ and $G$ are still available on $n_3$, while $D$ and $H$ can be found on $n_1$. If however also $n_3$ fails, (Figure 3), then chunks $C$ and $G$ are lost forever since they are no longer available in any of the node still on-line. The system is still holding 8 blocks: in particular chunks $A$ and $E$ are still available in two copies on nodes $n_1$ and $n_2$. In this case, the system is able to tolerate only the failure from one node before losing some of its data.

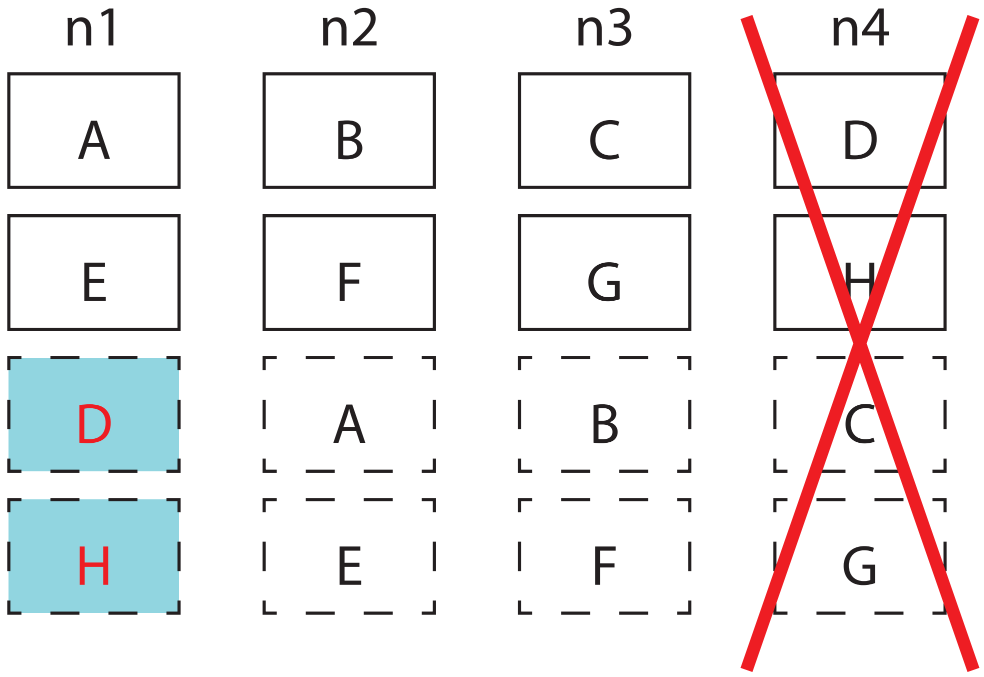

Figure 1: Full duplication of chunks: all nodes available.

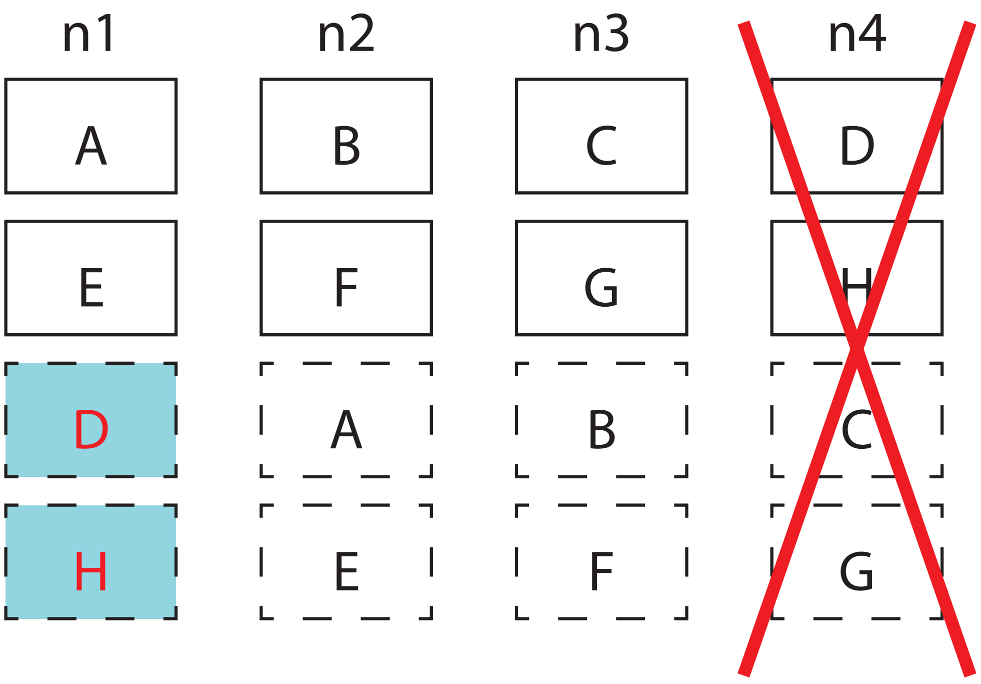

Figure 2: Full duplication of chunks: one node down.

## *3.2. Parity blocks*

Let us now consider a different approach. Instead of duplicating the chunks, we store in a block on one node the sum of the data contained in two chunks on two other nodes. For example, we store $A + B$ and $E + F$ on $n_3$, $B + C$ and $F + G$ on $n_4$ and so on, as shown in Figure 4. If again node $n_4$ breaks, then chunks $D$ and $H$ are no longer directly available in the system. However they can be reconstructed by reading respectively $D + A$ and $H + E$ from $n_2$ and subtracting $A$ and $E$ from $n_1$ (Figure 5). Now if also node $n_3$ fails, no data is lost: chunks $D$ and $H$ can be computed as just shown. Chunks $C$ and $G$ can be computed by subtracting the just reconstructed chunks from $C + D$ and $G + H$ stored in $n_1$ (Figure 6). This system, even if it still uses the same number of blocks, i.e. 16, as in the previous case, it is able of surviving up to two node failure without losing data.

A similar approach can instead be used to have the single node failure tolerance, while using a smaller total number of blocks. For example, in Figure 7, $n_3$ holds $A + B$, while $E + F$ is stored on $n_4$ and so on. If node $n_4$ breaks, as shown in Figure 8, the system is still able to recompute chunk

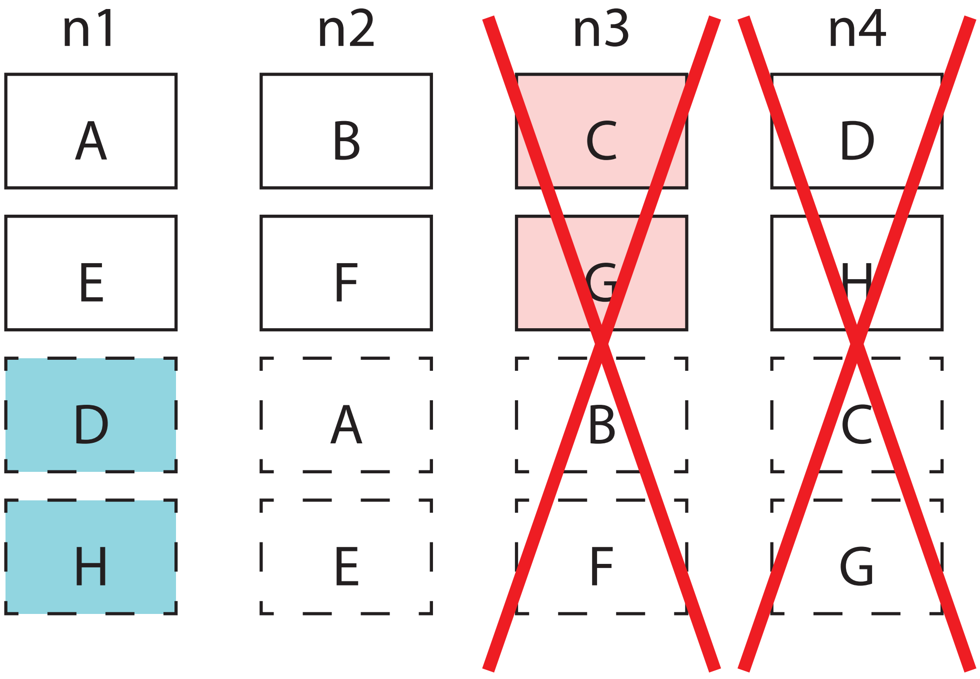

Figure 3: Full duplication of chunks: two nodes down.

$D$ by reading $C + D$ from $n_1$ and $C$ on $n_3$, and $H$ subtracting $G$ (still on $n_3$) from $G + H$ on $n_2$. Note that this configuration will no longer be able to correct two node failures, as shown is Figure 9. However, this configuration requires only 12 blocks instead of 16.

Note that the three different strategies have also a different evolution in the number of available chunks as the number of failure increases, as shown in Table 1. For example, while the second policy is the best in keeping all the eight chunks available for a larger number of nodes failures, the first one is the one that is able to maintain a larger number of available chunks even when there is just on single node left.

### *3.3. More complex parity models*

The previous techniques can be improved using more parity blocks. Figure 10 shows an example of a double parity system. Two different redundant blocks are generated for chunks $E$, $F$, $G$ and $H$: block $R = E + F + G + H$ on node $n_3$, and block $Q = E + 2F + 4G + 8H$. It is important that the parities are computed in different way so that they are independent one from the

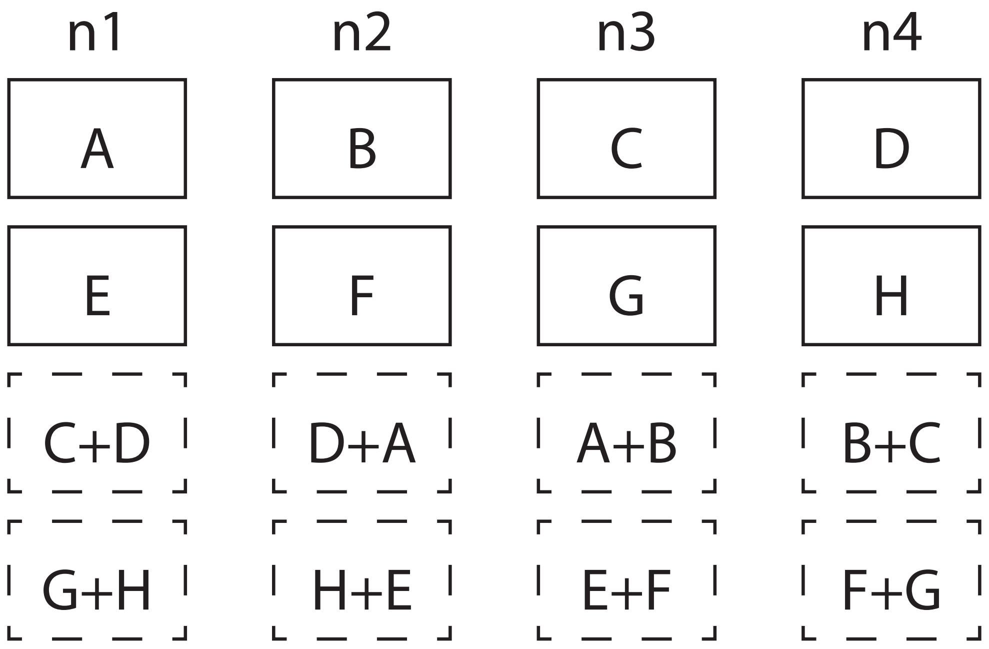

Figure 4: Using parity instead of duplication of chunks: all nodes available.

other. This requires that a different formula is used for the two parity blocks. With this strategy, the system can recover the data after two node failures with a limited overhead. For example, in 11 nodes $n_5$ and $n_6$ fail, however their content is not lost. Block $E$ and $F$ can be reconstructed from the two redundant pieces of information $R$ and $S$ on nodes $n_3$ and $n_4$, plus the two blocks $G$ and $H$ that are still available on nodes $n_1$ and $n_2$. In particular we have:

$$\begin{aligned} F &= S - R - 3G + 7H \\ E &= R - F - G - H \end{aligned}$$

In a similar way blocks $K$ and $L$ can be reconstructed as $L = (Q - 4P + 3I + 2J)/4$ and $K = P - I - J - L$.

The techniques that can used to compute parity blocks in different ways can be based on the so called *Erasure Codes*, which are a type of *Forward Error Correcting* codes that transform a set of $k$ symbols into a set of $n$, with $n > k$. The key property is that the $k$ original symbols can be reconstructed from a subset of $n$. In particular, *Maximum Distance Separable Codes*, denoted as $(n, k)$ MDS, are codes that can reconstruct the original

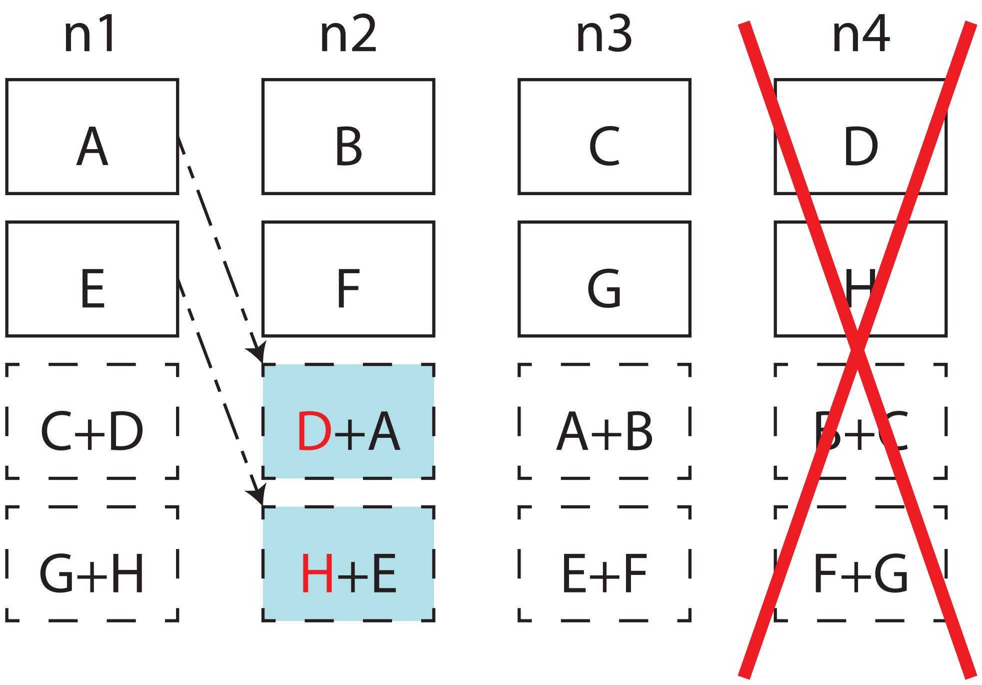

Figure 5: Using parity instead of duplication of chunks: one node down.

symbols form any set of $k$ out of the $n$ generated. A well known example of Maximum Distance Separable codes are the *Solomon-Reed codes used* to protect RAID 6 Plank (1997) disk arrays from multiple disks failures. Such techniques uses equations similar to the one proposed in the example in Figures 10 and 11, but exploits Galois fields to allow the solution of the corresponding equations to be computed using a special integer arithmetic. The $k$ initial symbols correspond to the data on $k$ disks, and $n = k + 2$ adds two parity disks to the array, allowing the system to be resilient up to two disk failures. Similar techniques are also used in *Network coding*, a technique used in telecommunication to increase the dissemination of contents while reducing the total bandwidth required.

## 4. The storage model

We focus on a system composed by $n$ nodes, where $m$ ($m >> n$) storage *chunks* have to be stored. For each chunk, the system tries to keep up to $k$ copies on different nodes. The copies of the chunks spread in the system

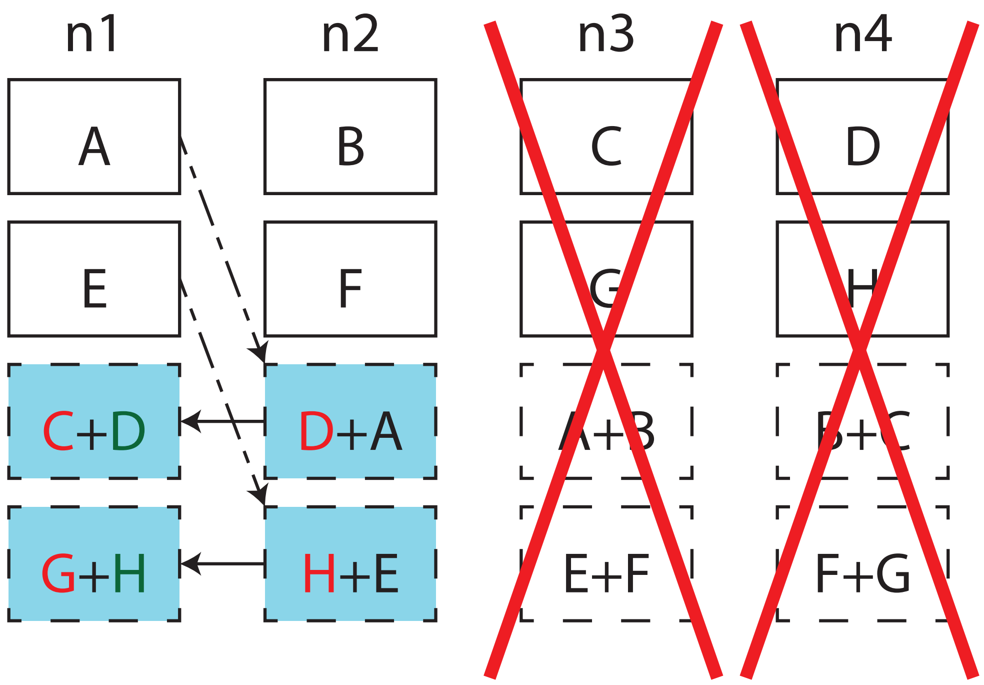

Figure 6: Using parity instead of duplication of chunks: two nodes down.

are called *blocks*. The number of blocks stored on a node is limited to $\nu_{\max}$. When a node fails, all the stored blocks are lost. If the total number of copies of a chunk reaches 0, it is lost forever. The reliability of the system is further improved by composing groups of $g$ different chunks. For each group, $h$ *redundancy blocks* are created. The redundant information is stored using erasure codes based techniques similar to the one introduced in Section 3: in particular, each of the original chunk can be reconstructed as long as there are still at least $g$ out of the $g + h$ blocks including the redundancy. To improve reliability even further, each block can belong to up to $p$ different groups. None of the blocks in any of parity groups should be stored on the same node, to prevent correlation among node failures. All the parameters of the model are summarized in Table 2. Replication of chunks also allows to reduce the time required to transfer the data. Requests can be sent in parallel to all the nodes storing the copies of the chunk, and the user can get the block from the fastest.

For instance, the examples proposed in 3.1 are all characterized by $n = 4$ and $m = 8$. The case with simple redundancy shown in Figures 1, 2 and 3 is

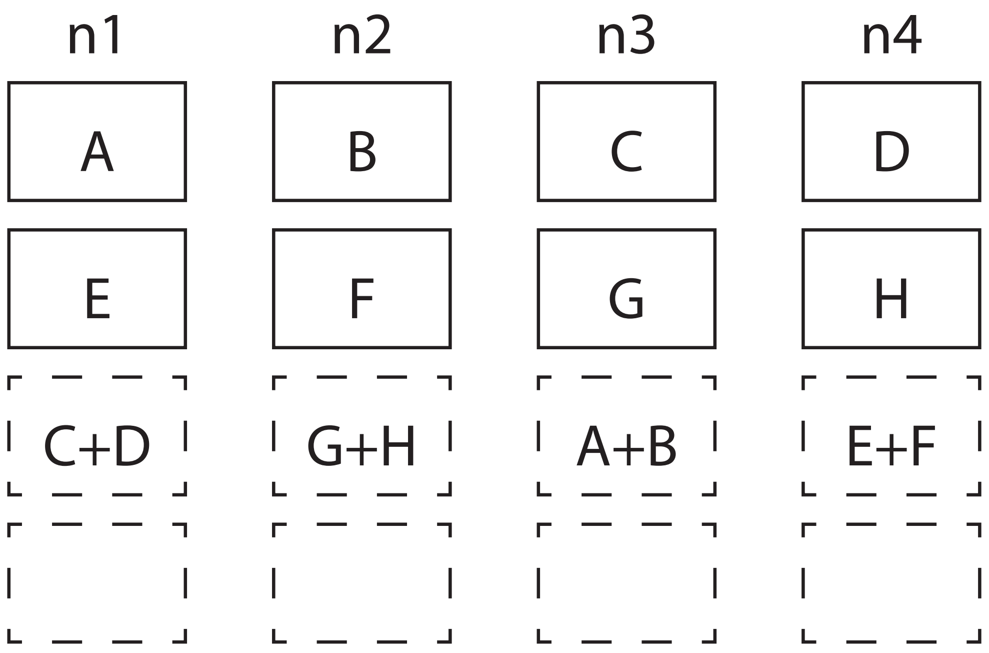

Figure 7: Using less parity blocks: all nodes available.

characterized by $k = 2$, $h = 0$, $g = 1$ and $p = 0$, i.e., each file is characterized by two copies, but there is no redundancy. The other two examples are both characterized by $k = 1$, $h = 1$ and $g = 2$: in those cases there is just one block for each chunk, and a single redundancy block is added for every couple of chunks. However, the case of Figures 4, 5 and 6 has $p = 2$, since each chunk is used two times (for example, chunk $A$ is used together with chunk $D$ in node 2, and together with chunk $B$ in node 3). Figures 7, 8and 9 report the case with $p = 1$. Finally, the model presented in Figures 10 and 11 is characterized by $n = 6$, $m = 12$, $k = 1$, $h = 2$, $g = 4$ and $p = 1$ since two redundant blocks are generated for every group composed of four chunks, and each chunk belongs to a single parity group.

In this scenario, the total number of blocks in the system can be computed as:

$$N = \left\lceil m \cdot \left( k + p \cdot \frac{h}{g} \right) \right\rceil \tag{1}$$

since each of the $m$ chunks is present in the system $k$ times. Moreover, the $h$ additional parity information is shared among the $g$ chunks of the group. However, since the chunks can belong up to $p$ groups, $h/g$ must be multiplied

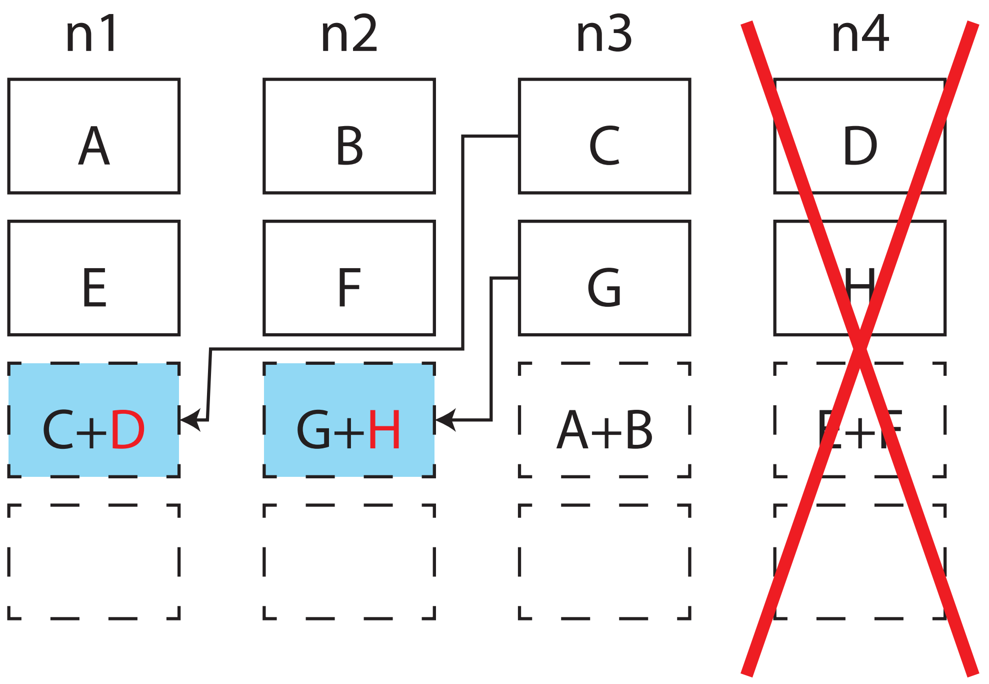

Figure 8: Using less parity blocks: one node down.

by $p$.

The evolution of the replication system is characterized by several rates. Each node can fail at rate $d$. Duplication is performed at rate $r$ and redundancy is computed at rate $q$. If no more actual copies of a block are available, the missing data can be reconstructed from the other blocks in the group and the redundant information at rate $s$. Finally, the chunks are requested by users at rate $c$, and the time required to transfer a block from node $i$ is defined by a random variable $T_i$.

## 5. Simulation

This section provides the description of the simulator we implemented to evaluate a storage system characterized by both replication and erasure codes. We suppose that the systems starts with all the chunks existing in a single copy, and uniformly distributed among the nodes. Then the simula-tion evolves according to five possible events: *copy of a chunk*, *redundancy computation for a chunk*, *failure of a node*, *chunk reconstruction* and *chunk*

Table 1: Survivability of blocks: number of blocks still available, depending on the number of failures

| # nodes available | 4 | 3 | 2 | 1 | 0 | Occupancy |
|---|---|---|---|---|---|---|
| Full replication | 8 | 8 | 6 | 4 | 0 | 16 |
| Double parity | 8 | 8 | 8 | 2 | 0 | 16 |
| Single parity | 8 | 8 | 4 or 6 | 2 | 0 | 12 |

Table 2: Parameters of the model

| Parm. | Description |
|---|---|
| $n$ | Number of nodes |
| $m$ | Number of chunks |
| $k$ | Number of full copies for each chunk |
| $\nu_{\max}$ | Maximum number of blocks stored on a node |
| $p$ | Number of parity groups |
| $g$ | Size of the groups |
| $h$ | Additional parity blocks per group |
| $d$ | Node failure rate |
| $r$ | Duplication rate |
| $q$ | Redundancy computation rate |
| $s$ | Reconstruction from redundancy rate |
| $c$ | Block request rate |
| $T_i$ | Transfer time distribution for block $i$ |

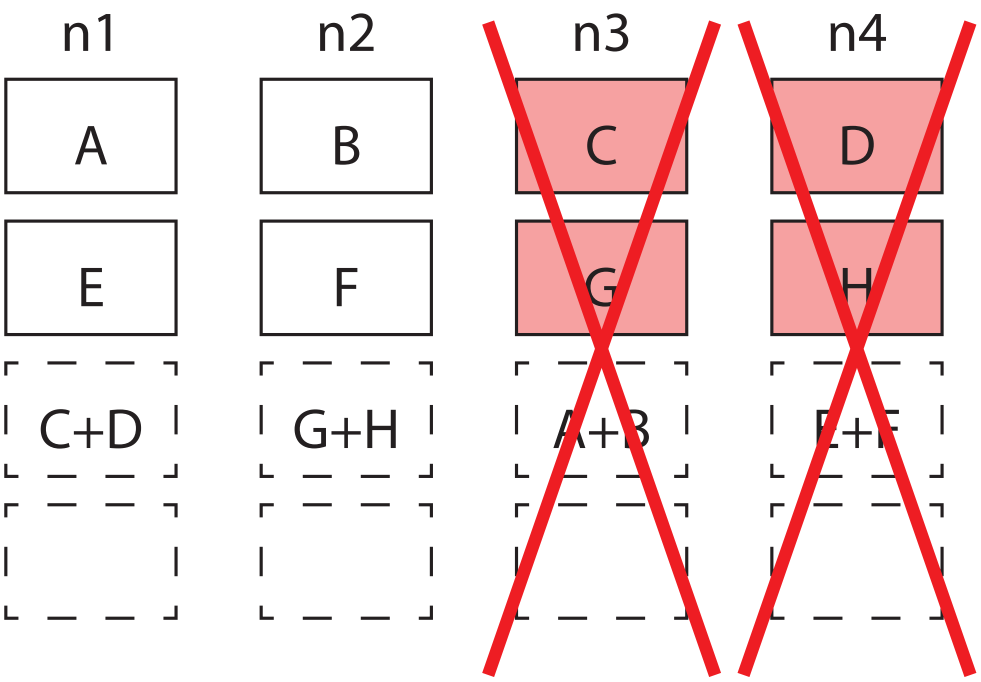

Figure 9: Using less parity blocks: two nodes down.

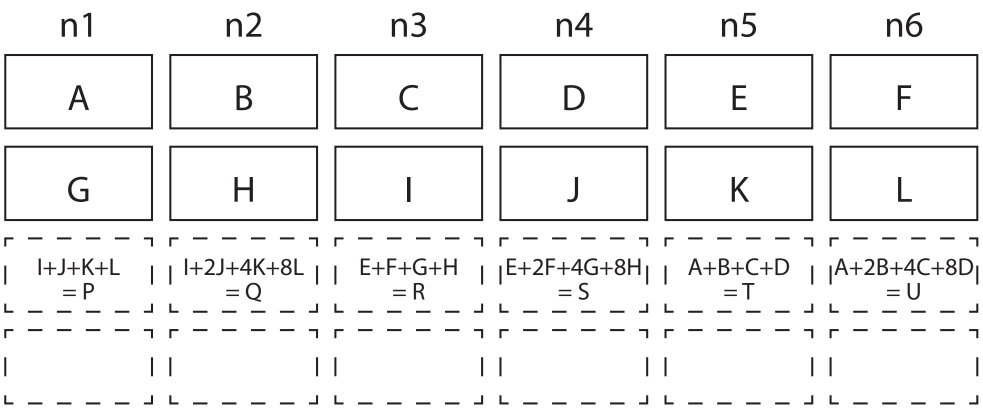

Figure 10: Using a double parity system: all nodes available.

*transfer*. The copy of chunks occurs until the number of existing copies of each block reaches $k$. Then, if some of the copies are lost, the copy event is scheduled again to restore the target number of copies. In the same way, the redundancy computation is scheduled until a chunk gets included into $p$ dif-

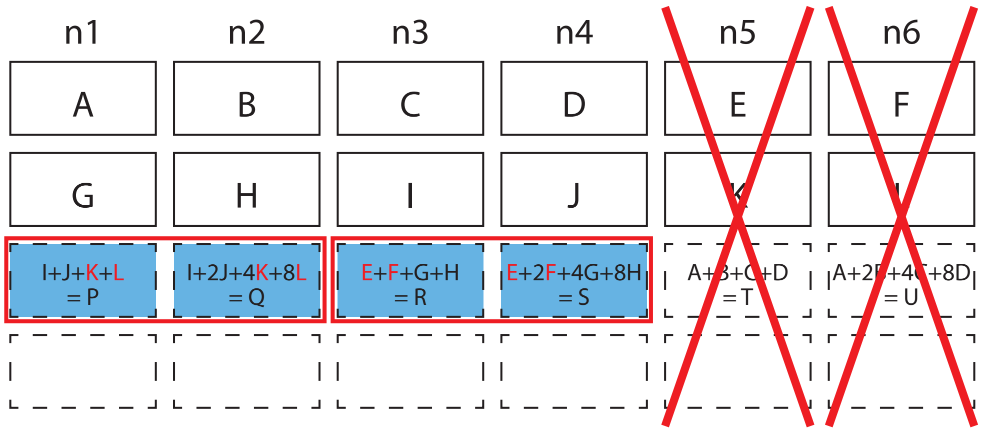

Figure 11: Using a double parity system: two nodes down.

ferent parity groups. The reconstruction of a chunk is instead scheduled only when the number of copies of a chunk reaches 0 and data must be restored using the parity mechanism. Node failure can always happen: to simplify the model, we disregard the repair time of the nodes and we imagine that when a node fails, it immediately starts working again as an empty node. Finally, chunk transfer is scheduled for every chunk that is available in the system (i.e. it exists at least in one copy, or it can be reconstructed using parities). In the following, we will briefly describe how events are handled by the simulator.

### *5.1. Copy of a chunk*

The copy of a chunk event occurs for chunks that have less than $k$ copies in the system: its goal is to find a node that is suitable to hold a new copy of the chunk, and transfer the corresponding data to that node. The node selection can be performed by applying one between the *Random* and the *Power-of-choice* Mitzenmacher et al. (2000) policies: the former uniformly uses all the possible nodes, whereas the latter selects randomly (following a uniform distribution) two valid nodes and then it chooses the less loaded. The two type of policies can lead to a different node usages, as it will be shown in Section6.5. In order to reduce correlation among failures, a node must satisfy several properties to be suitable to hold a new copy of a chunk. In particular, a node is valid for storing a chunk $A$ if it has available space and if it does not hold yet either a copy of the same chunk, or a copy of a chunk that belongs to a redundancy group containing $A$, or redundancy information added to a redundancy group to which $A$ belongs. Figure 12 shows an example for the copy of chunk $A$ in a case with $k = 2$, $p = 1$, $g = 2$ and $h = 1$. A parity group which includes nodes $A$ and $B$ has already been formed, and the corresponding parity block is stored on node $n_5$. The second copy of $A$ cannot be placed on $n_1$ since it already holds a copy of $A$. It cannot be put either on $n_2$ or $n_4$ since they are already full. It cannot be stored on $n_3$, since it includes $B$ which is in the same parity group of $A$, and cannot be placed on $n_5$ since it holds parity $Q = A + B$. Node $n_6$ is instead suitable since it does not violate any of the given requirements.

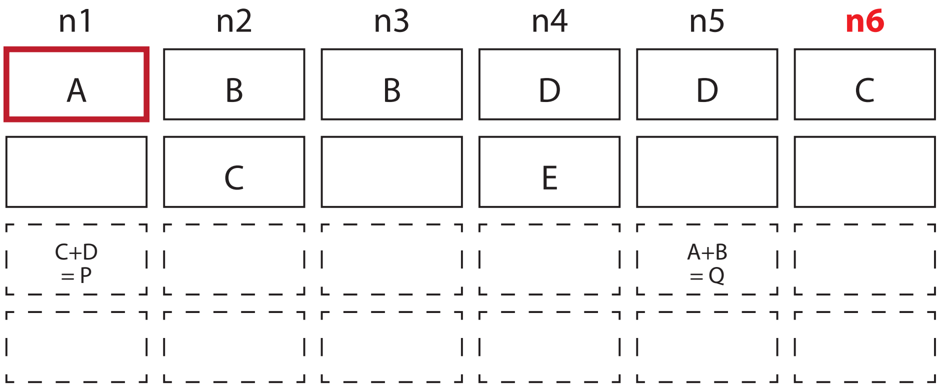

Figure 12: Selecting a node to place a copy of chunk $A$.

## *5.2. Redundancy computation*

The redundancy computation event has the goal of forming groups of $g$ chunks, and protecting them by adding an extra $h$ redundancy blocks. In order to avoid possible correlation among node failures, all the copies of the $g$ chunks of the group and all the $h$ related parity blocks must be stored on different nodes. Potentially, a parity group can span over $g \cdot k + h$ nodes (the $k$ copies for the $g$ chunks of the group, plus the $h$ parity blocks), and the number of nodes whose failure can be correlated with a block can grow up to $k + p \cdot (k \cdot (g - 1) + h)$ when considering that a chunk can belong up to $p$ parity groups. The construction of a valid parity group for a chunk is thus a quite complex task. The process starts from a chunk $A$ that has not been included yet in all the $p$ parity groups. If $p = 1$, this corresponds to a chunk that does not belong to any parity group yet. For the sake of simplicity, we will limit the description to the case with $p = 1$. In order to find suitable candidate chunks to be included in a new parity group, the procedure creates a list of invalid nodes that cannot be used since their failure would be correlated with the chunks already present in the group. This list starts by including the nodes already holding a copy of chunk $A$. The algorithm then selects a random node which is not in the invalid list, and starts looking for a chunk that is compatible: a chunk whose copies are not in any invalid node. As soon as the search finds a valid chunk, it adds all the nodes in which it is stored to the invalid list, and repeats the procedure until a group of $g$ chunks has been formed. If a node does not have any chunk that can be selected, that node is added to the invalid list, and the algorithm selects a new node. If no more nodes are available, the parity group can not be formed, and the process is ended. A similar procedure is used to select the nodes that can host the $h$ parity blocks. When parity and redundancy are completed, the group is considered done, and its definition is stored in the system. Figure 13 shows an example of parity group construction for a chunk $A$. Nodes $n_1$ and $n_2$ cannot be chosen since they already hold a copy of chunk $A$. Node $n_3$ contains chunk $B$, but it cannot be selected since a copy of block $B$ is on node $n_2$ together with $A$. The selection then tries node $n_4$: chunk $E$ cannot be used, since it already belongs to a parity group (with chunk $C$, and parity on node $n_1$). Chunk $D$ can be chosen since it does not violate any constraint. This adds nodes $n_4$ and $n_5$ to the invalid list, since those are the nodes where chunk $D$ is stored. The parity block can then be inserted on block $n_3$ or $n_6$: in this example, node $n_6$ is chosen.

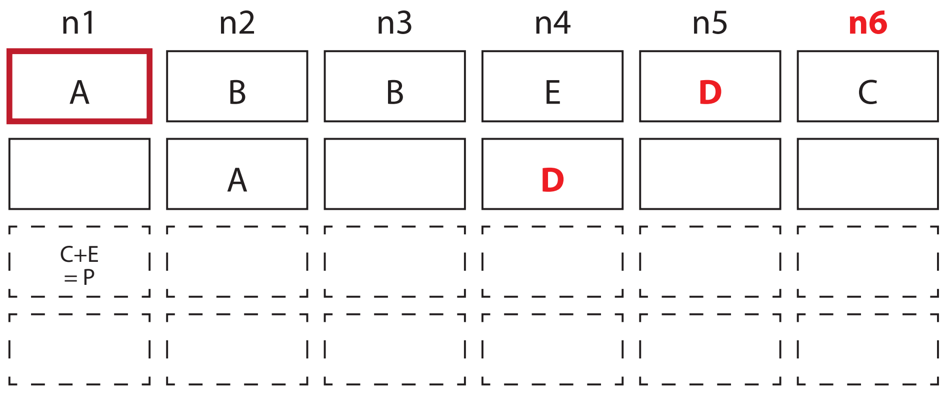

Figure 13: Selecting a parity group including chunk $A$.

### *5.3. Node failure*

When a node fails, all the blocks it stores are lost. For each chunk that was on the node, the number of available copies is reduced of one unit. If the number of copies reaches 0 two scenarios are possible: if the node belongs to a parity group where at least $g$ blocks remains, including chunks and redundancy blocks, then the chunk can be reconstructed. In this case the chunk reconstruction event is scheduled. If instead the chunk does not belong to any parity group, or if the parity groups are already missing $h$ of their $h + g$ blocks, the chunk is lost forever. In this case, the chunk is removed from the simulation, and no longer considered in the system evolution. All the parity groups to which the lost chunk belonged are also destroyed, since they are no longer capable of reconstructing any block, and this can lead to cascade failures. Similar considerations are also applied to parity blocks stored on the failed node.

### *5.4. Chunk reconstruction*

As introduced before, the reconstruction event is enabled when the failure of a node causes the loss of the last copy of a chunk, or the deletion of one parity block. As seen in Section 3.3, the necessary condition that must hold for a successfully reconstruction is that at least $g$ out of the $g + h$ blocks including the redundancy are available: when the event triggers, all the $g+h$ blocks are reconstructed. For what concerns data chunks, the algorithm chooses a valid node using the same procedure used for the copy event: either the random or the power-of-choice policy can be applied to select the new node where the chunk will be restored. For the sake of simplicity, parity blocks are always restored on the same node they were before the failure.

### *5.5. Block transfer*

Users access the blocks at a predefined rate. We assume that the network is correctly dimensioned and provides enough bandwidth to allow as many parallel transfers as required by the proposed technique. We suppose that the transfer time of a block from node $i$ is distributed according to a random variable $T_i$, which is independent from the one of any other node. Let us consider a chunk $b$ which is distributed over a set of nodes $N(b)$. If $|N(b)| > 0$ (i.e. there is at least one copy of the data), we suppose that the chunk is requested in parallel from all the nodes $j \in N(b)$. In this case, the transfer time distribution of block $b$, $X(b)$, can be computed as:

$$X(b) = \min_{i \in N(b)} T_i \tag{2}$$

If instead the block is not directly available in any node (that is, $|N(b)| = 0$), it must be reconstructed starting from the parity groups. In this case, we compute the transfer time required to collect the minimum number of chunks and parity blocks necessary for its reconstruction. Let us focus on the case when $p = 1$ (i.e. there is only one parity group): in this case the requested chunk can be restored if at least $g$ blocks from the group, either chunks or parity blocks, are transferred. The simulator determines the time required to download each chunk $b_j$, using $X(b_j)$, and each parity block on node $i$, using $T_i$, then it stops when $g$ blocks are available. Note that the procedure can become recursive if some of the required chunks are not directly available and must be reconstructed as well. If $p > 1$, the previous algorithm is repeated for every parity group, and the minimum is chosen.

### *5.6. Event List Optimization*

In order to speed up the simulation, three optimization lists are defined. These lists aim to provide the chunks that are valid for scheduling of duplication, redundancy computation, and reconstruction. Indeed, the first list includes all the chunks that are present in the system with less than $k$ copies; the second list accounts for the chunks that need their parity groups to be built. Finally, the third one lists the chunks that are lost and require to be reconstructed by the parity groups present in the system. The optimization makes the event scheduling faster, reducing the simulation run time to few seconds instead of several minutes.

## 6. Results

The considered system is evaluated through a set of simulation experiments. In particular, we consider a system composed by $n = 40$ nodes and $m = 200$ chunks. Even if the numbers seem to be a little limited for the considered scenario, results obtained in this setting can provide meaningful insights of the behavior of a much larger system, as it will be proven at the end of the section. To study the system in an unreliable environment, we have chosen a node failure rate $d = 0.01$ (one failure every 100 hours), a duplication rate of $r = 0.1$ (10 hours to complete a copy), a redundancy computation of $q = 0.1$ (redundancy is computed every ten hours) and a reconstruction rate $s = 0.2$ (in average, 5 hours are required to identify a missing chunk and reconstruct it). Results have been obtained by averaging 200 runs and computing the confidence intervals. For example, Figure 14 shows the 95% confidence intervals for the average number of chunks and blocks as a function of time for the configuration with $k = 2$ and $p = 0$. As it can be seen, confidence intervals are very tight: to simplify the presentation, we will not show them in the rest of the paper.

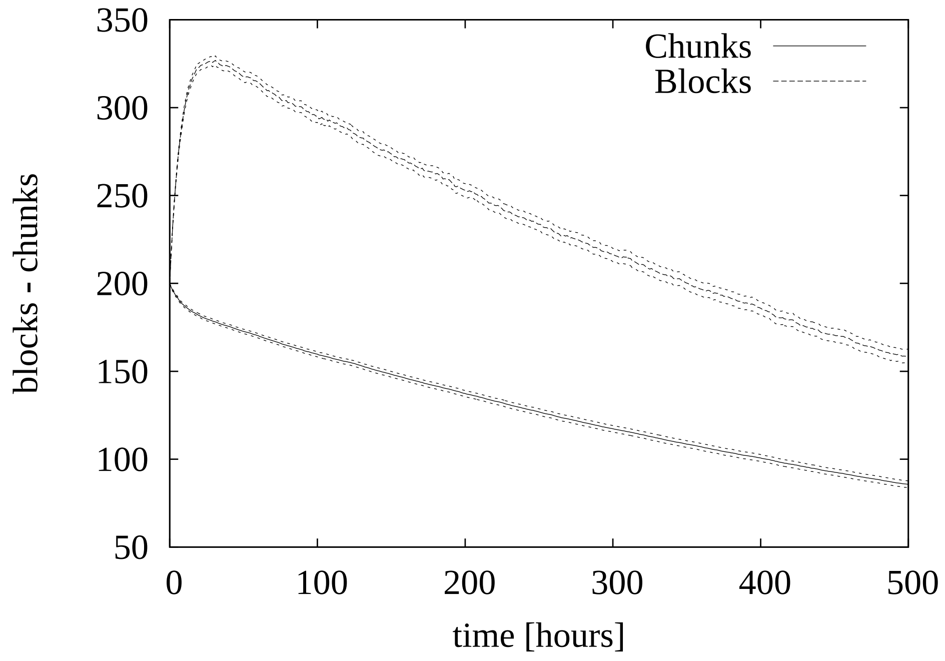

Figure 14: Confidence intervals of the average number of chunks and blocks as function of time for the configuration with $k = 2$ and $p = 0$.

## *6.1. Evolution of the average number of blocks and chunks*

We start considering only block replications for different values of the number of copies. Figure 15 shows the evolution of the number of available chunks in the system as a function of time. The number of chunks starts with $m$, since at time $t = 0$ all the chunks are available, and gradually reduces, with a rate that becomes smaller as the replication factor increases. Figure 16 considers the number of used blocks at the start of the system. As it can be seen, since initially there is only one copy per chunk, the plot starts with $m$ for all the considered values of $k$. However, since the copy mechanism starts copying the chunks, the number of blocks increases to $k \cdot m$ in the initial evolution of the system. This limit however is not reached due to the blocks that fail before the first copy has been completed. This shows that if we start the evolution of the system with just a single block per chunk, the system is subject to a high number of early failures. Figure 17 focuses on larger time scales: in this case the evolution of the average number of blocks tends to correspond to the one of the average number of chunks, multiplied by the average number of blocks per chunk.

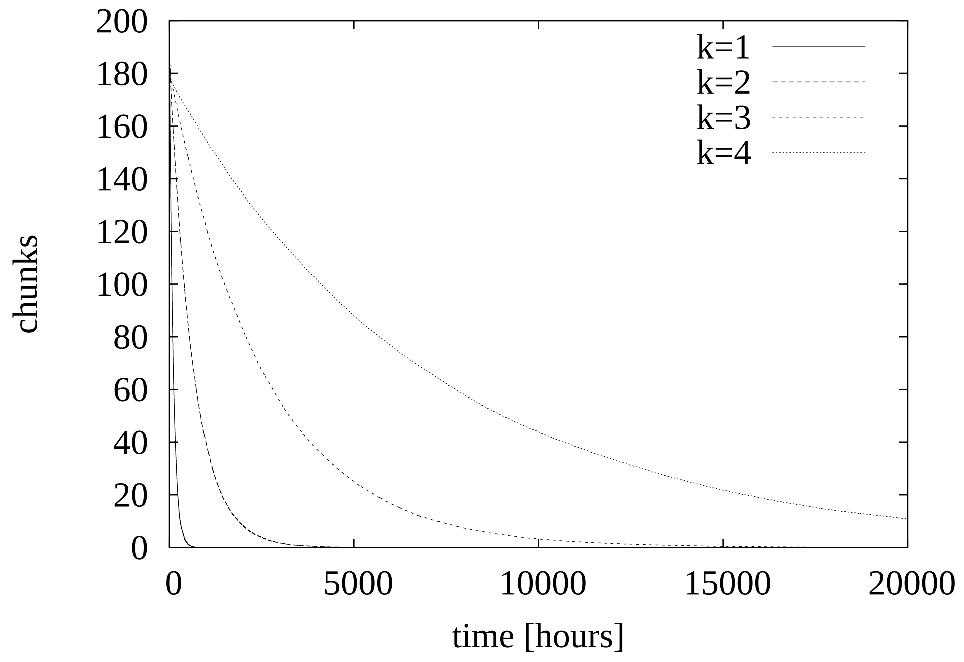

Figure 15: Blocks and chunks availability as function of time: average number of chunks.

## *6.2. Reliability*

The evolution of the number of chunks in the system is also an indication of its reliability, as shown in Figure 18. In the latter case, the reliability has been computed by sorting the failure times of the blocks, rather than counting the number of failures at a given time instant. It is interesting to see the evolution of the hazard rate of the blocks (numerically computed

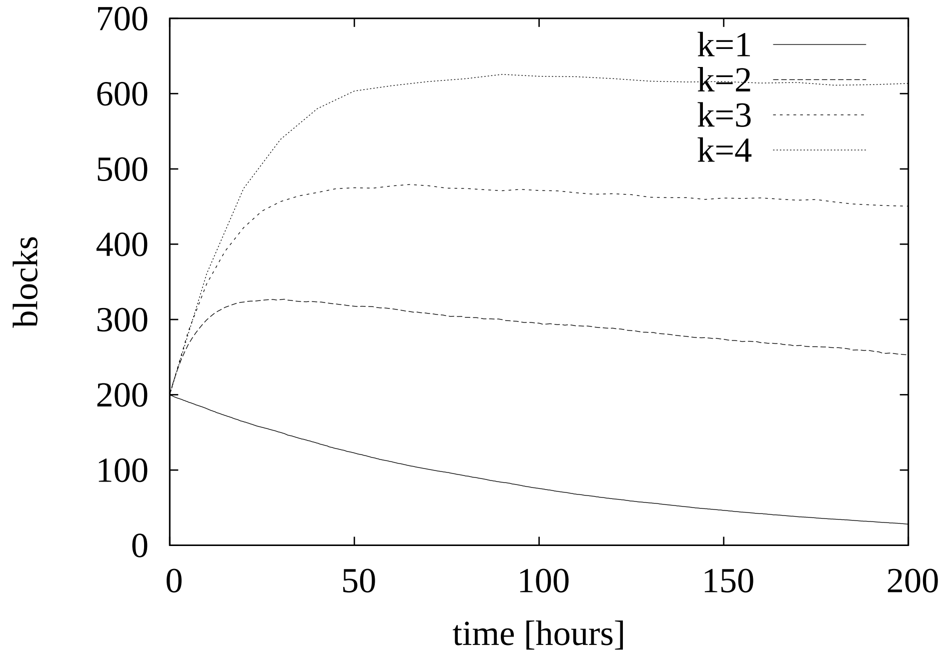

Figure 16: Blocks and chunks availability as function of time: average number of blocks (small time scale).

from the reliability shown in Figure 18), as represented in Figure 19: as it can be noticed, the life of blocks is characterized by a DFR (*Decreasing Failure Rate*) behavior, due to the high number of early failures of the system. After the initial period of time, the system tends to evolve with a CFR (*Constant Failure Rate*) behavior: this is due to the Poisson process that characterizes the failures of the nodes. Note that for $k = 1$, since no copy is performed, the reliability of the chunks corresponds to the one of the node, as shown by the corresponding horizontal line in the hazard rate.

## *6.3. Introducing Erasure Codes*

We then consider also cases where $p = 1$ parity groups are used. Initially we focus on a system where each chunk is replicated $k = 2$ times, the groups include $g = 4$ chunks and are protected by $h = 2$ redundancy information. In this case the chunks evolve among 6 different states. Chunks can still require both copies and redundancy information (state `Incomplete`). A chunk can be in a state where all copies have been done, but redundancies are still being

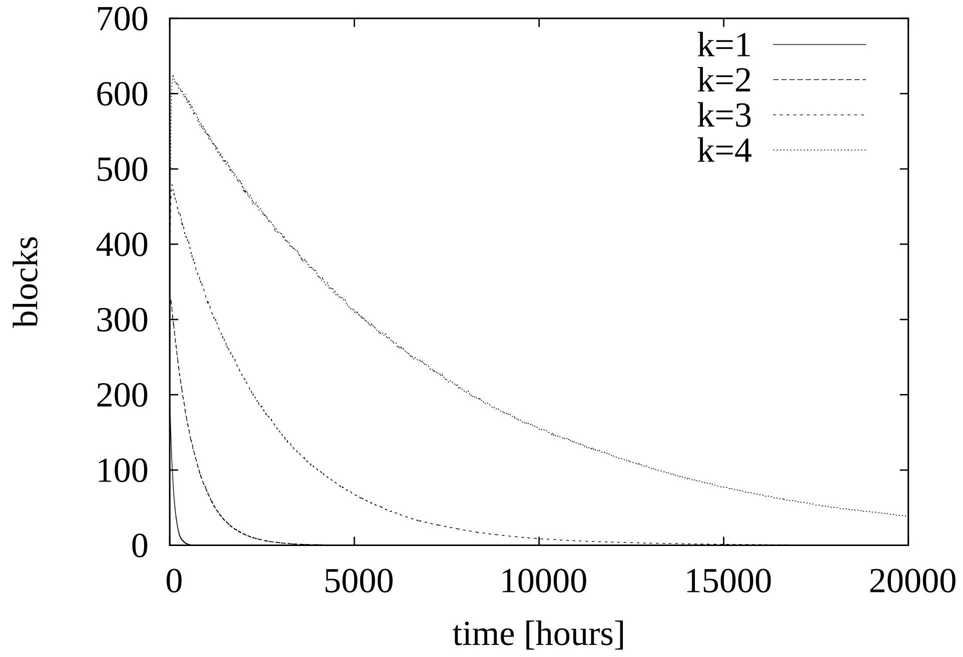

Figure 17: Blocks and chunks availability as function of time: average number of blocks (large time scale).

computed (state `Copies done`), or on the opposite redundancies have been computed but some copies are still missing (state `Red. done`). When all copies and redundancy information have been computed, the chunk jumps in a state `All done`. This is the case in which the chunk has the maximum protection: the chunk will jump back in one of the previous state when the failure of node deletes some of the copies or destroys parity groups that can no longer be effective since not enough blocks to reconstruct the missing data remains. When a chunk has no more copies available, but it can still be saved using the parity groups, it jumps in state `Reconstructing`. If the chunk can no longer be saved, it ends in the `Dead` state. Figure 20 shows the evolution for the initial life of the system. The probability of being in state `Incomplete` tends to zero as copies are being done and parity groups are being constructed. The probability of having completed the copies and the redundancy information grows rapidly at the beginning, but then they tend to fall up to an almost constant level. This is due to node failures, that destroys some of the copies or of the parities built for a chunk. At around

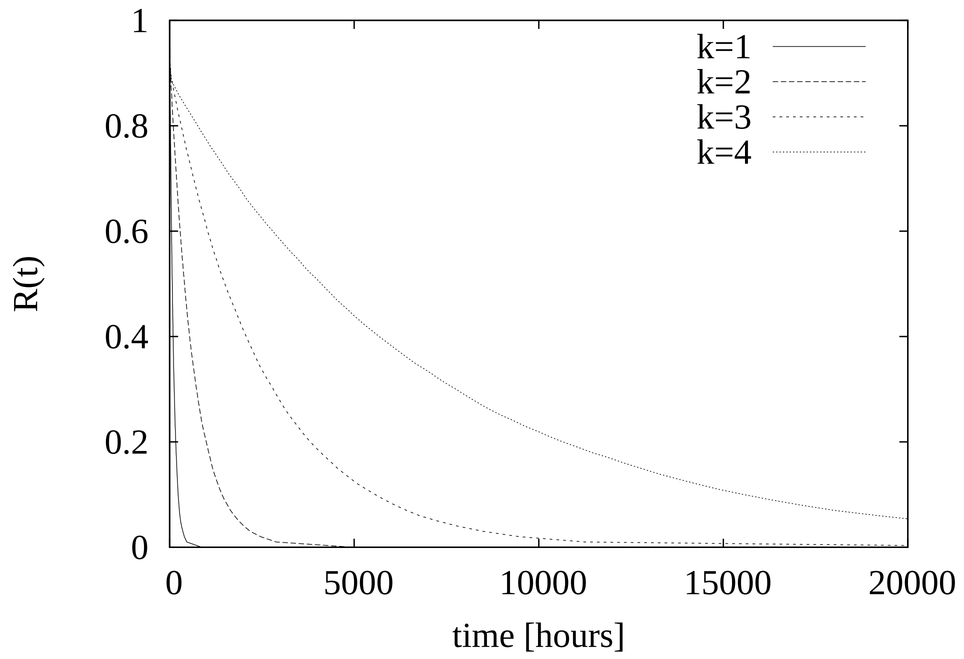

Figure 18: Reliability of the system for different maximum number of copies $k$ when no redundancy is used.

$t = 40$ most of the chunks are in the `All done` state, representing the fact that they have completed all the copies and computed all the required parities to protect them. The reconstruction state starts to slowly be more present as time increases. Figure 21 shows the average state of the blocks for larger time scales: in this case, the probability of loosing a chunk increases, thus reducing the probability of being in the other states. It is interesting to note that second most probable state is the one in which redundancy information are done, but copies still needs to be completed (state `Red. done`). This is because as soon as a node fails, it is necessary to restore the number of available copies of each of its chunks to $k$. As time passes, the probability of having a cascade of failures that leads to a non-recoverable state increases, and eventually all the chunks will be lost.

## *6.4. Parity group size*

Let us move the focus on the effects of the number of blocks in the group $g$ and on the total number of redundancy information $h$, for a system where

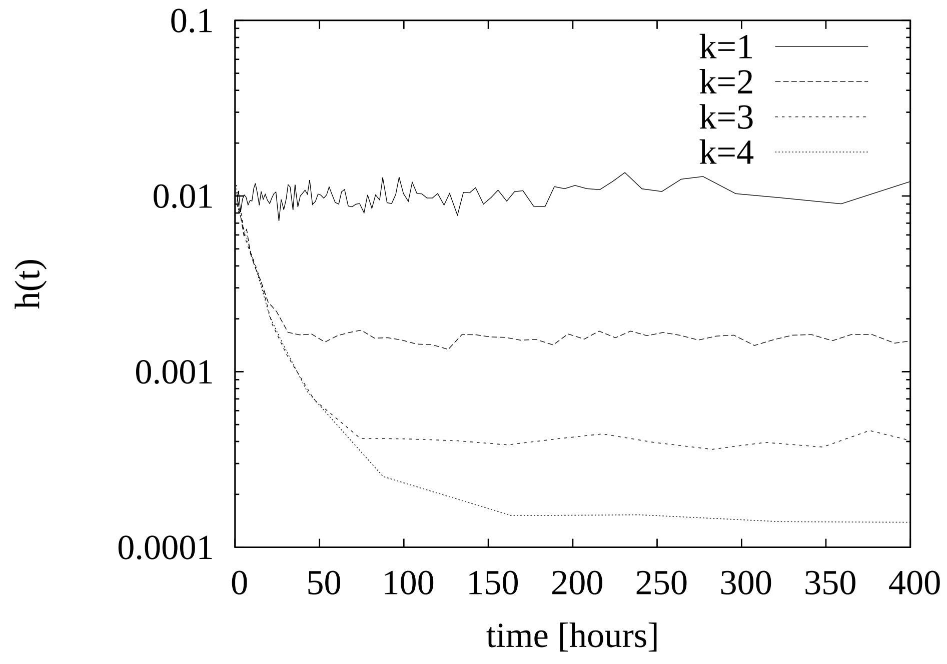

Figure 19: Hazard rate of the system for different maximum number of copies $k$ when no redundancy is used.

a single parity group $p = 1$ is used, and blocks are introduced in a single copy (that is, $k = 1$). We start considering an increasing group size $g = 2 \ldots 12$ for two different numbers of parity information available in the group: Figure 22 shows the reliability for $h = 1$ and Figure 23 for $h = 2$. In both cases, curve $p = 0$ represents the case where no parity mechanism is used. As it can be seen, as the size of the parity group $g$ increases, the reconstruction mechanism becomes less effective reducing the reliability of the chunks. In particular, for groups with $g = 8$ the effectiveness of the technique remains very limited for $h = 1$, while it can still provide some protection for $h = 2$.

Figure 24 shows the effect of changing the number of parity blocks $h$ used in a group of $g = 4$ chunks. As expected, increasing the number of parity information greatly increases the reliability of the blocks. It is interesting to compare the curve of $h = 4$ with the one with $k = 2$, that represents a system with no parity information $p = 0$ and two copies per chunk. Both configurations have an overhead of the same number of blocks (since $g = 4$); however, the parity mechanism provides a higher reliability, at the expense

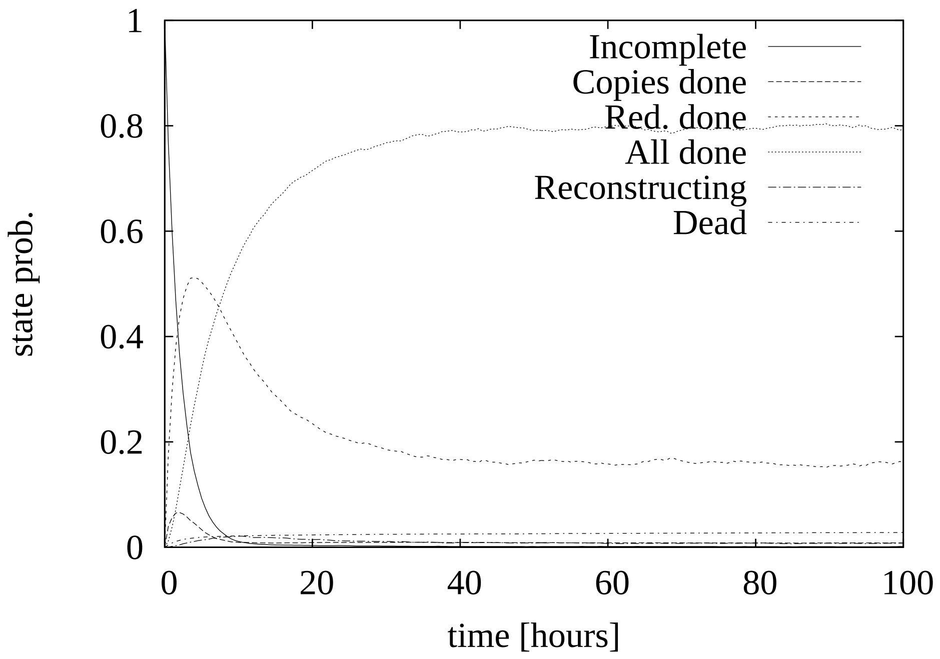

Figure 20: State evolution of the system at the initial life of the system.

of a more complex reconstruction of missing data. This is even more evident for the case with $h = 8$ and $k = 3$ (three copies of one block, and no parities, with the same average blocks requirement of $h = 8$): in fact, the reliability of the curve with $k = 3$ is still less than the one with $h = 3$.

If no performance parameter has to be taken into account, this result suggest that the erasure codes technique is much more effective than the replication mechanism with the same overhead in terms of blocks storage.

## *6.5. Nodes occupancy distribution*

Next we focus on the node occupancy of the different policies. Figure 25 shows the distribution of the number of nodes having a given maximum number of parities, data blocks and total number of blocks (data + parities) in the system for $k = 2$, $p = 1$, $g = 4$ and $h = 2$. The capacity of the nodes has been set to $\nu_{\max} = 2.5 \cdot (m/n) \cdot k = 25$. In particular, during the simulation the maximum total number of blocks being stored in a node during the temporal evolution is computed, and the distribution is derived at the end of the run. This provide us with an idea of the maximum occupancy

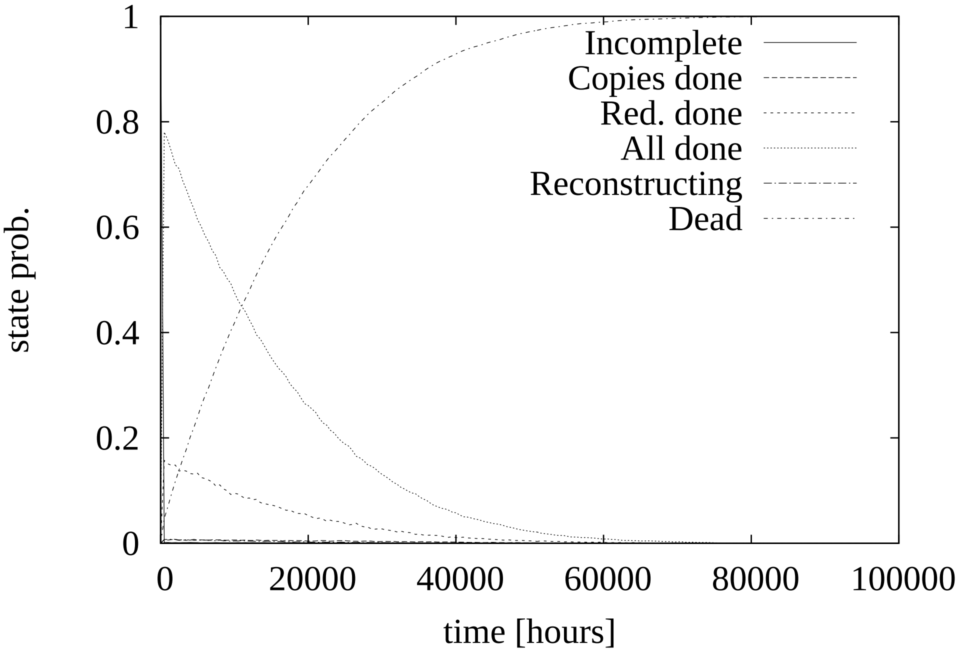

Figure 21: State evolution of the system at large timescale.

that a given configuration can require from a node. As it can be seen, a large number of nodes tends to have the same maximum occupancy for what concerns data block, while redundancy information is more evenly spread. The total maximum number of blocks (including both data and redundancy) tends to be evenly spread since it is the sum of the two.

Figure 26 shows instead the distribution of the maximum total number of blocks for different configurations of parity groups and replications. As it can be seen, the maximum number of copies $k$ shifts the distribution to right since it corresponds to an increased occupancy. Note that the peak at 50 blocks of the case with $k = 4$ corresponds to the maximum capacity of the node, which with such settings is $\nu_{\max} = 50$. The increase in $h$ instead shifts less effectively the distribution (since the redundancy block is shared among all the nodes in the group), but it increases its variance: this is due to the restriction that imposes that a redundant block can be placed only on a node that does not already host any block belonging to any copy of the chunks included in the group, making the selection of the destination node more complex.

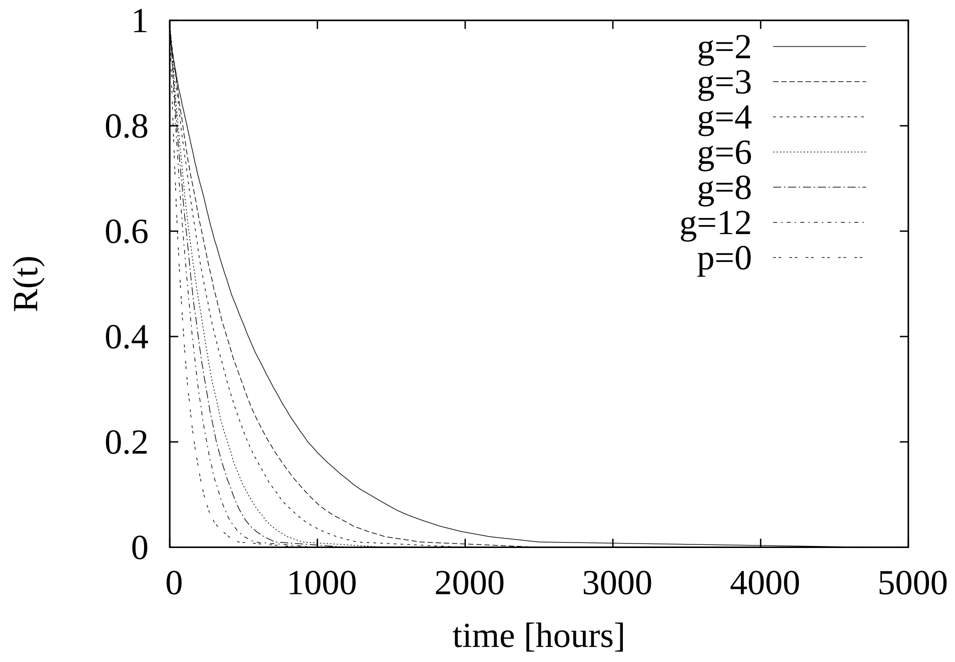

Figure 22: Reliability of the system for $p = 1$: varying the group size with $h = 1$.

Finally, figure 27 considers the case with $g = 4$ and $h = 2$ for $k = 1$ and $k = 2$ under two blocks distribution policies: random, and power-of-choice. Indeed the power-of-choice policy, by selecting the best of two randomly chosen nodes, is able to reduce the maximum number of blocks required during the simulations. This becomes particularly evident for the case with $k = 3$, where the power-of-choice technique is able to reduce the maximum node occupation of about 40%.

## *6.6. Effects of the copy, redundancy computation and reconstruction rates*

We then examine the effects of the copy, redundancy computation and reconstruction rates on the MTTF (mean time to failure) of the system. We focus on a system where each chunk has $k = 3$ copies, and it belongs to a parity group composed of $g = 4$ chunks with $h = 2$ redundancy blocks. Figure 28 shows the results for a fixed node failure rate $d = 0.01$ failure per hour. The redundancy computation rate $q$ has very small influence on the system: since $h = 2$, each group requires the loss of at least three chunks to become invalid. Since each chunk is replicated 3 times, the event of failure

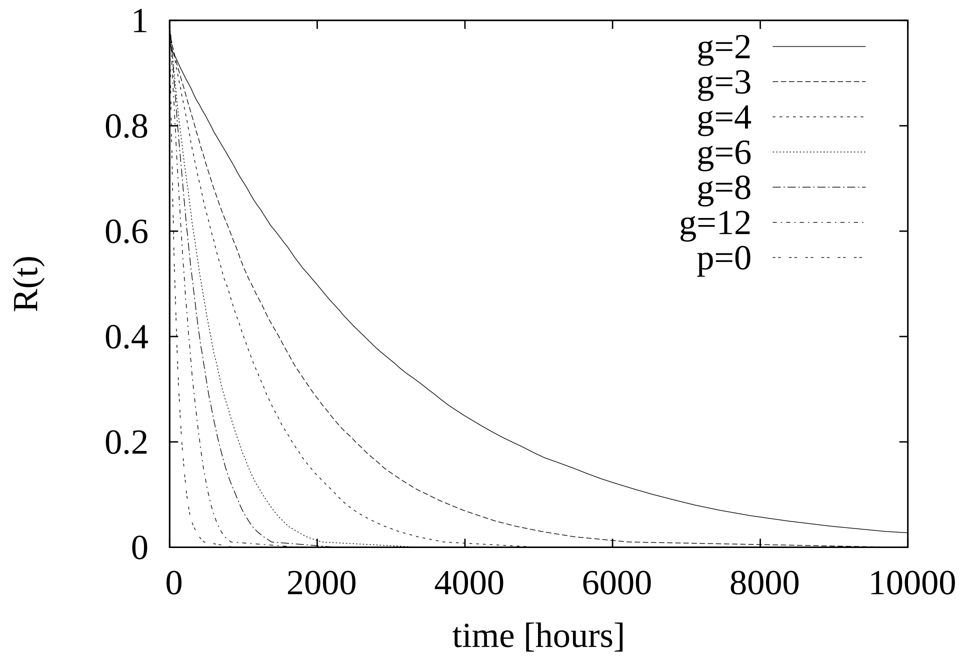

Figure 23: Reliability of the system for $p = 1$: varying the group size for $h = 2$.

of a parity group is very rare, thus the rate at which it is built does not affect too much the reliability of the system. The configuration is instead very sensible to the copy rate $r$: a decrease of one order of magnitude in the time required to copy one block results in an increase of almost three order of magnitude in the MTTF. The faster is the replication mechanism, the smaller is the probability of loosing all the blocks were a chunk is stored. Moreover, the erasure codes technique reduces even further the probability of not having anymore available copies of the chunk. More interesting is the influence of chunk reconstruction rate $s$, where the MTTF experience a minimum. Indeed, it seems to be counter-intuitive that there is an increase of the MTTF when also the time required to reconstruct a chunk increases. This however happens only when the reconstruction rate becomes very close to the node failure rate. In the considered configuration the parity group can sustain up to two failures, but the repair process is started as soon as one failure occurs. This means that if another failure occurs during the reconstruction, it will benefit from the restoration process initiated during the previous failure and it will be repaired much sooner, leading to an increase

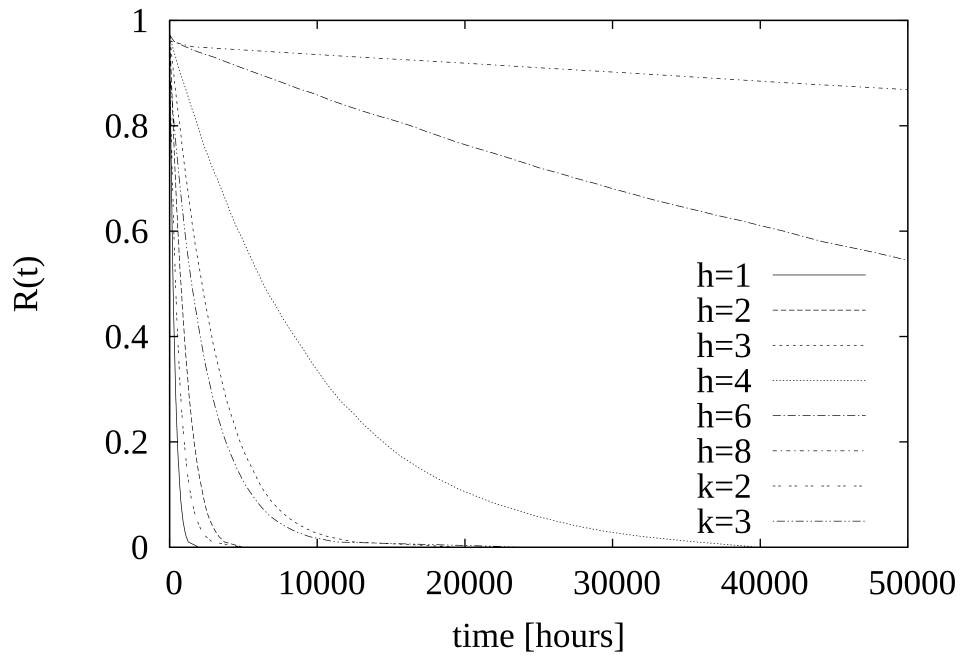

Figure 24: Reliability of the system for $p = 1$: varying the number of parities for $g = 4$.

in the MTTF.

## *6.7. Transfer time distribution*

We now take into account the performance of the various configurations, to determine the impact of parameters on the transfer time required to either upload or download a chunk. We can compute the transfer time distribution for different configurations, as shown in Figure 29. In our scenario, we imagine that if a chunk is available on more than one node, the user can access it from the one with the shorter response time. We suppose that the user sends its request in parallel to all the nodes having the chunk, and then that she reads or write it on the node that answered first. We suppose that chunks at requested at a rate $c = 0.02$ requests per hour, and that the time required to access a block follows a Normal distribution (see Dandoush et al. (2009)). We set the mean transfer time to $\mu = 100$ *msec.*, and its standard deviation $\sigma = 25$ *msec.* The actual transfer time is then computed using the technique described in Section 5.5. As we can see in Figure 29, the most sensitive parameter is the number of replicas $k$. Since the transfer time cor-

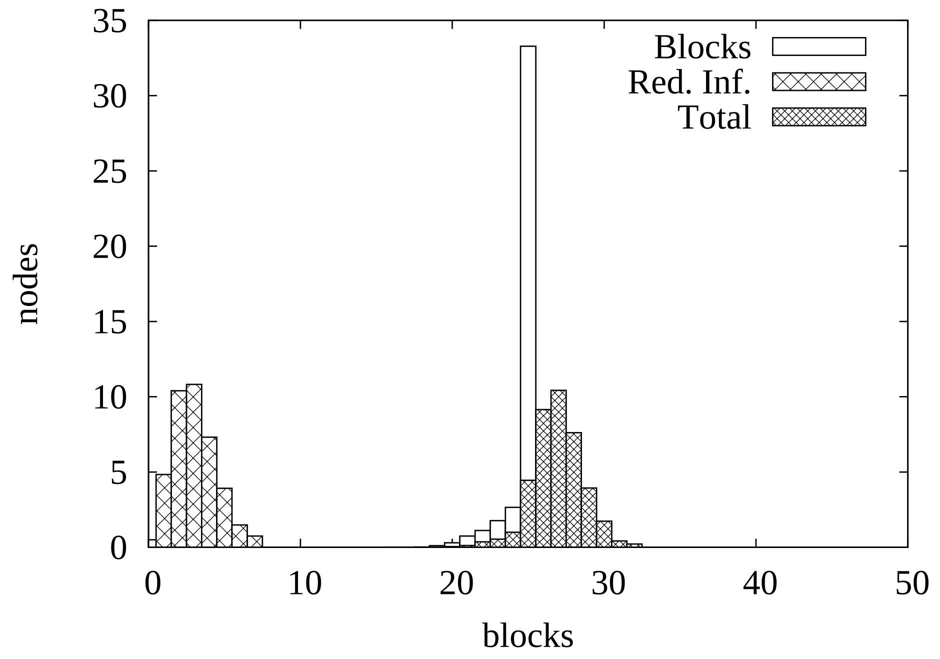

Figure 25: Distribution of the number of nodes having a given maximum number of parities, data blocks and total number of blocks (data + parities) in the system for $k = 2$, $p = 1$, $g = 4$ and $h = 2$.

responds to the minimum of $k$ distributions, a larger value of $k$ results in a reduced response time. It is interesting to see that also the variance of the response time reduces as $k$ increases. The size of the groups and the number of redundancy blocks used seem to have a marginal role in the transfer time distribution: this is because the increased transfer time required to access a chunk when it must be reconstructed from the blocks that have survived in a group must be considered only for the cases in which the data is not directly available, which are not that frequent.

### *6.8. Considering larger scenarios*

We finally focus on the size of the system by showing that, using proper normalizations, the measures computed for a smaller system are also indicative of the behavior of a larger one. In Figures 30, 31 and 32 we focus on four configurations of increasing size: $n = 20$, $m = 40$; $n = 40$, $m = 200$; $n = 200$, $m = 20000$; and $n = 2000$, $m = 10^6$. Figure 30 shows the reliabil-

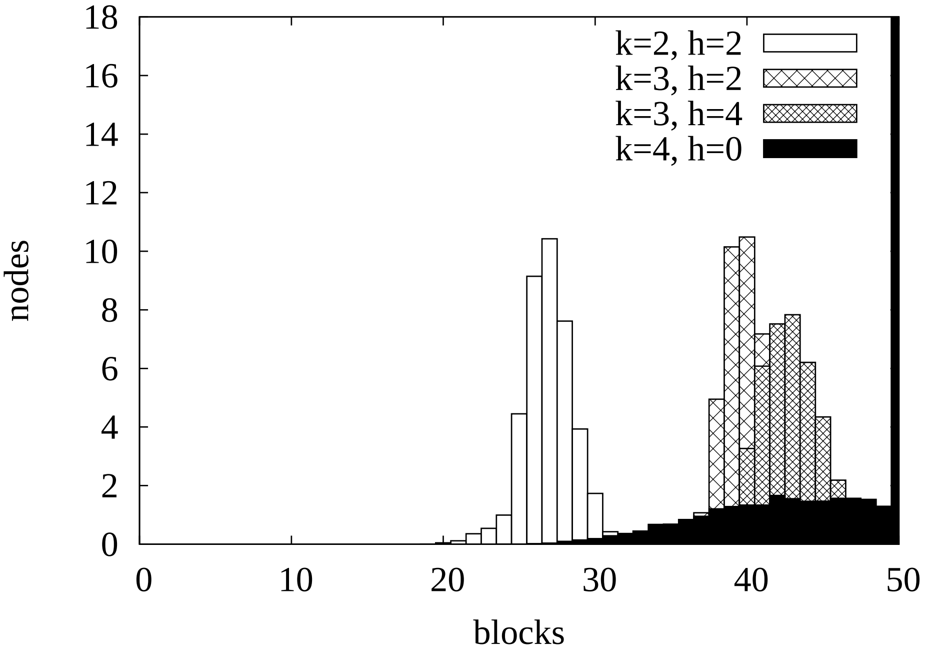

Figure 26: Distribution of the number of nodes having a maximum total number of blocks for different configurations of parity groups and replications.

ity of the blocks for the considered configurations. Since all the nodes and chunks are characterized by the same temporal evolution, the reliabilities of the four models are almost identical regardless of the size. The total number of blocks in the system is proportional to the initial number of chunks $m$. In order to compare models of different size, we have to normalize the measure dividing the average number of blocks by $m$. Figure 31 compares the normalized number of blocks in the four configurations: as it can be seen, despite the different sizes of the considered cases, the four curves are almost identical. The distribution of the maximum total number of blocks can be compared after a more complex normalization procedure, as shown in Figure 32. The $x$ axis, which corresponds to the maximum total number of blocks in a node, should be normalized by the average number of chunks in a node: $m/n$. The $y$ axis, that counts the number of blocks in a node, should instead be normalized by the average number of nodes having an average number of chunks, that is it should be divided by $n/(m/n)$. The four normalized curves match very closely: this proves that the technique proposed in this paper

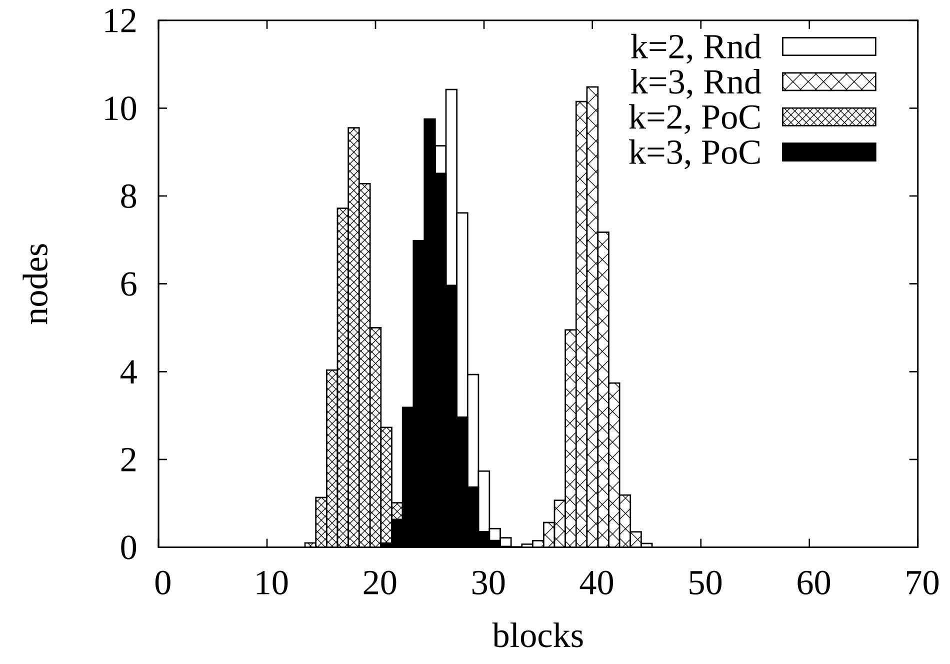

Figure 27: Distribution of the number of nodes: comparison of the random and power of choice block distribution policies.

can be used also to study very large systems, since results can be scaled with simple algebraic expressions.

## *6.9. Results summary*

In Table 3 we summarize some of the results that can be drawn from the previous analysis. The first four columns report the parameters of some of the considered configurations, as defined in Table 2. The fifth column shows the corresponding MTTF of the configuration. The sixth column reports the target occupancy (*Targ. occ.*): the average number of blocks that each node would have to store if they would be uniformly spread. In particular it can be computed by dividing the result of Equation 1 by the number of nodes $n$, that is:

$$\text{Targ. Occ.} = \left\lceil \frac{m}{n} \cdot \left( k + p \cdot \frac{h}{g} \right) \right\rceil \tag{3}$$

The seventh column proposes the average maximum occupancy (*Av. max. occ.*), which is determined as the average of the nodes per block distributions

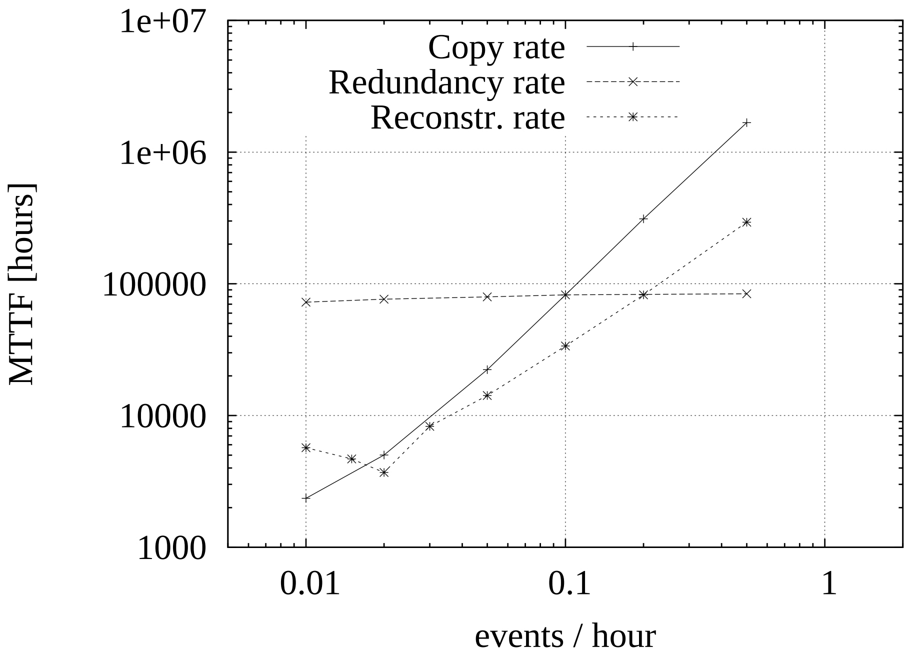

Figure 28: Effects of the copy rate, parity building rate and reconstruction rate on the MTTF of the system.

presented in Figures 25, 26 and 27. The last column reports the average transfer time (*Av. tr. time*), which is computed from the distributions presented in Figure 29. Looking at the MTTF column, we can conclude that the parity mechanism introduced by erasure codes is more effective than replication in increasing the reliability of the system using the same amount of blocks. However, looking at the average transfer time column, we can see that the erasure codes alone cannot improve the efficiency of the system. Mixing replication and erasure codes seems to be a promising approach to both increase the reliability and the performance of the system, without requiring a large overhead, as it can be seen from the sixth and seventh column of the table.

## 7. Conclusions

We model a storage system by simulating the management layer based on a redundancy approach. The simulator is able to scale to different ar-

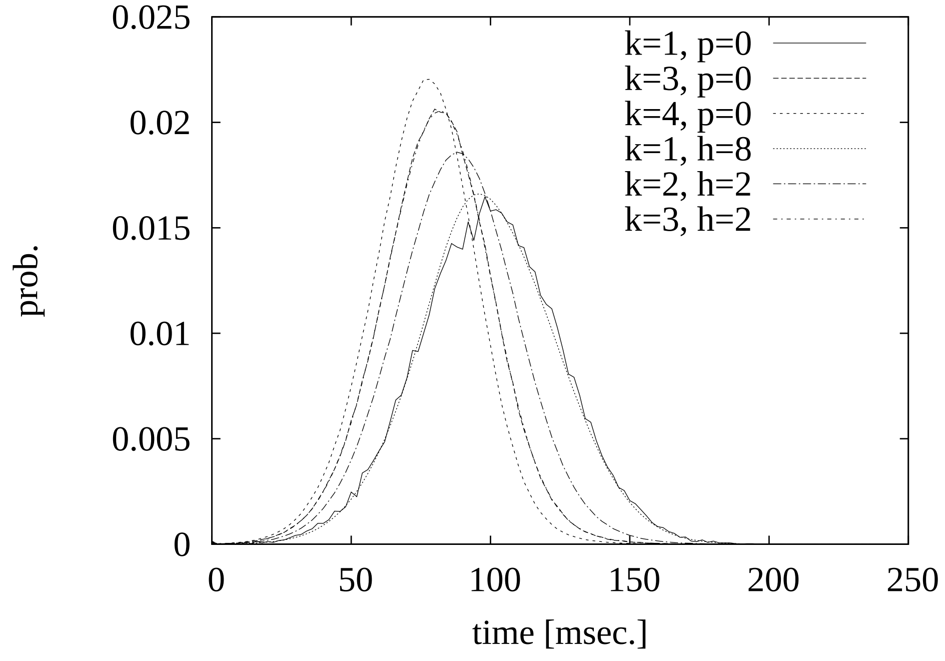

Figure 29: Transfer time distributions for different configurations of the system.

chitectures and large number of nodes and data. By using the proposed model, we derived both reliability and performance indexes. We point out that mixing replication and erasure codes makes the reconstruction mechanism more effective improving the data availability. Moreover, this work shows that by tuning appropriately the replication parameters the transfer time is also reduced. The flexibility of the model allows to easily consider different strategies and scenarios.

Future works include the inclusion of other features, such as the variation of the storage pool, the inclusion of temporary unavailability, the explicit introduction of the effects of network architecture and structure, the analysis of bandwidth limitations due to network overload, the evaluation in alternative situations, such as low availability of storage nodes or high delays between a failure in a storage node and the reaction, to investigate the behavior of the proposed solution when multiple failures can challenge the resilience to faults.

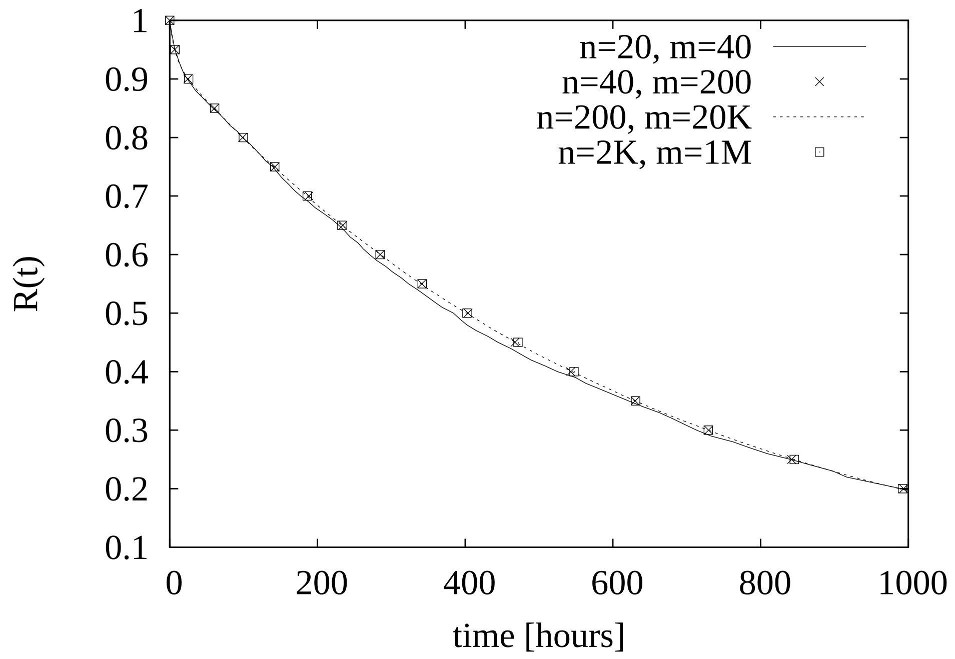

Figure 30: Configuration with $k = 2$ and $p = 0$ under different number of nodes $n$ and chunks $m$ in the system: normalized distribution of the maximum total number of blocks.

## References

Aguilera, M., Janakiraman, R., Xu, L., June 2005. Using erasure codes efficiently for storage in a distributed system. In: Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on. pp. 336–345.

Barbierato, E., Gribaudo, M., Iacono, M., 2014. Performance evaluation of nosql big-data applications using multi-formalism models. Future Generation Computer Systems 37 (0), 345–353.
URL `http://www.sciencedirect.com/science/article/pii/S0167739X14000028`

Barbierato, E., Gribaudo, M., Iacono, M., 2015 (to appear). Modeling and evaluating the effects of big data storage resource allocation in global scale cloud architectures. International Journal of Data Warehousing and Mining.

Borthakur, D., 2008. Hdfs architecture guide. HADOOP APACHE

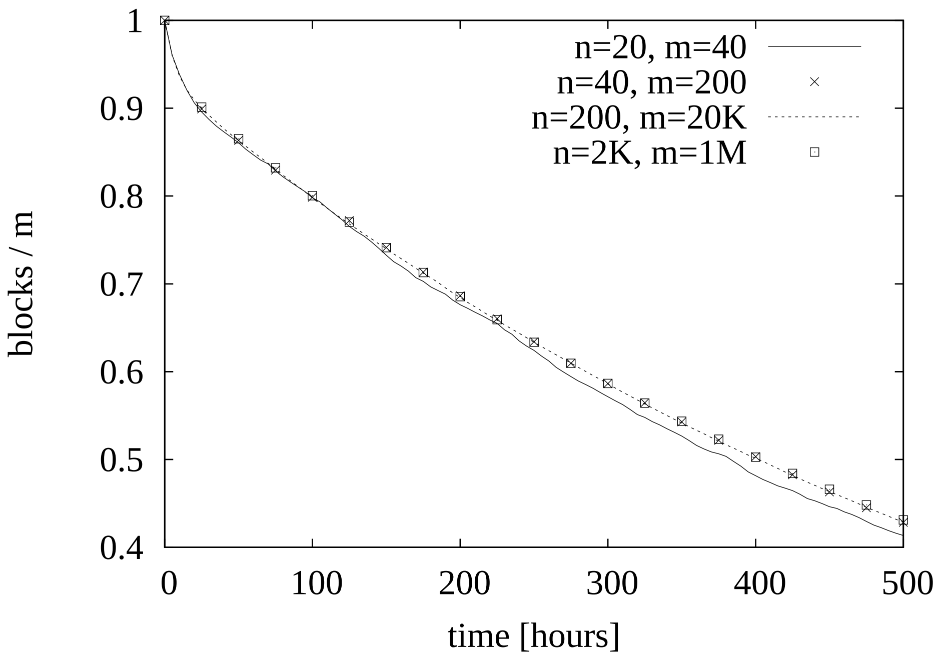

Figure 31: Configuration with $k = 2$ and $p = 0$ under different number of nodes $n$ and chunks $m$ in the system: normalized average number of blocks.

PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf.

Calder, B., J.Wang, A. O., Nilakantan, N., Skjolsvold, A., McKelvie, S., Xu, Y., Srivastav, S., Wu, J., et al, H. S., 2011. Windows azure storage: a highly available cloud storage service with strong consistency. In: In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles.

Castiglione, A., Gribaudo, M., Iacono, M., Palmieri, F., 2014a. Exploiting mean field analysis to model performances of big data architectures. Future Generation Computer Systems 37 (0), 203–211.
URL `http://www.sciencedirect.com/science/article/pii/S0167739X13001611`

Castiglione, A., Gribaudo, M., Iacono, M., Palmieri, F., 2014b. Modeling performances of concurrent big data applications. Software: Practice and

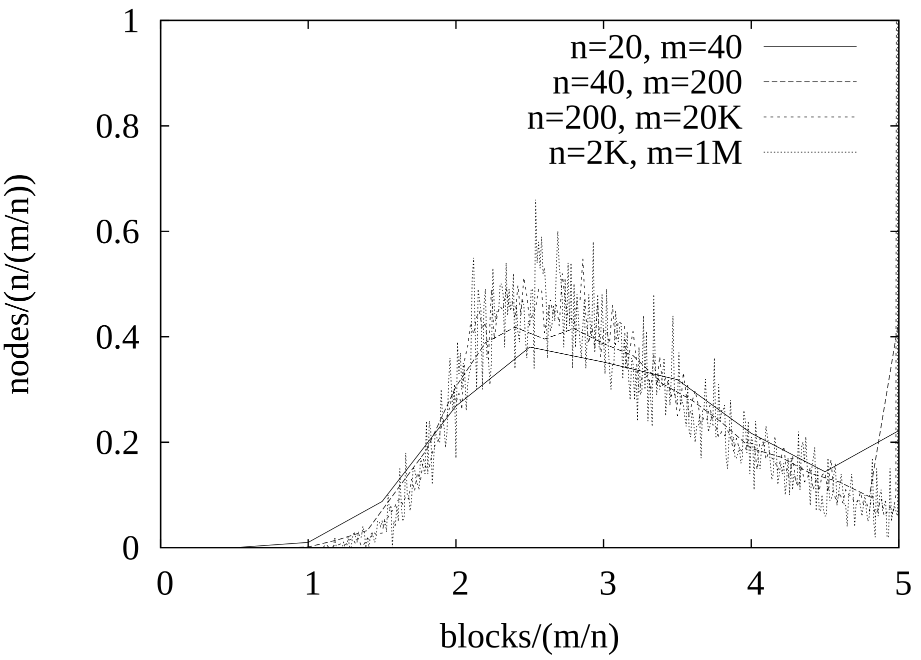

Figure 32: Configuration with $k = 2$ and $p = 0$ under different number of nodes $n$ and chunks $m$ in the system: normalized distribution of the maximum total number of blocks.

Experience, n/a–n/a.
URL `http://dx.doi.org/10.1002/spe.2269`

Cerotti, D., Gribaudo, M., Iacono, M., Piazzolla, P., 2015 (to appear). Modeling and analysis of performances for concurrent multithread applications on multicore and gpu systems. Concurrency and computation: practice and experience.

Dandoush, A., Alouf, S., Nain, P., Sept 2009. Simulation analysis of download and recovery processes in p2p storage systems. In: Teletraffic Congress, 2009. ITC 21 2009. 21st International. pp. 1–8.

Distefano, S., Puliafito, A., 2014. Information dependability in distributed systems: The dependable distributed storage system. pp. 3–18.

Friedman, R., Kantor, Y., Kantor, A., 2014. Replicated erasure codes for storage and repair-traffic efficiency. In: 14th IEEE International Conference on Peer-to-Peer Computing, P2P 2014, London, United Kingdom,

Table 3: Results summary

| $k$ | $p$ | $g$ | $h$ | MTTF [hours] | Targ. occ. [blocks] | Av. max. occ. [blocks] | Av. tr. time [msec.] |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 102 | 5 | 5 | 100 |
| 2 | 0 | 0 | 0 | 598 | 10 | 15, 8 | 88.2 |
| 3 | 0 | 0 | 0 | 2283 | 15 | 30.5 | 81.3 |
| 4 | 0 | 0 | 0 | 6317 | 20 | 45.4 | 76.8 |
| 1 | 1 | 4 | 1 | 320 | 7 | 8 | 100.8 |
| 1 | 1 | 4 | 2 | 999 | 8 | 11.9 | 100.7 |
| 1 | 1 | 4 | 8 | 359531 | 15 | 22.6 | 99.6 |
| 2 | 1 | 4 | 2 | 16287 | 13 | 27 | 88.4 |
| 3 | 1 | 4 | 2 | 82531 | 18 | 39.8 | 81, 4 |
| 3 | 1 | 4 | 4 | 417187 | 20 | 42.8 | 81.46 |

September 9-11, 2014, Proceedings. pp. 1–10.
URL `http://dx.doi.org/10.1109/P2P.2014.6934310`

Ghemawat, S., Gobioff, H., Leung, S.-T., 2003. The google file system. In: In: Proceedings of the 9th ACM symposium on Operating systems principles. New York.

Kameyama, H., Sato, Y., March 2007. Erasure codes with small overhead factor and their distributed storage applications. In: Information Sciences and Systems, 2007. CISS '07. 41st Annual Conference on. pp. 80–85.

Lian, Q., Chen, W., Zhang, Z., June 2005. On the impact of replica placement to the reliability of distributed brick storage systems. In: Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on. pp. 187–196.

Mitzenmacher, M., Richa, A. W., Sitaraman, R., 2000. The power of two random choices: A survey of techniques and results. Kluwer.

Plank, J. S., Sep. 1997. A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. Softw. Pract. Exper. 27 (9), 995–1012.
URL `http://dx.doi.org/10.1002/(SICI)1097-024X(199709)27:9<995::AID-SPE111>3.3.`

Rodrigues, R., Liskov, B., Feb. 2005. High availability in dhts: Erasure coding vs. replication. In: Peer-to-Peer Systems IV 4th International Workshop IPTPS 2005. Ithaca, New York.

Sathiamoorthy, M., Asteris, M., Papailiopoulos, D., Dimakis, A. G., Vadali, R., Chen, S., Borthakur, D., 2013. Xoring elephants: novel erasure codes for big data. In: Proceedings of the 39th international conference on Very Large Data Bases. PVLDB'13. VLDB Endowment, pp. 325–336.
URL http://dl.acm.org/citation.cfm?id=2488335.2488339

Simon, V., Monnet, S., Feuillet, M., Robert, P., Sens, P., May 2014. SPLAD: scattering and placing data replicas to enhance long-term durability. Rapport de recherche RR-8533, INRIA.
URL http://hal.inria.fr/hal-00988374

Weatherspoon, H., Kubiatowicz, J., 2002. Erasure coding vs. replication: A quantitative comparison. In: Revised Papers from the First International Workshop on Peer-to-Peer Systems. IPTPS '01. Springer-Verlag, London, UK, UK, pp. 328–338.
URL http://dl.acm.org/citation.cfm?id=646334.687814

Wu, F., Qiu, T., Chen, Y., Chen, G., 2005. Redundancy schemes for high availability in dhts. In: Pan, Y., Chen, D., Guo, M., Cao, J., Dongarra, J. (Eds.), ISPA. Vol. 3758 of Lecture Notes in Computer Science. Springer, pp. 990–1000.
URL http://dblp.uni-trier.de/db/conf/ispa/ispa2005.htmlWuQCC05

Xiang, Y., Lan, T., Aggarwal, V., Chen, Y. F. R., Sep. 2014. Joint latency and cost optimization for erasurecoded data center storage. SIGMETRICS Perform. Eval. Rev. 42 (2), 3–14.
URL http://doi.acm.org/10.1145/2667522.2667524

Xu, L., Cipar, J., Krevat, E., Tumanov, A., Gupta, N., Kozuch, M. A., Ganger, G. R., Oct. 2014. Agility and performance in elastic distributed storage. Trans. Storage 10 (4), 16:1–16:27.
URL http://doi.acm.org/10.1145/2668129

Yan, F., Riska, A., Smirni, E., June 2012. Fast eventual consistency with performance guarantees for distributed storage. In: Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on. pp. 23–28.

Mauro Iacono is a tenured Researcher and Assistant Professor in Computing Systems at Seconda Università degli Studi di Napoli. He received a master degree in computer engineering from Università "Federico II" di Napoli and a PhD degree in electronic engineering from Seconda Università degli Studi di Napoli.  He published more than 50 peer reviewed scientific papers and has served as chairman, committee member and referee for many conferences, and as guest editor, editorial board member and referee for several journals. His research is mainly centered on the field of performance modeling of complex computer-based systems, with a special attention for multiformalism modeling techniques, and applications to critical systems and infrastructures, Big Data architectures and dependable systems. More at http://www.mauroiacono.com.

Daniele Manini is a Researcher and Assistant Professor at the Computer Science Department of the University of Turin. He graduated in Computer Science and received the qualification of Ph.D. in Computer Science from the University of Turin. Daniele Manini focused his research activities on the development of analytical and simulative models used for describing and studying multi-domain systems. In particular, this activity has been addressed toward three fields: performance evaluation of communication networks, the analysis of biological systems by formal methods, and the development of models describing typical phenomena of complex. Daniele Manini participated to national projects FIRB – PERF e PRIN – PATTERN, and in MASP project in the context of "Polo di Innovazione ICT". He is recently involved in the COST Action "Random Network Coding and Designs over GF(q)" (2012-16).

Marco Gribaudo is a tenured researcher and assistant professor at Politecnico di Milano. He works in the performance evaluation group. His current research interests are multi-formalism modeling, queueing networks, mean-field analysis and spatial models. The main applications to which the previous methodologies are applied come from cloud computing, multi-core architectures and wireless sensor networks.