P2P Storage Systems: Study of Different Placement Policies*

Stéphane Caron[1], Frédéric Giroire[2], Dorian Mazauric[3], Julian Monteiro[4], and Stéphane Pérennes[2]

[1]Technicolor Palo Alto Research Center, Palo Alto, US.
[2]COATI, CNRS, I3S, University of Nice Sophia Antipolis, INRIA, Sophia Antipolis, France.
[3]Columbia University, Department of Computer Science, New York, US.
[4]Instituto de Matemática e Estatística - IME-USP, Sao Paulo, Brazil

Abstract

In a P2P storage system using erasure codes, a data block is encoded in many redundancy fragments. These fragments are then sent to distinct peers of the network. In this work, we study the impact of different placement policies of these fragments on the performance of storage systems. Several practical factors (easier control, software reuse, latency) tend to favor data placement strategies that preserve some degree of locality. We compare three policies: two of them are *local*, in which the data are stored in logical neighbors, and the other one, *global*, in which the data are spread randomly in the whole system. We focus on the study of the probability to lose a data block and the bandwidth consumption to maintain such redundancy. We use simulations to show that, without resource constraints, the average values are the same no matter which placement policy is used. However, the variations in the use of bandwidth are much more bursty under the *local* policies. When the bandwidth is limited, these bursty variations induce longer maintenance time and henceforth a higher risk of data loss. We then show that a suitable degree of locality could be introduced in order to combine the efficiency of the global policy with the practical advantages of a local placement. Additionally, we propose a new *external reconstruction* strategy that greatly improves the performance of local placement strategies. Finally, we give analytical methods to estimate the mean time to the occurrence of data loss for the three policies.

Keywords: distributed storage, P2P system, data placement, data life time, mean time to data loss, performance evaluation, Markov chains.

*This work was partially funded by région PACA, by european project EULER.

1 Introduction

The key concept of P2P storage systems is to distribute redundant data among peers to achieve high reliability and fault tolerance at low cost. The addition of redundant data could be done by trivial *Replication* [27, 6, 4], in which identical copies of data are sent to different nodes in the system; or be based on *Erasure Codes* [21, 30], such as Reed Solomon and Tornado, as used by some RAID schemes [22]. When using Erasure Codes, the original user data (e.g. files, raw data, etc.) is cut into data-blocks that are divided into s initial *fragments* (or pieces). The encoding scheme produces $s + r$ fragments that can tolerate r failures. In other words, the original data-block can be recovered from any s of the $s + r$ encoded fragments. In a P2P storage system, these fragments are then placed on $s + r$ different peers of the network. In this work, we focus on the analysis of systems that uses Erasure Codes as they are usually more efficient in terms of storage overhead (see [30]).

To keep a durable long-term storage despite disk failures, the system must be capable to maintain a minimum number of fragments available in the network. This means that the system continuously monitors the number of fragments of each data-block. This control is done in a distributed way by the means of a Distributed Hash Table (DHT) [20]. If this number of fragments drops below a certain level, the block is reconstructed. After the reconstruction, the regenerated missing pieces are spread among different nodes. A fundamental question for such systems is how much resource (bandwidth and storage space) is necessary to maintain redundancy and to ensure a given level of reliability?

It has been shown that fragment (or replica) placement has a strong impact on the system performance [9, 19]. In this paper, we study three different strategies of data placement. The first, a *global & random* placement policy, spreads the fragments on peers taken uniformly at random among all the peers of the system (see [1, 28, 5]). The other two, namely *Chain* policy and *Buddy* policy, distribute the fragments among a closed set of neighbor peers, and henceforth are designated as *local*. The Chain policy corresponds to what is done in most distributed systems implementing a DHT (see [27, 6, 10]). The Buddy policy roughly corresponds to a collection of RAID systems.

The use of the Global strategy allows to distribute more uniformly the load among peers, leading to a faster reconstruction and a smoother operation of the system [5]. However, the use of *local* strategies brings practical advantages [7]. For instance, the DHT update mechanisms of the leafset can be used to simplify the management of the system (e.g. to know the states of the blocks stored locally). Also, the management traffic and the amount of meta-information to be stored by the nodes are kept low, meaning in $O(s + r)$ when there are in $O(N)$ for the Global placement, with N the number of peers in the system. Furthermore, since routing is done directly among known nodes of the leafset, there is no massive use of the slow routing algorithms. Hence, the induced delay of the control traffic is kept low and this gives a better latency to access the data.

We compare the three policies for two different scenarios. In the first one, *provisioning scenario*, peers do not have bandwidth constraints. It allows to estimate the bandwidth use for different sets of parameters. The second scenario, where peers have resource constraints, corresponds to the operation of practical systems. We focus our analysis on three main metrics:

- *Bandwidth*: Average bandwidth consumption per peer, i.e., estimated from the number of fragments transmitted and received per hour due to the reconstruction process;
- *FDLPY*: Fraction of Data Loss Per Year, which gives the probability to lose a data-block per year;
- *MTTDL*: Mean Time To Data Loss, i.e., the period of time between two occurrences of data loss in the system.

Our contributions are the following:

- As far as we know, we present the first practical study of data placement for systems using Erasure Codes. We show that, even for *local* policies, they experience less data loss than replication schemes with the same resources.
- We show that, *without bandwidth constraints*, the distribution of the bandwidth usage among peers is much more smoother for the Global policy, moreover, all policies have the same average bandwidth consumption and probability to lose data. However, the mean time between data loss events is much longer for the *local* policies.
- When *limiting the maximum available bandwidth* per peer, we exhibit that the Global policy experiences a lot less data loss than the *local* policies for similar available resource. In addition, the loss events for *local* policies are much more frequent when compared to the provisioning scenario (in certain cases even more frequent than for the Global).
- We then discuss the size of the leafset in the *local* policies. We show that these policies can be adapted to achieve performances close to the *Global* placement, while keeping the practical advantages of locality.
- We additionally propose a new reconstruction scheme, namely *external reconstruction*, which reduces by 40 to 50 percent the number of block losses when using the *local* policies.
- Finally, we provide analytical methods to estimate the mean time to the occurrence of data loss for the three policies.

Related Work.

The majority of existing or proposed systems, e.g. Intermemory [14], Farsite [4], CFS [6], PAST [11], Glacier [15], use a local placement policy. For example, in PAST, the authors use the Pastry DHT to store replicas of data into logical neighbors. In the opposite way, some systems use a Global policy, as OceanStore [18], pStore [2], TotalRecall [3] or GFS [12]. GFS spreads chunks of data on any server of the system using a pseudo-random placement.

Chun et al. in [5] also discuss the impacts of local placement (namely *small scope*). They state that local placement is easy to maintain but induces higher reconstruction times. Conversely, larger scope (Global policy) has lower reconstruction time and henceforth higher durability. However, they do not address the impact of different bandwidth limits, neither Erasure Codes redundancy.

Ktari et al., in [17], discuss the impact of data placement. They do a practical study of a large number of placement policies for a system with high churn. They exhibit differences of performance in terms of delay, control overhead, success rate, and overlay route length.

Lian et al., in n [19] study the impact of data placement on the Mean Time to Data Loss (MTTDL) metric, but for a system based of simple replication. They show that the MTTDL is lower for the Global policy (called random placement) when compared to the local policy (called sequence). But they do not discuss other very important metrics: the probability to lose a block and the bandwidth usage.

There are also other studies that evaluate the replica placement, however with focus on the lookup latency and/or throughput performance [29]. Others are focused on Content Delivery Networks [16], which is not our case here. We aim at analyzing P2P storage systems to achieve high durability and availability at low cost in bandwidth usage.

Organization

The remainder of the paper is organized as follows: in the next section, we detail the characteristics of the distributed systems we analyse, explain the different placement policies, and present our simulation tools. In Section 3.1 we study the behavior of the system without resource constraints, and then under bandwidth constraints in Section 3.2. Finally, we propose some improvements of the placement and reconstruction architectures in Section 4, followed by analytical methods to estimate the MTTDL metric for the three placements in Section 5.

2 Description

2.1 The System

The detailed characteristics of the studied distributed storage system are presented in this section.

Data Redundancy. Erasure Codes schemes [21] are used to introduce data redundancy in the system. The user data is cut into data-blocks. Each data-block is divided into s initial pieces, then r pieces of redundancy are added, in such a way that the initial block can be reconstructed from any subset of s pieces among the $s+r$. The pieces are then sent to $s+r$ different peers according to one of the three data placement policies of study.

Failures. It is assumed that the nodes stay connected almost all the time into the system. So, we model the case of *peer failures*, mainly caused by a disk crash or by a peer that definitively leaves the system. In both cases, it is assumed that all the data on the peer's disk are lost. Following most works on P2P storage systems [1, 19, 24, 23], peers get faulty independently according to a memoryless Poisson process. Given a peer failure rate λ, the probability for a peer to be alive after a time T is given by $e^{-\lambda T}$. To avoid the problem of transient failures and deal with churn, a peer is just considered lost if it has left the system for a period longer than a given timeout [25] (set to $\theta = 12$ hours in our simulations). As failures happen continuously in a large system, it is essential to the system to *monitor* the data-blocks' state and maintain the redundancy by *rebuilding* the lost fragments.

Reconstruction Strategy. Different reconstruction strategies can be considered. Delaying the reconstruction (i.e. waiting for the block to lose more than one fragment before rebuilding it) amortizes the costs over several failures. Hence, we study a *saddle based policy* in this paper. When the number of fragments of a block drops to a threshold value r_0, the reconstruction starts. Note that, when r_0 is set to $r-1$, the reconstruction starts as soon as a first piece is lost. This special case is called *eager policy.* Setting a low value for r_0 decreases the number of reconstructions (as the reconstruction starts only after that $r-r_0$ pieces are lost), but increases the probability to lose a block.

The reconstruction is done in three consecutive phases: the *retrieval*, the *recoding* and the *sending.* First, the peer in charge of the reconstruction has to download s fragments among the remaining block's fragments (retrieval). It then recodes the block (decoding). Last, it sends the reconstructed fragments to peers (sending). We consider here that the CPU recoding time is negligible. Therefore the *reconstruction time* is the sum of the retrieval and sending phases.

Control. Some sort of Distributed Hash Table substrate is assumed to be implemented, so the management of the system is distributed. In this

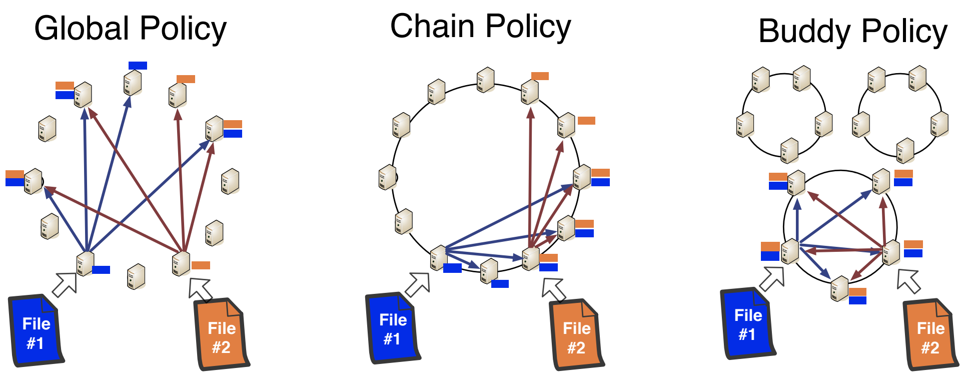

Figure 1: Placement of two blocks b_1 and b_2 in the system. Global: $s+r$ fragments are placed at random among all peers; Chain: fragments are placed on $s+r$ neighboring peers; Buddy: many small subsystems of size $s+r=5$, in this case all peers inside each small group contain the same data.

paradigm, the control traffic per peer has order of $\Theta(\log N)$. As an example, to monitor a peer, it is sufficient to have $\log N$ other peers in charge of it, e.g. pinging it periodically: if it stops to answer, this information is forwarded to all the peers, which could be done using the standard error tolerant broadcasting. The factor $\log N$ is a classical redundancy factor to handle failures in DHT, that allows to maintain the redundancy level of the data-blocks and peers' state. Since the traffic induced by the fragments transfers of the reconstructions is much bigger than the control traffic, this later can be considered negligible here.

2.2 Data Placement Policies

In distributed storage systems using *erasure codes*, each block of data is divided into s fragments. Then, r fragments of redundancy are added in such a way that any subset of s fragments from the $s+r$ fragments are enough to regenerate the data block. The $s+r$ fragments are then sent to different peers of the network. The different placement policies studied in this chapter are detailed in the following and depicted in Figure 1.

Global Policy

In the Global policy, the $s+r$ fragments of a block are sent to $s+r$ peers chosen uniformly at random *among all the N peers* present in the system. In this case, the peer in charge of monitoring the state of a block and re-constructing it is also selected among all peers in the system.

Chain Policy

In the Chain, which is a *local* policy, the network is seen as a directed ring of N peers. The fragments are then sent to $s + r$ consecutive peers chosen uniformly at random among the N possible sequences of $s + r$ peers. This policy corresponds to what is done in most distributed systems implementing a DHT. That is, the peer responsible for a data block stores the blocks' fragments on its closest neighbors. Note that these neighbors are also in charge of monitoring and reconstructing these blocks.

Buddy (or RAID) Policy

The Buddy is an extreme case of a *local* policy, in which the peers are divided into C independent clusters of size exactly $s + r$ each. The fragments are then sent to a cluster chosen uniformly at random among the clusters. In this situation, all peers of a cluster store fragments of the same set of blocks. It could be seen as a collection of local RAID like storage.

2.3 Simulations

To evaluate such a system, we developed a *custom cycle-based simulator* that implements all the characteristics described in Section 2.1. The simulator models a detailed view of the system, as it monitors the state and the localisation of each fragment individually.

2.3.1 Bandwidth Queue Model

We model the bandwidth as a resource constraint per peer, and not as a global shared link constraint as is done in [19, 24]. Each peer has a maximum upload and download bandwidth, resp. BW_{up} and BW_{down}. We assume asymmetric capacities, as often encountered in practice, e.g. ADSL lines (in our experiments $BW_{down} = 5BW_{up}$). So the limiting resource is the upload bandwidth and it is the one presented in our results.

To model bandwidth, we implemented a *non blocking FIFO queue with one server*. When there is a peer failure, the blocks to be reconstructed are put in the queue. The simulator processes these blocks at each cycle (*FIFO order*). For each block it tries to retrieve the fragments, then resends the redundancy fragments to different peers (if the corresponding peers has still some available bandwidth). If some fragments are not available (i.e., the peers storing it already reached the maximum upload capacity), then it tries to download the fragments of the next block in the queue (*non blocking policy*).

Table 1: Summary of main notations and their default values in our experiments. See Remarks 1 and 2 for the choice of the parameter values.

N	Number of peers	$1,005$
B	total of blocks in the system	10^5
s	Number of fragments in the initial block	9
r	Number of redundancy fragments	6
r_0	reconstruction threshold value	2
τ	time step of the model	1 hour
$MTBF$	Peer mean time between failures	90 days
BW_{up}	Upload bandwidth per peer	6-18 kbit/s and Unlim.

2.3.2 Simulation parameters

We did a large number of simulations for different sets of the parameters. The default parameter values used in the simulations are given in Table 1, otherwise they are explicitly indicated. The considered size of fragment is 400 KB. With $s = 9$ and $r = 6$, the original data block size 3.6 MB and the total size with redundancy is 6 MB. Two remarks on the choice of the parameter values:

Remark 1 (Size of the simulated system): In practice, peers have huge disks of tens of Gigabytes, each one containing tens of thousands of blocks. Furthermore a real system with 1000 peers would deal with tens of millions of fragments. As we want to be able to simulate a storage system for several years in a reasonable time (for instance, our simulations correspond to 200 simulated years and the time granularity is 1 hour), we choose a disk size around 100 times smaller than the one expected in practice. Each peer stores only 1500 fragments in the system, which still corresponds to a total of 1.5 millions of fragments and 571GB of data. Note that the upload bandwidth (BW_{up} spans from 6 to 18 kbit/s in our experiments) directly derives from this choice: disks containing 100 times more data would need a peer bandwidth 100 times larger to maintain the redundancy, that is already in order of Mbits/s and close to the bandwidth limits encountered in practice.

Remark 2 (Measuring block losses): The parameters of real systems are set in such a way that the occurrence of a data loss is a very rare event. As it is impossible to simulate in a reasonable time events of very low probability, for example 10^{-15}, we choose *non realistic values* for some parameters (in particular, the reconstruction saddle $r_0 = 2$ and the disk $MTBF = 90$ days are set very low). In this way, we experience data loss in our simulations. Of course, real systems would have a completely different probability to lose blocks than the one reported here for the sake of comparison.

3 Comparison of Three Placement Policies via Simulation

In this section, we evaluate the three data placement policies for the three following metrics: use of bandwidth, number of dead blocks, and mean time to data loss. First, we study the provisioning scenario (*unlimited bandwidth*) in Section 3.1, which is important to measure the required bandwidth to maintain the system. Afterwards, we analyze scenarios with *constrained resources*, in Section 3.2.

3.1 Without Resource Constraints

Briefly, the results shown here are: (1) the three placement strategies have the same value of average bandwidth demand; (2) however *local* policies exhibit strong variations in resource usage across peers; (3) they have the same probability to lose a data-block per year, (4) but the MTTDLs of the Buddy and the Chain policies are longer.

A summary of the simulation results are presented in Table 2. We then discuss: first, the bandwidth consumption; then, the data loss rate; and finally, the mean time do data loss.

Table 2: Summary of results (without bandwidth constraints).

Policy	Bandwidth (kbit/s)	FDLPY (blocks)	MTTDL (years)
Global	1.99 (± 1.34)	$4.1 \cdot 10^{-4}$ ($\pm\ 0.6 \cdot 10^{-4}$)	0.02 (± 0.02)
Chain	1.99 (± 12.83)	$4.1 \cdot 10^{-4}$ ($\pm\ 8.6 \cdot 10^{-4}$)	4.0 (± 3.0)
Buddy	1.99 (± 15.92)	$4.4 \cdot 10^{-4}$ ($\pm\ 25.4 \cdot 10^{-4}$)	25.8 (± 21.7)

3.1.1 Average Bandwidth Usage

The left column of Table 2 shows the average value of upload bandwidth usage across peers during time (i.e., at each time step we measure the average number of fragments transmitted by each peer), along with the experimental standard deviation (in parenthesis).

First, as expected, the average bandwidth use across peers is roughly the same for all policies, 1.99 kbit/s. The reason is that the different placement policies do not change the number of fragments that have to be reconstructed.

However, the variations are not the same, because the policies change the repartition of these pieces among peers. The Chain policy and Buddy policy variations are significantly higher, respectively, 9.5 and 11.8 times more than the Global policy. Figure 2 gives an explanation of this behavior. It depicts the bandwidth usage of the 1005 users of the system at a typical instant of time for the three different policies under the same failure scenario. We see that the load is around 2 kbit/s for all the users and all strategies. However, we see that the distributions of the bandwidth are not the same at all.

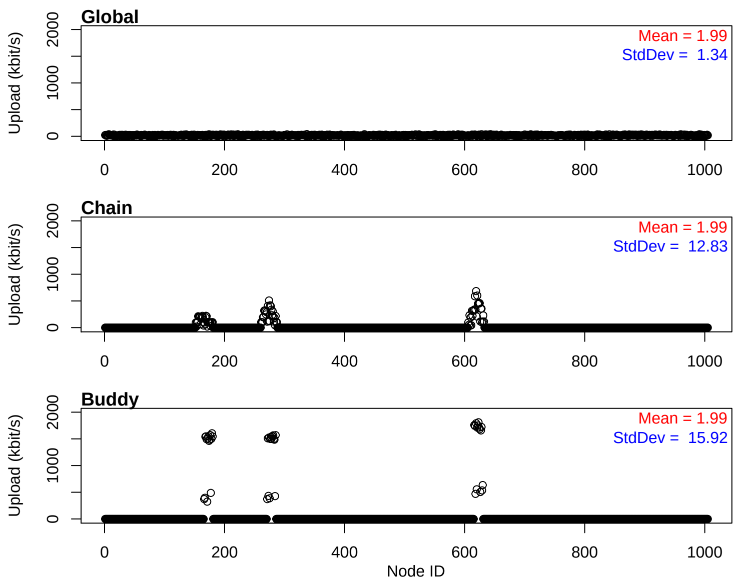

Figure 2: Variations of bandwidth usage across users for the three different strategies for a typical time step and under the same failure scenario.

In the case of the Global policy (top graph), the fragments of the blocks are placed among all the peers in the system. Consequently the load of the retrieval phase of the reconstruction is uniformly distributed among all peers. Furthermore, the peers in charge of a block reconstruction are also randomly chosen among all peers. So the sending phase of the reconstruction is also evenly distributed. In the example, the standard deviation of the bandwidth of the Global is 1.34 kbit/s.

On the opposite (bottom graph), in the Buddy setting, some groups of $s + r$ users have a very high bandwidth demand, e.g. around 1500 kbit/s. We can identify three groups that correspond to three different sets of peer crashes that triggered reconstructions. In the Buddy policy (similarly to RAID systems), when a failure happens, only the immediate neighbors possess the remaining fragments of the blocks. Moreover, these neighbors are also in charge of the reconstructions, leading to a very high bandwidth load on these peers, while the others peers in the system are spared[1].

[1]There are two groups of peers in each spike of the Buddy. A bigger one around 1500 kbit/s, that corresponds to peers doing the retrieval and sending phases of the reconstruction (i.e., $s + r - r_0$ uploads for each block). The smaller one, with an upload bandwidth around 400 kbit/s, correspond to peers that have failed and were replaced with empty disks. As they are empty, they do not send fragments to the reconstructors (no retrieval upload), but they are in charge of some reconstructions, so we see their sending upload

The situation for the Chain policy (middle graph) is similar to the Buddy. We also observe three spikes for certain subsets of users. But, differently, these spikes (1) involve more peers and (2) have shapes of pyramids. It is explained as follows. (1) A peer stores fragments of blocks that are managed by peers at distance $s + r$ (chain size). In addition, a block reconstruction affects peers at distance $s+r$. Hence, when a peer crashes, peers at distance $2(s + r) - 1$ contribute to the reconstruction. (2) The spikes correspond to multiple disk failures. In this scenario, peers close to several failed peers contribute more than peers close to a single failed peer. Hence, the pyramid shaped spikes. To conclude, we see that a peer failure is a quite *big local event* for the two local policies.

3.1.2 Probability to Lose a Block

We then compare the probability to lose a block in the three different policies. The results are shown in the middle column of the Table 2, normalized as the Fraction of Data Loss Per Year (FDLPY).

When there is no bandwidth limit, the *expected number of dead blocks is the same for the three policies* (roughly 0.04% of blocks lost per year). As a matter of fact, the probability for a block to die does not depend on where its fragments are placed. It can be easily calculated using a Markov Chain Model, see for example [1, 13]. But, we note that the *deviations* during time of the number of dead blocks is higher for *local* policies. This is further explained by looking at the MTTDL metric.

3.1.3 Mean Time To Data Loss

The measure of the time between two occurrences of data loss shows that the three policies have very distinct behaviors, as depicted in the right column of Table 2. In our simulations, the Global policy loses a data-block every 9 days, the Chain policy every 4 years and Buddy every 25.8 years. However, as we have seen before, the three policies have in average the same number of dead blocks per year. In other words, the average quantity of data loss per year is the same, but the distribution across time of these losses is very different.

The Figure 3 illustrates an example of the cumulative number of dead blocks for a period of 3 years for the three placement policies under the same failure scenario. We see that the loss occurs regularly for the Global policy. Conversely, they occur very rarely for the Buddy placement, but, when they occur, they affect a large batch of data. Basically, all the blocks of a small buddy subsystem of size $s + r$ peers lose all their blocks at the same time. The behavior of the Chain policy is somewhere in the middle of both.

(i.e. $r - r_0$ fragments for each block).

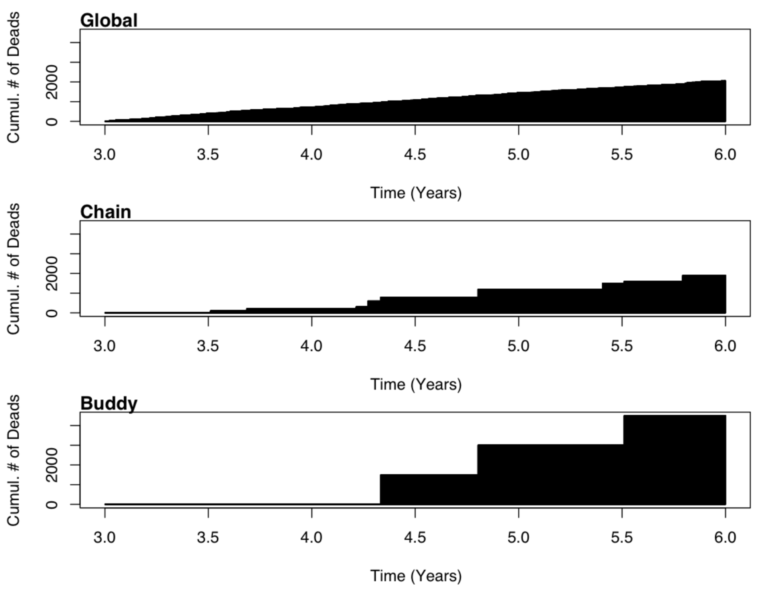

Figure 3: Illustrative example of the cumulative number of dead blocks for a period of three years.

It has also to be noticed that, due to its very large variations of behavior, the buddy policy has the drawback of being not very predictable. We see in Figure 3 that the Global and the Chain policies experienced around 2,000 block losses after 6 years, when the Buddy policy experienced almost 4,000. Even if they have the same probability to lose data. In a long-run, the number of expected losses are the same.

3.2 Results under Resource Constraints

In this section, we study the behavior of the system with bandwidth limitation per peer (meaning that now each peer has a maximum upload and download bandwidth). In this context we show that, using similar available resources, the amount of data loss is no more the same for the three data placement policies. The Global policy behaves considerably better in comparison to the Chain and Buddy policy. Furthermore, the *local* policies now experience more loss events (smaller MTTDL).

3.2.1 Reconstruction Time versus Bandwidth

Figure 4 gives the average reconstruction time for different upload bandwidth limits BW_{up}, ranging from 6 kbit/s to 18 kbit/s. The unlimited bandwidth value is given for the sake of comparison.

We see that the average reconstruction time is a lot longer for the Chain policy and even more for the Buddy policy compared to the Global one. As

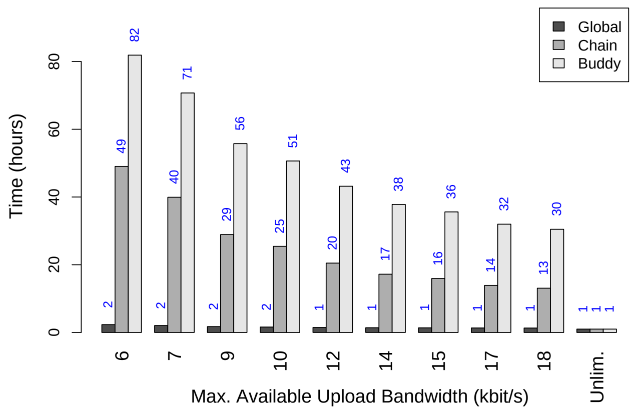

Figure 4: *Average reconstruction time* for different bandwidth limits for the three strategies. The values for the local policies (Chain and Buddy) are higher, and decrease gradually with the increase of the available bandwidth. The unlimited value of 1 is the granularity of our simulator (one hour).

an example, for a maintenance bandwidth of 6 kbit/s, the reconstruction time is around 49 hours for the Chain policy and 82 hours for the Buddy, but only 2 hours for the Global policy. This bandwidth limit corresponds to three times the average bandwidth usage of the system (as computed without resource constraints). Hence, we see that the imbalance of the reconstruction load among peers has a very strong impact on the reconstruction time, even if each policy has the same average bandwidth demand. For a bandwidth limit of 18 kbit/s, which represents *9 times the average bandwidth* needed, the difference is still very large: 1, 14, and 30 times for the Global, Chain and Buddy, respectively. Thus, under resource constraints, the big local events constituted by peer failures induce longer reconstruction time and henceforth an increase of data loss when using the *local* policies, as shown in the following.

3.2.2 FDLPY versus Bandwidth

A critical performance measure of a P2P storage architecture is the probability to lose a block for a given amount of bandwidth. Figure 5 compares the trade-offs of the three policies for different values of BW_{up}. We see that the Global policy behaves a lot better for any bandwidth limit than the Chain policy, which itself is more efficient than the Buddy policy. For example, for a bandwidth limit of 18 kbit/s (which represents 9 times the average bandwidth need of the system), the Global experiences 0.04% of data loss per year, to compare with 0.78% and 3.2% for the Chain and the

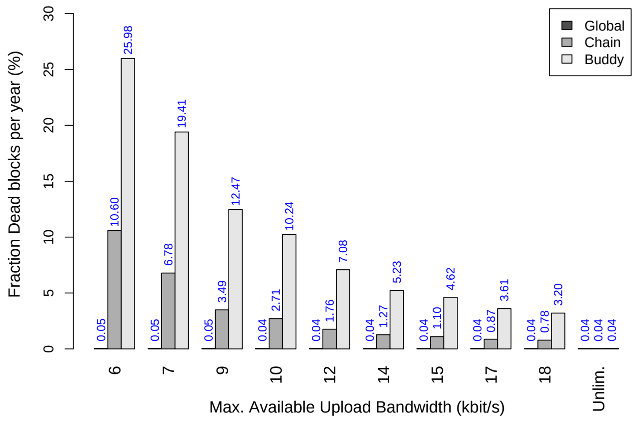

Figure 5: *Fraction of block losses per year* (see Remark 2) for different bandwidth limits for the three strategies. The differences between policies are stronger than for the reconstruction times of Fig 4, as explained in Section 4.2.

Buddy, respectively.

3.2.3 MTTDL versus Bandwidth

Opposed to what was showed without bandwidth constraints, the Global policy behaves better than the others with low bandwidth limitations. The following tables shows the MTTDL and the FDLPY for the three policies under resource constraints.

MTTDL (hours)

Max.BW (kbit/s) =	6	9	12	15
Global	166	180	193	219
Chain	53	160	323	565
Buddy	75	178	341	582

FDLPY (%)

Max.BW (kbit/s) =	6	9	12	15
Global	0.05	0.05	0.04	0.04
Chain	10.6	3.5	1.8	1.1
Buddy	26.0	12.5	7.1	4.6

For instance, without resource constraints, the time between data loss were 0.02, 4.0, and 25.8 years respectively for the Global, Chain and Buddy. Conversely, with an available bandwidth of 6 kbit/s, these values are 166, 53, and 75 (in hours), many orders of magnitude less. These results show

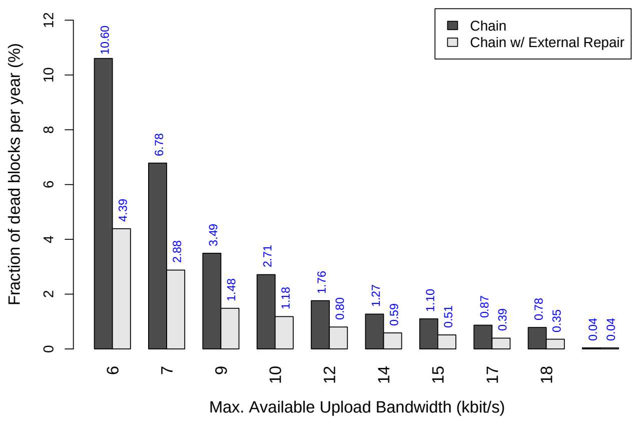

Figure 6: Comparison between the Chain policy with internal reconstruction and with external reconstruction. The fraction of block losses (see Remark 2) for different bandwidth limits is presented. There is an improvement of about 50% on the fraction of data losses.

that the impact of the bandwidth limits per peer needs to be taken into account when analyzing such systems.

4 Proposition for P2P Storage System Architectures

In this section, we suggest some modifications of the architecture of systems implementing *local* policies. We also discuss the best choice of their parameters. First, we propose an *external reconstruction strategy* for the *local* policies, and show that it can lower the duration of the sending phase of reconstructions, and thus reduce the probability to lose data. Second, we show that having a larger neighborhood is sufficient to greatly improve the Chain policy performance. Hence, an architecture with the advantage of locality and performance close to the ones of a Global strategy can be obtained. Finally, we carry out some comparisons between Replication and Erasure Code schemes. We show by simulations that, for the same amount of bandwidth and space overhead, the Erasure Codes are better even for the Chain policy.

4.1 External Reconstruction Strategy

We propose here a new reconstruction architecture for the Chain policy, namely *external reconstruction*. The idea is to use peers outside the Chain group to carry out the reconstruction process. In this way, the bandwidth usage is more uniformly spread among peers. More precisely, only the upload bandwidth of the retrieval phase of the reconstruction is needed locally, while the bandwidth for the sending phase is provided by all the peers of the system. Hence, the *External Reconstruction* has two main advantages:

- a local control for discovering failed peers and updating the data-blocks' states;
- a more uniform distribution of resources among peers, which lowers the reconstruction time.

However, a small cost is paid: in the internal reconstruction, the peer in charge may be chosen in such a way that it possesses a piece of the block to be reconstructed. It reduces by a factor of $(s-1)/s$ the bandwidth needed for the retrieval phase of the reconstruction. Conversely, in the external reconstruction, the reconstructor does not contain any piece.

A rough estimate of the gain in terms of reconstruction time can be given. In the internal reconstruction, the *local peers* have to upload $(s-1)+(r-r_0)$ fragments, noted as $up_{internal}$. However, when using the external reconstruction they only have to upload s fragments, noted as $up_{external}$. As the local peers are the bottleneck of the reconstruction, the gains in terms of bandwidth and hence of reconstruction time are roughly $1-s/(s-1+r-r_0)$ (it comes from the ratio $1-up_{external}/up_{internal}$). With the parameters chosen in our experiments, this factor would be 0.25. Note that the gains in terms of data loss will be significantly higher (see Section 4.2).

Figure 6 compares the internal and external policies. It gives the trade-off between the average number of dead blocks per year and the available bandwidth. For the same bandwidth, the fraction of data loss decreases by a factor between 0.5 and 0.6 for this set of parameters. We see that, by choosing to carry out the reconstructions externally, the chain policy behaves substantially better.

4.2 What Should Be the Size of the Neighborhood?

We showed above that the Global policy in practice (under tight resource constraints) behaves significantly better than the local policies. Nevertheless, as already stated in the introduction, there exist important practical considerations that explain the choice of *local* placement. Would it be possible to obtain the same practical advantages as the local policies (a small sub-network to monitor) but without paying the high cost of the Chain and Buddy in terms of probability of data loss?

In this section, we study the impact of the *size of the block neighborhood* on the system performance. The block neighborhood is defined as the peers that can receive fragments of this block (of size $s+r$ for Buddy and Chain, and of size N for Global).

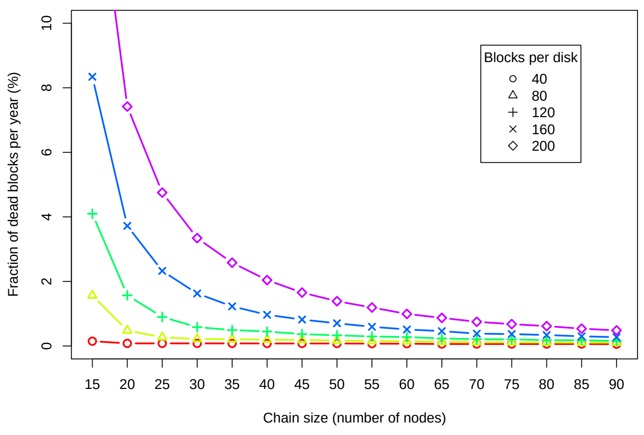

Figure 7: Study of the size of the block neighborhood. Fraction of block losses (see Remark 2) per year for different sizes of neighborhood and different number of fragments per disks.

Figure 7 shows the average number of dead blocks per year for different sizes of the neighborhood. The sizes range from $s + r = 15$ (corresponding to the size of the neighborhood for the Chain policy) to 90. The experiment was done for different amounts of data per disk (i.e., number of blocks per disk), from 40 to 200, which is, as we will see, an important parameter when choosing the neighborhood size. We see that barely increasing the neighborhood from 15 to 20 has a striking impact on the data loss: with 120 blocks per disk, the fraction of data loss drops from 4.1% to 1.6%, representing a decrease of almost 61 percent. Thus, increasing the size even by few units leads to strong improvements of the system performance. However, the number of dead blocks decreases from 0.17% to 0.16% for neighborhoods of sizes 85 and 90 nodes. The marginal improvement strongly decreases.

The shapes of the curves may be explained as follows. When there is a peer failure, its neighbors are in charge of reconstructing the lost fragments. Hence the reconstruction time (minus the discovery time) depends almost linearly on the neighborhood size. This dependence can be directly translated to the probability to lose data:

Exponential relation between the probability to die and the reconstruction time. During a reconstruction, a block dies if it loses $r_0 + 1$ fragments before it finishes. The probability for a peer to be alive after a time T is $\exp(-\lambda T)$, where λ is the peer failure rate. Hence a good approximation of the probability to die during a reconstruction lasting a time T is

Table 3: Comparison of Replication and Erasure Codes when using the Chain placement. Number of dead blocks per year and average bandwidth usage for different values of redundancy k.

Estimated Fraction of Data Loss per Year (%)

$k =$	1	2	3	4
Repl.	$2.2 \cdot 10^1$	$3.0 \cdot 10^{-1}$	$3.8 \cdot 10^{-4}$	$1.9 \cdot 10^{-4}$
Erasure	$6.9 \cdot 10^1$	$2.5 \cdot 10^{-6}$	$3.2 \cdot 10^{-17}$	$4.3 \cdot 10^{-29}$

Upload Bandwidth usage (kbit/s)

$k =$	1	2	3	4
Repl.	1.08	3.24	4.34	5.40
Erasure	1.45	2.47	3.42	4.37

In the case of Erasure Codes $s = 4$ and $r = sk$

given by

$$\Pr[die|Rtime = T] = \binom{s + r_0}{r_0 + 1}(1 - e^{-\lambda T})^{r_0+1}(e^{-\lambda T})^{s-1}.$$

Hence we have an exponential relationship between the number of block losses and the neighborhood size.

The neighborhood size should mainly be chosen in function of two parameters: the disk size (or the number of fragments per disk) and the peer bandwidth. Note that a size of $\frac{D}{(r-r_0)BW_{up}}$ allows to reconstruct the blocks in one time step and is sufficient to get the benefits of Global (with D the number of fragments per disk, BW_{up} expressed in blocks/time step and $1/(r - r_0)$ the fraction of blocks of the lost disk that go beyond the saddle value).

Concluding, to implement a local policy, the neighborhood should at least be a little bit larger than $s + r$, as the marginal utility of increasing the block neighborhood is tremendous for very small sizes. In addition, the neighborhood size should be chosen in function of the disk size: The larger the number of fragments per disk, the larger the block neighborhood should be.

4.3 Replication versus Erasure Codes

Other experimental studies on data placement analyze the case of replication instead of Erasure Codes, see e.g. [28, 10, 6, 19]. It is shown in [30] that Erasure Codes could be used to achieve a high availability of data storage with low space overhead, but these studies assume a Global and random placement strategy. We show here that, *even for local policies*, the Erasure Codes scheme is more efficient, meaning that it has less probability to lose data for the same storage and bandwidth usage. Hence, we confirm the pertinence to carry out an analysis of data placement when using Erasure Codes.

Note first that replication is a special case of Erasure Codes, but with only one initial fragment ($s = 1$). Hence, we used exactly the same simulator to carry out the experiments. We evaluated the system for different number of replicas k, with values varying from 1 to 4. To have the same storage overhead factor, we compare the scenarios using k replicas with a system with $r = ks$ erasure coded fragments. Then we choose the reconstruction saddle value equals to $r_0 = r - s$ ($k_0 = k - 1$ to the case of replication), so that we experience the same number of reconstructions, that is, roughly the same bandwidth usage. We present in Table 3 the average bandwidth use and fraction of data loss per year for both techniques. In the case of Erasure Codes, the number of initial fragments is $s = 4$. For instance, for a value of replication $k = 3$ (this means $r = 12$ to the case of Erasure Codes), the reconstruction starts when $k = k_0 = 2$ (and when $r = r_0 = 8$ for the Erasure Codes).

We see that, for $k = 1$, the system with replicas behaves better than Erasure Codes, while using less bandwidth. But, as soon as $k \geq 2$, systems with Erasure Codes behave strikingly better: they experience a lot fewer block losses while using a little less bandwidth. In practice, to have a very low probability to lose data, real systems use values of k larger than 4, see e.g. [28]. Thus, systems with Erasure Codes have less probability to lose data for the same amount of resources and realistic levels of redundancy.

5 Analytical Estimations of MTTDL

In the last sections it is shown that fragment placement has a strong impact on the system performance. In the following we describe analytical methods to compute exact values and approximations of the MTTDL for the three policies.

Here we study the case where the reconstruction starts as soon as one of its fragments is lost, namely *eager* reconstruction strategy. In addition, the blocks are reconstructed in one time step, i.e., there is enough bandwidth to process the reconstruction quickly. After the reconstruction, the regenerated missing fragments are spread among different peers. Hence, after each time step, the system is fully reconstructed.

Data Loss Rate. As already discussed, a data loss occurs when at least one block is lost. A block is considered lost if it loses at least $r + 1$ fragments during one time step, otherwise, recall that all the $s + r$ fragments are fully reconstructed at next time step. The data loss rate for a given block comes straightforward. This loss rate does not depend on the placement policy (as soon as it is assured that all fragments are stored on different peers). Hence, we have the same expected number of lost blocks for the three placement policies. However, as stated in Section 3.1, the measure of the time to the first occurrence of data loss shows that the three policies have very distinct behaviors. It is shown by simulations that the average quantity of data loss per year is the same, but the distribution across time of these losses is very different (see Figure 3).

In the next three sections (Section 5.1, 5.2 and 5.4), we present methods to compute exact values and approximations of the MTTDL for the three placement policies. For each policy, we calculate the probabilty $\mathbb{P}_{policy}$ to lose data at any given time step. Then, we deduce $MTTDL_{policy} = 1/\mathbb{P}_{policy}$. We start by presenting the calculations for the Buddy policy as it is the simple one, then we present the Global and Chain in following.

5.1 Buddy Placement Policy

In the Buddy placement policy, the N peers are divided into C clusters of size $s+r$ each. In this strategy, the calculation of the MTTDL$_{buddy}$ is straightforward. Given a cluster, the probability to have a block loss is the probability that the cluster loses at least $r+1$ peers (i.e., fragments), is given by

$$\mathbb{P}_{cluster} = \sum_{j=r+1}^{s+r} \binom{s+r}{j} \alpha^j (1-\alpha)^{s+r-j}. \tag{1}$$

In fact, when that happens all the data stored on that cluster is lost. Remember that α is the probability of a given peer to fail at one time step. Since all the C clusters are independent, the probability to have a data loss is given by $\mathbb{P}_{buddy} = 1 - (1 - \mathbb{P}_{cluster})^C$.

If the average number of cluster failures per time step $C{\cdot}\mathbb{P}_{cluster} \ll 1$, as expected in a real system (i.e., the probability of simultaneous cluster failures is small), then we have $\mathbb{P}_{buddy} \approx C{\cdot}\mathbb{P}_{cluster}$, and so MTTDL$_{buddy} \approx 1/(C \cdot \mathbb{P}_{cluster})$.

If $(s+r)\alpha \ll 1$, we can approximate even more. In other words, this assumption means that the probability of a peer failure α is small. Since the ratio between two consecutive terms in sum of Equation (1) is $\leq (s+r)\alpha$, we can bound its tail by a geometric series and see that it is of $O((s+r)\alpha)$. We obtain $\mathbb{P}_{cluster} \approx \binom{s+r}{r+1}\alpha^{r+1}$. Then we have

$$\boxed{\text{MTTDL}_{buddy} \approx \frac{1}{\frac{N}{s+r} \cdot \binom{s+r}{r+1}\alpha^{r+1}}.} \tag{2}$$

5.2 Global Placement Policy

In the Global policy, block's fragments are parted between $s+r$ peers chosen uniformly at random. First, we present the exact calculation of the

MTTDL$_{global}$. We then present approximated formulas that give an intuition of the system behavior.

First, we consider i failures happening during one time step. Let F denote the set of the placement classes (i.e., groups of $s+r$ peers) that hold at least $r+1$ of these i failures; we have:

$$\#F = \sum_{j=r+1}^{i} \binom{i}{j}\binom{N-i}{s+r-j} \tag{3}$$

Then, suppose we insert a new block in the system: his $s+r$ fragments are dispatched randomly in one of the $\binom{N}{s+r}$ placement classes with uniform probability. Thus, the probability $\mathbb{P}_{block}(i)$ for the chosen class to be in F is:

$$\mathbb{P}_{block}(i) := \mathbf{P}\left[\text{placement in } F\right] = \frac{\sum_{j=r+1}^{i}\binom{i}{j}\binom{N-i}{s+r-j}}{\binom{N}{s+r}}$$

As block insertions are *independent*, if we consider our B blocks one after the other, the probability that none of them falls in F is $(1-\mathbb{P}_{block}(i))^B$. We then come back to the global probability to lose data considering different failure scenarii:

$$\begin{aligned}\mathbb{P}_{global} := \mathbf{P}\left[\text{lose data}\right] &= \mathbf{P}\left[\bigcup_{\{i \text{ failures}\}}[\text{failure kills a block}]\right] \\ &= \sum_{i=r+1}^{N}\binom{N}{i}\alpha^i(1-\alpha)^{N-i}\mathbf{P}\left[i \text{ failures kill a block}\right]\end{aligned}$$

Which gives us the MTTDL of the system using the global policy:

$$\text{MTTDL}_{global}^{-1} \approx \sum_{i=r+1}^{N}\binom{N}{i}\alpha^i(1-\alpha)^{N-i}\left[1-\left(1-\frac{\sum_{j=r+1}^{i}\binom{i}{j}\binom{N-i}{s+r-j}}{\binom{N}{s+r}}\right)^B\right] \tag{4}$$

5.3 Approximation

We provide here approximations for systems with low peer failure rates. One example is *Brick storage systems* [19]. Each peer is a "brick" dedicated to data storage, that is, a stripped down computer with the fewest possible components: CPU, motherboard, hard drive and network card. In these backup systems, as we want a very high data life time, we have either $\alpha N \ll 1$ or $\alpha N \sim 1$, i.e., we have a not too high mean number of peer failures per time step.

Computations of this complicated sum suggests that only its first terms matter, and especially the very first term when $\alpha N \ll 1$. We can formalize this: let us consider three "zones" for $i \in [\![r+1, N]\!]$: (I) $i \sim s+r$, (II) $s+r \ll i \ll N$ and (III) $i \sim N$. We introduce the following notations:

$$\begin{array}{rclcrcl} A_i & = & \sum_{j=r+1}^{s+r} \binom{i}{j}\binom{N-i}{s+r-j} & ; & C_i & = & 1 - \frac{A_i}{\binom{N}{s+r}} \\ \Gamma_i & = & 1 - C_i^B & ; & \Delta_i & = & \binom{N}{i}\alpha^i(1-\alpha)^{N-i}\Gamma_i \end{array}$$

Where A_i is nothing but $\#F$ in case i failures happen. In fact, and for the sake of curiosity, we can compute it easily with the following relation.

Lemma 1. *For $i \geq r+1$, $A_{i+1} = A_i + \binom{i}{r}\binom{N-(i+1)}{s-1}$.*

Proof. F is the set of placement classes with at least $r+1$ of them falling into a given "failure" set of size i. Let us see what happens when we increment the size of this failure set. We denote by S_i the initial failure set of F and $S_{i+1} = S_i \cup \{x\}$. A placement class falls in S_{i+1} iff it has at least $r+1$ peers in it, which is equivalent to either (a) having more that $r+1$ peers in S_i or (b) containing x and *exactly* r peers in S_i (cases where there are more than $r+1$ peers in S_{i+1}, including x, are already counted in (a)). From this we conclude that: $A_{i+1} = A_i + \binom{i}{r}\binom{N-(i+1)}{s-1}$. $\square$

The ratio between two consecutive terms of sum (4) is:

$$\rho := \frac{\Delta_{i+1}}{\Delta_i} = \frac{\alpha}{1-\alpha}\frac{N-i+1}{i+1}\frac{\Gamma_{i+1}}{\Gamma_i} \approx \alpha N \cdot \frac{\Gamma_{i+1}}{i\Gamma_i} \tag{5}$$

In zones (II) and (III), we can show this ratio is low enough so we can bound the tail of our sum by a geometric series of common ration $\rho \ll 1$.

Lemma 2. *In zone (I), under the assumption $\frac{N}{(s+r)^2} \gg 1$,*

$$\Delta_i \approx B\binom{s+r}{r+1}(\alpha N)^{i-(r+1)}\alpha^{r+1}(1-\alpha)^{N-i} \tag{6}$$

Proof. When $i \sim s+r$, we usually (read: in practice) have $A \Big/ \binom{N}{s+r} \ll 1$. Under our (strong) assumption, which is also verified in practice, we indeed have the simple bound $A \Big/ \binom{N}{s+r} \leq \left(\frac{(s+r)^2}{N}\right)^{r+1} \frac{s}{(r+1)!} \ll \frac{1}{B}$. Thus, Γ_i is almost proportional to C_i in zone (I), which implies $\Delta_i \approx B\alpha^i(1-\alpha)^{N-i}A\binom{N}{i} \Big/ \binom{N}{s+r}$. But simple combinatorics show that $A\binom{N}{i} = \sum_{j=r+1}^{s+r}\binom{s+r}{j}\binom{N-(s+r)}{i-j}\binom{N}{s+r}$, leading us to equation (6). $\square$

Lemma 3. *In zone (II), $\rho \approx \frac{\alpha N}{i}$.*

Proof. When $s + r \ll i \ll N$, we have

$$\begin{aligned}
A_i &\approx \sum_{j=r+1}^{s+r} \frac{i^j}{j!} \frac{(N-i)^{s+r-j}}{(s+r-j)!} \\
C_i &\approx \left(1 - \frac{i}{N}\right)^{s+r} \sum_{j=0}^{r} \binom{s+r}{j} \left(\frac{i}{N-i}\right)^j \\
&\approx \sum_{j=0}^{r} \binom{s+r}{j} \left(\frac{i}{N}\right)^j \sum_{l=0}^{s+r-j} (-1)^l \left(\frac{i}{N}\right)^l
\end{aligned}$$

Taylor expansion to second order in $\frac{i}{N}$ leads us to $\Gamma_i \approx B\,[2(s+r)-3]\left(\frac{i}{N}\right)^2$. Hence we see that $\frac{\Gamma_{i+1}}{\Gamma_i} \approx \left(1+\frac{1}{i}\right)^2 \approx 1$, equation (5) leading us to $\rho \approx \alpha N/i$. □

Lemma 4. *In zone (III), $\rho \leq \frac{\alpha N}{i}$.*

Proof. Let $\epsilon_i = 1 - \frac{i}{N}$: when $i \sim N$, we have $C_i \approx \sum_{j=0}^{r} \left(\frac{i}{N}\right)^j \epsilon_i^{s+r-j} \binom{s+r}{j} \approx \epsilon_i^s \binom{s+r}{r}$. Hence, $C_{i+1} - C_i \approx \frac{1}{N^s}\left(\epsilon_{i+1}^{s-1} + \cdots + \epsilon_i^{s-1}\right)\binom{s+r}{r} \leq \frac{1}{N^s} s \epsilon_i^{s-1} \binom{s+r}{r} \ll 1$. Then, Taylor expansion of the convex function $f(x) = 1 - x^B$ leads us to ($f'' < 0$):

$$\begin{aligned}
\Gamma_{i+1} - \Gamma_i &\leq (C_{i+1} - C_i) f'(C_i) \\
&\leq \frac{1}{N^s} s \epsilon_i^{s-1} \binom{s+r}{r} B C_i^{B-1} \\
\frac{\Gamma_{i+1}}{\Gamma_i} &\leq 1 + \frac{B \epsilon_i^{s-1} s \binom{s+r}{r}}{N^s} \frac{C_i^{B-1}}{1 - C_i^B}
\end{aligned}$$

Since in practice we have $B \ll N^s$, this upper bound is close to 1 and we conclude – as usual – with equation (5) giving $\rho \leq \alpha N/i$. □

Lemmas 3 and 4 tell us that, when $i \gg s + r$, our big sum is bounded by a geometric series of common ratio $\leq \frac{\alpha N}{i} \ll 1$, so only the terms before zones (II) and (III) numerically matter.

Lemma 2 can provide us with a stronger result. Equation (6) leads to $\rho \approx \alpha N$ in zone (I). Hence, if we also have $\alpha N \ll 1$, that is, mean number of failures per time step is really low (or, equivalently, time step is short enough), then only the first term of the sum matters.

If we simplify it further, we find:

$$\boxed{\mathrm{MTTDL}_{global} \approx \frac{1}{B\binom{s+r}{r+1}\alpha^{r+1}}} \qquad (7)$$

5.4 Chain Placement Policy

For the Chain policy, the computation of MTTDL$_{chain}$ is more difficult than the two previous ones, mainly because the chains are not independent of each other. From the definition of the Chain policy, a data loss occurs only when $r+1$ (or more) peer failures are located at $s+r$ consecutive peers.

We present in this chapter two approaches to compute or approximate the MTTDL for the Chain policy. We first describe computations using Markov chains techniques, and we then describe an analytical approximation value assuming that α is small enough.

5.4.1 Markov Chain Approach

The idea is to survey the N sequences $S_1, S_2, \ldots, S_N$ of $s+r$ consecutive peers. First, we define a binary-vector $(b_i, b_{i+1}, \ldots, b_{i+s+r-1})$ for each S_i, where the elements of this vector represent the state of peers of S_i: $b_j = 1$ if the peer numbered j is failed, $b_j = 0$ otherwise, $i \leq j < i+s+r$. Peer numbered $N+k$ is really the peer numbered k. Remark that the binary-vector of S_{i+1} is $(b_{i+1}, \ldots, b_{i+s+r})$.

As an example, consider a system composed of $N = 10$ peers with the values $s = 3$ and $r = 2$. The first sequence S_1 of peers is associated with the vector $(b_1, \ldots, b_5)$. If $\sum_{i=1}^{5} b_i \geq 3$, then it means that there is a data loss. Otherwise we have for example the vector $(0,0,1,0,0)$. Thus we now look at the vector $(b_2, \ldots, b_6)$ associated with the second sequence S_2 of peers. To get this new vector, we remove the first bit b_1 of the previous vector and we add the new bit b_6 at the end. We get for example $(0,1,0,0,1)$ if $b_6 = 1$. Two peer failures appear in the sequence S_2, and so we do not have a data loss. If for example $b_7 = 1$, then the vector associated with S_3 is $(1,0,0,1,1)$. In that case a data loss is found.

We now want to compute the probability to find at least one "bad" sequence S_i containing at least $r+1$ bits 1 in its vector. We use a discrete time discrete space Markov chain to represent the transitions between sequences. Indeed, the set of states V of such Markov chain is the set of all possible binary-vectors of size $s+r$ such that the sum of its elements is at most r, plus an absorbing state namely v_{dead} (containing all other binary-vectors of size $s+r$ in which the sum of its elements is greater than r). For a binary-vector $(b_i, b_{i+1}, \ldots, b_{i+s+r-1})$, we have two possible transitions: $(b_{i+1}, \ldots, b_{i+s+r-1}, 1)$ with probability α and $(b_{i+1}, \ldots, b_{i+s+r-1}, 0)$ with probability $1-\alpha$. One of these vectors (states) could belong to v_{dead}. Remark that we can see this Markov chain as a De Bruijn graph [8].

Consider the previous example with $s = 3$ and $r = 2$. Figure 8 describes the two possible transitions from the state $(1,0,0,1,0)$ (corresponding to the current sequence S_i): the last peer of the next sequence S_{i+1} is failed with probability α, and it is not failed with probability $1-\alpha$. The two

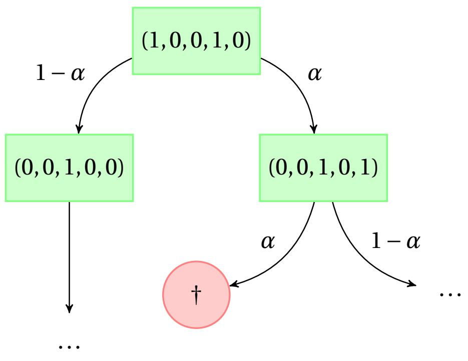

Figure 8: Sample part of the Markov chain for $s + r = 5$ and $r + 1 = 3$.

possible states are $(0, 0, 1, 0, 1)$ and $(0, 0, 1, 0, 0)$, respectively. Furthermore from state $(0, 0, 1, 0, 1)$, it is possible to transit to state v_{dead} because with probability α the vector of the next sequence is $(0, 1, 0, 1, 1)$.

First, we assume that the N peers are ordered in a *line* instead of a *ring*. In other words we do not take into consideration such vectors of sequences: $(\ldots, b_N, b_1, \ldots)$. In that case we look at $N - (s + r) + 1$ sequences. We compute the distribution of probability π after N steps as follows: $\pi = v_0 M^N$ where $v_0 = (0, 0, \ldots, 0)$ is the state without peer failures and M is the transition matrix of our Markov chain. In that case $\mathbb{P}_{line}$ is $\pi(v_{dead})$.

To get the value $\mathbb{P}_{chain}$, we have to carefully take into consideration sequences containing peers on both borders of the network (becoming a ring again). The concerned sequences admit vectors $(\ldots, b_N, b_1, \ldots)$. We get $\pi = \sum_{v \in V} P(v)(v_0 M_{b_{i_1}} \ldots M_{b_{i_{s+r}}} M^{N-(s+r)} M_{b_{i_1}} \ldots M_{b_{i_{s+r-1}}})$ with $P(v)$ the probability to have v as initial state, and M_k, $k \in \{0, 1\}$, the transition matrix replacing α by k.

The number of states of the previously described Markov chain is $|V| = 1 + \sum_{i=0}^{r} \binom{s+r}{i}$ states. Lemma 5 proves that we can reduce this number of states showing some properties.

Lemma 5. *There exists a Markov chain having the same $\pi(v_{dead})$ such that:*

$$|V| = 1 + \sum_{i=0}^{r} \binom{s+r}{i} - \sum_{k=1}^{r} \sum_{j=0}^{k-1} \binom{s+k-1}{j} \tag{8}$$

Proof. One of the peer failures in the chain is meaningful if and only if it can be present in some following chain containing at least $r+1$ failures. For example, in the state $(1, 0, \ldots, 0)$, the first dead is not meaningful because, even if we have r dead peers following, it will be too far away to make a chain with $r + 1$ peer failures. In this sense, states $(0, 0, \ldots, 0)$ and $(1, 0, \ldots, 0)$ are equivalent and we can merge them.

More generally suppose we have k peer failures in the current state (current sequence of peers): we miss $r+1-k$ peer failures to make a data loss; hence, a peer failure in the current sequence will have incidence if and only if it is one of the last $s+k-1$ peers of the chain: otherwise, even if the next $r+1-k$ peers are dead, they won't fit with our k dead in a frame of size $s+r$.

Thus, among all the states with k peer failures, only those where all failures are in the tail of size $s+k-1$ are meaningful. As to the others, the first failures do not matter and we can forget them. This merging algorithm leads us to state space size (8): in a nutshell, we forget all states with k failures and less than k peer failures in the tail of size $s+k-1$. $\square$

We presented a method to compute the exact value of $\mathbb{P}_{chain}$ ($\text{MTTDL}_{chain} = 1/\mathbb{P}_{chain}$). We now propose a simple method to approximate the MTTDL using Absorbing Markov chains techniques. We first consider that the number of peers is infinite. In fact peers numbered i, $i+N$, $i+2N$, …, $i+kN$, … represent the same peer numbered i but at different time steps. Then the corresponding fundamental matrix gives us the average time t_{abs} to absorption, that is the average number of consecutive sequences of peers to find a data loss. Thus $MTTDL_{chain} \approx \lfloor t_{abs}/N \rfloor$. Indeed let P and Q denote the transition matrices of respectively the complete chain (described before) and the sub-chain where we removed the absorbing state and all its incident transitions. Then the fundamental matrix $R = (I-Q)^{-1}$ gives us the time to absorption t_{abs} starting from any state (see [26] for details). t_{abs} is not exactly the MTTDL since $N-(s+r)$ steps correspond to one time step (we survey the whole ring). Hence, $\lfloor t_{abs}/N \rfloor$ gives us the expected number of *time* steps before we reach the absorbing state, which is, this time, the MTTDL we are looking for.

5.4.2 Analytical Approximation

In the rest of this section, a *syndrome* is a sequence of $s+r$ consecutive peers containing at least $r+1$ peer failures. Under the assumption that α is "small enough" (we will see how much), we can derive an analytical expression of the MTTDL.

$$\boxed{MTTDL_{chain} \approx \frac{1}{N\frac{r+1}{s+r}\binom{s+r}{r+1}\alpha^{r+1}}.} \tag{9}$$

Let us begin with two lemmas.

Lemma 6. *The probability to have two distinct syndromes is negligible compared to the probability to have only one and bounded by*

$$\mathbf{P}\left[\exists \textit{ two distinct syndromes} \mid \exists \textit{ a syndrome}\right] < \frac{\alpha N(s+r)\cdot(\alpha(s+r))^{r-1}}{r!} \tag{10}$$

Proof. The probability for a syndrome to begin at a given peer (the beginning of a syndrome being considered as his first peer failure) is given by $p = \alpha \sum_{i=r}^{s+r-1} \binom{s+r-1}{i} \alpha^i (1-\alpha)^{s+r-1-i}$. Meanwhile, we have

$$\mathbf{P}\left[\exists \text{ 2 distinct syndromes}\right] = \mathbf{P}\left[\cup_{|i-j|\geq s+r} \exists \text{ 2 syndromes beginning at peers } i \text{ and } j\right],$$

which is $\leq \binom{N}{2} p^2 < (pN)^2$. Normalizing by pN gives us the probability to have two syndromes knowing that there is at least one:

$$\mathbf{P}\left[\exists \text{ two distinct syndromes} \mid \exists \text{ a syndrome}\right] < pN.$$

Hence, we would like to show that pN is negligible. An upper bound on p is easy to figure out: given that $\alpha(s+r) \ll 1$, we have $p \approx \binom{s+r-1}{r} \alpha^r (1-\alpha)^{s-1} \leq (\alpha(s+r))^r / r!$, and so $pN \leq (\alpha N(s+r))(\alpha(s+r))^{r-1}/r!$. Hence, assuming $\alpha N(s+r) \ll 1$ (or otherwise $r \geq \log N$) suffices to conclude. □

Lemma 7. *The probability to have more than $r+1$ dead peers in a given syndrome is negligible and bounded by*

$$\mathbf{P}\left[\exists > r+1 \textit{ dead peers} \mid \exists \geq r+1 \textit{ peers}\right] < \alpha(s+r) \tag{11}$$

Proof. Since we are working in a syndrome, the probability we want to bound is, in a given chain:

$$\begin{aligned}\mathbf{P}\left[\exists > r+1 \text{ dead peers} \mid \exists \geq r+1 \text{ dead peers }\right] &= \frac{\sum_{r+2}^{s+r} \binom{s+r}{i} \alpha^i (1-\alpha)^{s+r-i}}{\sum_{r+1}^{s+r} \binom{s+r}{i} \alpha^i (1-\alpha)^{s+r-i}} \\ &\leq \frac{\sum_{r+2}^{s+r} \binom{s+r}{i} \alpha^i (1-\alpha)^{s+r-i}}{\binom{s+r}{r+1} \alpha^{r+1} (1-\alpha)^{s-1}}\end{aligned}$$

Since the ratio between a term of the binomial series and its predecessor is $\frac{\alpha}{1-\alpha} \cdot \frac{s+r-i}{i+1}$, we can bound the tail of the binomial sum by a geometric series of common ratio $q = \frac{\alpha}{1-\alpha} \cdot \frac{s-1}{s+r} \ll 1$. Thus we have:

$$\begin{aligned}\mathbf{P}\left[\exists > r+1 \text{ dead peers} \mid \exists \geq r+1 \text{ dead peers}\right] &< \frac{\alpha}{1-\alpha} \cdot \frac{s-1}{r+2} \cdot \frac{1}{1-q} \\ &< \alpha(s+r) \ll 1.\end{aligned}$$

□

Therefore, if we only look for a single syndrome with exactly $r+1$ dead peers, we get a close approximation of the MTTDL.

$$\begin{aligned}\mathbb{P}_{chain} &= \mathbf{P}\left[\exists \text{ one syndrome}\right] \\ &= \mathbf{P}\left[\cup_i \exists \text{ one syndrome beginning at peer } i\right] \\ &= (N-(s+r))p\end{aligned}$$

Indeed, since there is only one syndrome, the events [syndrome begins at peer i] are exclusives. Here p is the probability for the syndrome to begin at a given peer, which we saw in proof of lemma 6. Given lemma 7, we can approximate it by $\binom{s+r-1}{r}\alpha^{r+1}(1-\alpha)^{s-1}$, which leads us too:

$$\mathrm{MTTDL}'_{chain} \approx \frac{1}{N\binom{s+r-1}{r}\alpha^{r+1}} \tag{12}$$

One may notice that this is the same formula as (2) in the Buddy case with $c = N\frac{r+1}{s+r}$.

5.4.3 Validity of the approximation

Numerical results suggests that the Equation (12) is a good approximation for $\alpha < 10^{-3}$, while s have little influence (and r almost none) on the relative variation between simulation and approximation.

5.5 Discussion

The approximations given by the Equations (2), (7), and (9) give an interesting insight on the relation between the placement policies. For instance, note that the ratio between MTTDL_{buddy} and MTTDL_{chain} does not depend of N, nor B, nor s. When $B \ll \binom{N}{r+1}$, the ratio between MTTDL_{buddy} and MTTDL_{global} depends on the number of fragments per disk $B(s+r)/N$.

$$\frac{\mathrm{MTTDL}_{buddy}}{\mathrm{MTTDL}_{chain}} \approx r+1, \quad \frac{\mathrm{MTTDL}_{buddy}}{\mathrm{MTTDL}_{global}} \approx \frac{B(s+r)}{N}, \quad \frac{\mathrm{MTTDL}_{chain}}{\mathrm{MTTDL}_{global}} \approx \frac{B(s+r)}{N(r+1)}.$$

We succeeded in quantifying the MTTDL of the three policies. The Buddy policy has the advantage of having a larger MTTDL than the Chain and the Global. However, when a failure occurs a large number of reconstructions start. When the bandwidth available for reconstruction is low, the reconstructions are delayed which may lead to an increased failure rate. This trade-off has still to be investigated.

6 Conclusion

In this paper, we show that placement policies strongly impact the performance of P2P storage systems. We study three different policies, a Global and two *local*, and show that under resource constraints, the Global policy behaves better in terms of probability to lose data and MTTDL than the *local* policies.

We suggest architectural choices to improve the performances of local policies. We show that, by using a new reconstruction strategy, namely *external reconstruction*, and by increasing the size of the neighborhood, local policies can have performances almost equivalent to the ones of the Global, while keeping their practical advantages.

References

[1] Sara Alouf, Abdulhalim Dandoush, and Philippe Nain. Performance analysis of peer-to-peer storage systems. *Proceedings of the 20th International Teletraffic Congress (ITC)*, LNCS 4516:642–653, June 2007.

[2] Christopher Batten, Kenneth Barr, Arvind Saraf, and Stanley Trepetin. pStore: A secure peer-to-peer backup system. Technical Memo MIT-LCS-TM-632, Massachusetts Institute of Technology Laboratory for Computer Science, October 2002.

[3] Ranjita Bhagwan, Kiran Tati, Yu chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: System support for automated availability management. In *Proceedings of the 1st USENIX symposium on Networked Systems Design and Implementation (NSDI)*, pages 337–350, 2004.

[4] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. *ACM SIGMETRICS Performance Evaluation Review*, 28(1):34–43, June 2000.

[5] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M. Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient replica maintenance for distributed storage systems. In *Proceedings of USENIX symposium on Networked Systems Design and Implementation (NSDI)*, pages 45–58, Berkeley, CA, USA, 2006.

[6] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 202–215, Canada, October 2001.

[7] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a DHT for low latency and high throughput. In *Proceedings of Usenix symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, California, March 2004.

[8] N. G. De Bruijn. A combinatorial problem. *Kibern. Sb., Nov. Ser.*, 6:33–40, 1969.

[9] John R. Douceur and Roger Wattenhofer. Competitive hill-climbing strategies for replica placement in a distributed file system. In *DISC '01: Proceedings of the 15th International Conference on Distributed Computing*, pages 48–62, London, UK, 2001. Springer-Verlag.

[10] John R. Douceur and Roger P. Wattenhofer. Large-scale simulation of replica placement algorithms for a serverless distributed file system. In *Proceedings of Intl. Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, volume 0, pages 311–322, 2001.

[11] P. Druschel and A. Rowstron. PAST: a large-scale, persistent peer-to-peer storage utility. In *Proceedings of 8th Workshop on Hot Topics in Operating Systems*, pages 75–80, Schoss Elmau, Germany, May 2001.

[12] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, October 2003.

[13] Frédéric Giroire, Julian Monteiro, and Stéphane Pérennes. Peer-to-peer storage systems: a practical guideline to be lazy. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, Miami, USA, December 2010. To appear.

[14] A. V. Goldberg and P. N. Yianilos. Towards an archival intermemory. In *ADL '98: Proceedings of the Advances in Digital Libraries Conference*, page 147, Washington, DC, USA, 1998. IEEE Computer Society.

[15] Andreas Haeberlen, Alan Mislove, and Peter Druschel. Glacier: highly durable, decentralized storage despite massive correlated failures. In *Proceedings of USENIX symposium on Networked Systems Design and Implementation (NSDI)*, pages 143–158, Berkeley, USA, 2005.

[16] Magnus Karlsson, Mallik Mahalingam, Magnus Karlsson, and Mallik Mahalingam. Do we need replica placement algorithms in content delivery networks. In *7th International Workshop on Web Content Caching and Distribution (WCW)*, August 2002.

[17] Salma Ktari, Mathieu Zoubert, Artur Hecker, and Houda Labiod. Performance evaluation of replication strategies in dhts under churn. In *MUM '07*, pages 90–97, New York, USA, 2007. ACM.

[18] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, et al. OceanStore: an architecture for global-scale persistent storage. *ACM SIGARCH Computer Architecture News*, 28(5):190–201, 2000.

[19] Qiao Lian, Wei Chen, and Zheng Zhang. On the impact of replica placement to the reliability of distributed brick storage systems. In *International Conference on Distributed Computing Systems (ICSCS'05)*, pages 187–196, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[20] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing (PODC'02)*, pages 233–242, 2002.

[21] M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, D.A. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proceedings of the 29th annual ACM symposium on Theory of computing*, pages 150–159. ACM New York, NY, USA, 1997.

[22] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of ACM International Conference on Management of Data (SIGMOD'88)*, pages 109–116, New York, NY, USA, 1988.

[23] Fabio Picconi, Bruno Baynat, and Pierre Sens. Predicting durability in dhts using markov chains. In *Proceedings of the 2nd Intl. Conference on Digital Information Management (ICDIM)*, volume 2, pages 532–538, Oct. 2007.

[24] Sriram Ramabhadran and Joseph Pasquale. Analysis of long-running replicated systems. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–9, Barcelona, Catalunya, Spain, April 2006.

[25] Rodrigo Rodrigues and Barbara Liskov. High availability in dhts: Erasure coding vs. replication. In *Workshop on Peer-to-Peer Systems (IPTPS), Peer-to-Peer Systems IV*, pages 226–239. LNCS, 2005.

[26] Sheldon M. Ross. *Introduction to Probability Models, Ninth Edition.* Academic Press, Inc., Orlando, FL, USA, 2006.

[27] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 188–201, New York, NY, USA, 2001.

[28] Robbert van Renesse. Efficient reliable internet storage. In *Workshop on Dependable Distributed Data Management*, Florianopolis, Brazil, October 2004.

[29] Robbert van Renesse and Fred B. Schneider. Chain replication for supporting high throughput and availability. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation (OSDI)*, pages 7–7, Berkeley, CA, USA, 2004.

[30] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Revised papers from the 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, volume LNCS 2429, pages 328–337, Cambridge, MA, USA, 2002.