P2P Storage Systems: How Much Locality Can They Tolerate?

Frédéric Giroire, Julian Monteiro, Stéphane Pérennes

▶ **To cite this version:**

HAL Id: inria-00408078
https://inria.hal.science/inria-00408078

Submitted on 28 Jul 2009

INRIA

P2P Storage Systems: How Much Locality Can They Tolerate?

Frédéric Giroire — Julian Monteiro — Stéphane Pérennes

N° 7006

Juillet 2009

Thème COM

Rapport de recherche

P2P Storage Systems: How Much Locality Can They Tolerate?

Frédéric Giroire* , Julian Monteiro* , Stéphane Pérennes*

Thème COM — Systèmes communicants
Projets Mascotte

Rapport de recherche n° 7006 — Juillet 2009 — 20 pages

Abstract: Large scale peer-to-peer systems are foreseen as a way to provide highly reliable data storage at low cost. To achieve high durability, such P2P systems encode the user data in a set of redundant fragments and distribute them among the peers. We study here the impact of different data placement strategies on the system performance when using erasure codes redundancy schemes. Several practical factors (easier control, software reuse, latency) tend to favor data placement strategies that preserve some degree of locality. In this paper, we compare three policies: two of them *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. Finally, we propose a new *external reconstruction* strategy that greatly improves the performance of local placement strategies.

Key-words: P2P storage system, data placement, performance evaluation, data durability

* MASCOTTE, INRIA, I3S, CNRS, Univ. Nice Sophia, Sophia Antipolis, France, firstname.lastname@sophia.inria.fr

Systèmes de Stockage Pair-à-Pair : Quelle degré de Localité Peuvent-ils Tolérer ?

Résumé : Les systèmes pair-à-pair à grande échelle ont été proposé comme un moyen fiable d'assurer un stockage de données à faible coût. Pour assurer la pérennité des données sur une période très longue, ces systèmes codent les données des utilisateurs comme un ensemble de fragments redondants qui sont distribués entre les pairs. Nous étudions ici l'impact de différentes stratégies de placement sur les performances de systèmes utilisant des codes correcteurs comme méthode d'introduction de redondance. En pratique, les stratégies de placement qui préservent un certain degré de localité sont souvent utilisées en raison de différents avantages comme un contrôle plus facile, une latence plus faible ou la réutilisation de codes écrits pour les DHTs (Distributed Hash Tables) par exemple. Dans ce papier, nous comparons trois politiques : pour deux d'entre *locales* les données sont stockées sur des voisins logiques et pour la dernière *globale* les données sont réparties dans tout le système. Nous nous intéressons à l'étude de la probabilité de perdre un bloc de données et à l'utilisation en bande passante nécessaire pour maintenir la redondance. Nous montrons, à l'aide de simulations, que si les ressources ne sont pas limitées, les valeurs moyennes sont les mêmes pour les différentes politiques de placement. Cependant, les variations de l'utilisation de bande passante sont beaucoup plus irrégulières pour les politiques *locales*. Quand la bande passante est limitées, ces fortes variations induisent un temps de maintenance beaucoup plus long et en conséquence un risque plus fort de perdre des données. Nous montrons ensuite qu'un degré de localité approprié peut être introduit de faon à obtenir à la fois l'efficacité de la politique globale et les avantages pratiques d'un placement local. Enfin, nous proposons une stratégie de *reconstruction extérieure* qui améliore grandement les performances des politiques de placement locales.

Mots-clés : système de stockage pair-à-pair, P2P, placement de donnés, évaluation de performance, durabilité des données

1   Introduction

The key concept of Peer-to-Peer 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* [22, 5, 3], in which identical copies of data are sent to different nodes in the system; or be based on *Erasure Codes* [16, 25], such as Reed Solomon and Tornado, as used by some RAID schemes [17]. 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 [25]).

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) [15]. 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 [7, 14]. In this paper, we study three different strategies of data placement. The first, a *global & random* placement policy, spreads the fragments on peers taken randomly among all the peers of the system (see [1, 23, 4]). 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 [22, 5, 8]). The Buddy policy roughly corresponds to 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 [4]. However, the use of *local* strategies brings practical advantages [6]. 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. Basically, their complexities (i.e. time, bandwidth, and space overhead) are all in $O(s + r)$. Conversely, for the Global placement, it is $O(N)$, with N the number of peers in the system. 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 study these 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.

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 fewer 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.

- Finally, we 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.

The remainder of this paper is organized as follows: after presenting the related work, we describe the system characteristics in the next section. 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 our conclusions.

Related Work.

The majority of existing or proposed systems, e.g. Intermemory [10], CFS [5], Farsite [8], Pastry [21], TotalRecall [2], Glacier [11], use a local placement policy. For example, in PAST [22], 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 [13] or GFS [9]. GFS spreads chunks of data on any server of the system using a pseudo-random placement.

Chun et al. in [4] also discuss the local placement (namely *small scope*) impacts. 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.

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	# of peers	$1,005$
F	total of fragments in the system	$1.5 \cdot 10^6$
s	# of fragments in the initial block	9
r	# 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.

In [14] the authors study the impact of data placement on the Mean Time to Data Loss (MTTDL) metric. 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 [24]. Others are focused on Content Delivery Networks [12], 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.

2 System Description

2.1 The System

The detailed characteristics of the studied P2P storage system are presented in this section.
Data Redundancy. Erasure Codes schemes [16] 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, 14, 19, 18], 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 [20] (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 pol-*
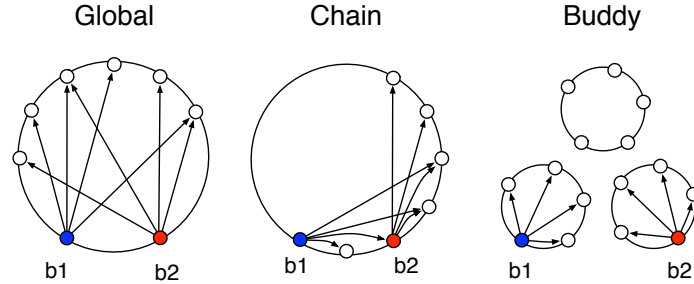
Figure 1: Placement of two blocks $b1$ and $b2$ in the system. Global: $s + r$ fragments are placed at random among all peers; Chain: fragments are placed on $s + r$ neighboring peers; Buddy: many small subsystems of size $s + r$, in this case all peers inside each small group contain the same data.

icy 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 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

The different placement policies studied in this paper are detailed in the following and depicted in Figure 1.

- **Global Policy:** The $s+r$ pieces 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 reconstructing it is also selected among all peers in the system;

- **Chain Policy:** In this *local* policy, each block is managed by a given peer (e.g., the peer with id in the DHT closest to the block id). Its fragments are stored on the $s+r$ "logically" closest (consecutive) neighbors. Note that these neighbors are also in charge of monitoring and reconstructing these blocks.

- **Buddy (or RAID) Policy:** This is an extreme case of a *local* policy, in which the system is composed of small independent subsystems with $s+r$ peers each. It could be seen as a collection of local RAID like storage. In this situation, each peer stores pieces that belong to the $s+r$ peers in the group. To simplify the analysis, we only study system sizes multiple of $s+r$. In this way, each Buddy group contains the same number of peers (otherwise, one of the group has more peers). This does not change the analysis as soon as N is large enough.

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.

Monitored metrics. The simulator keeps detailed traces of different performance metrics. To be sure that we are studying a system in a steady state, the first part of the simulation traces is thrown away. We focus our analysis on three main metrics:

- *Bandwidth*: Average bandwidth consumption per peer, i.e., estimated from the number of pieces 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.

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 amount of stored data and the number of peers are kept constant during the simulation, this means that dead blocks are re-injected in the system. Crashed disks reappear empty. The size of a user data block is 3.6MB. Thus, with $s = 9$, the fragment size is 400KB. With redundancy $r = 6$, the system block size will be 6MB. 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 20 simulated years and the time granularity is 1 hour), we chose 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 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 chose 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.*

Peer bandwidth. We model the bandwidth as a resource constraint per peer, and not as a global shared link constraint as is done in [14, 19]. 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.

When the peer's bandwidth is limited, not all blocks can be reconstructed at the same time. To model a peer's 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 queues of the peers in charge of the reconstruction.

3   Simulation Results

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. In the following, we use these values to analyse 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

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

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

usage across peers; (3) they have the same probability to lose a data-block, (4) but the MTTDLs of the Buddy and the Chain policies are longer.

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 pieces that have to be reconstructed, but they change the repartition of these pieces among peers.
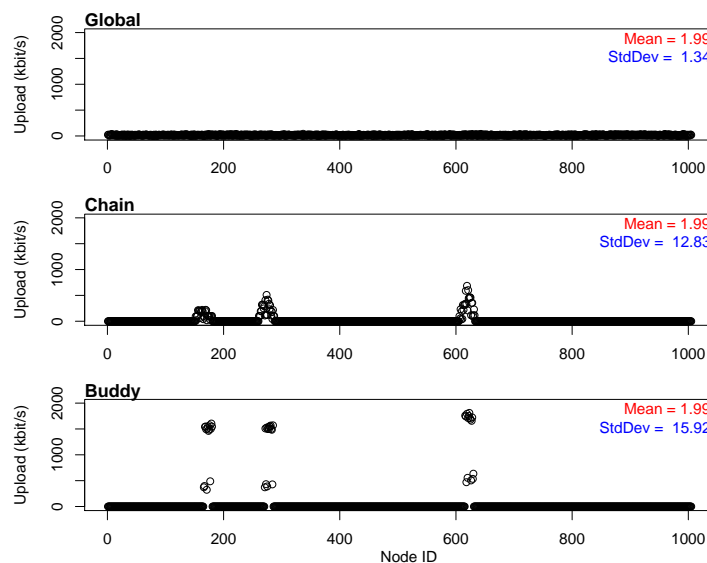

Figure 2: Variations of bandwidth usage across users for the three different strategies for a typical timestep and under the same fault scenario.

However, the variations are not the same. 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 fault 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.

In the case of the Global policy (top graph), the pieces 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 std. deviation of the bandwidth to 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 pieces 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].

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 corresponds 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]. But, we note that the *deviations* during

---

[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 (i.e. $r - r0$ pieces for each block).

time of the number of dead blocks is higher for *local* policies. To explain that, we look at the MTTDL.

3.1.3 Mean Time To Data Loss (MTTDL)

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.
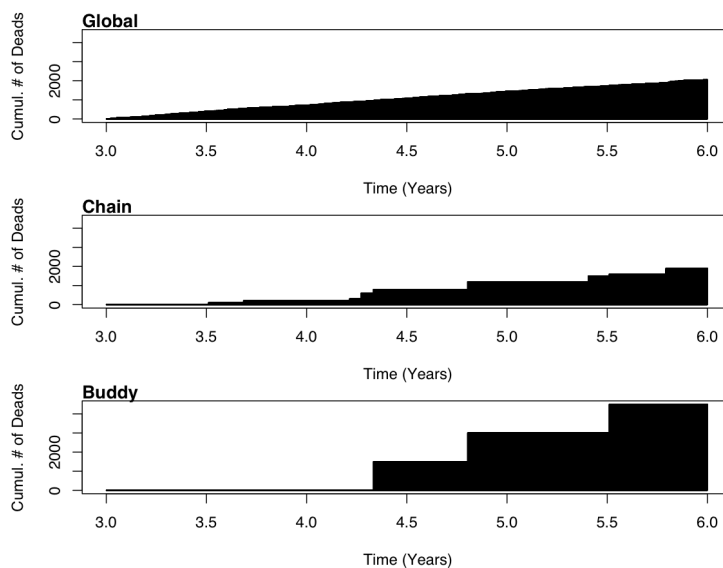

Figure 3: Illustrative example of the cumulative number of dead blocks for a period of three years.

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 fault 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.

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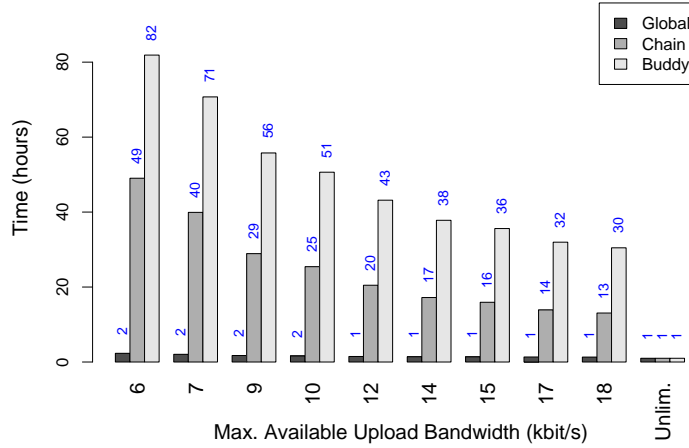
Figure 4: *Average reconstruction time* for different bandwidth limits for the three strategies. The values for the local policies (Chain and Buddy) are higher, and decrease gradually with the increase of the available bandwidth. The unlimited value of 1 is the granularity of our simulator (one hour).

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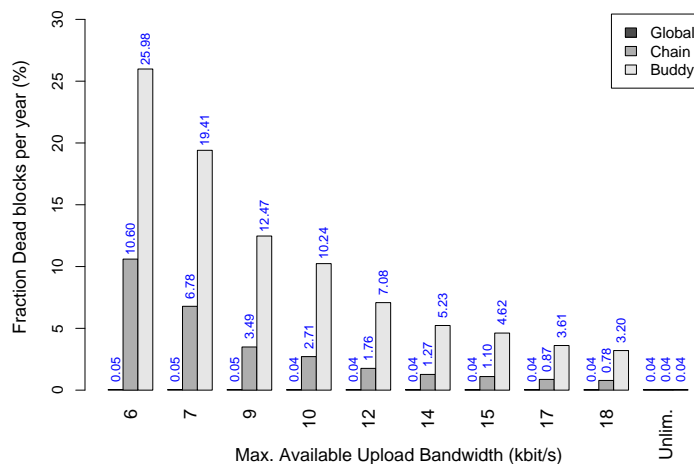
Figure 5: *Fraction of block losses per year* (see Remark 2) for different bandwidth limits for the three strategies. The differences between policies are stronger than for the reconstruction times of Fig 4, as explained in Section 4.2.

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 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 table shows the MTTDL 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	550

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 that the impact of the bandwidth limits per peer needs to be taken into account when analysing 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 improve 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.
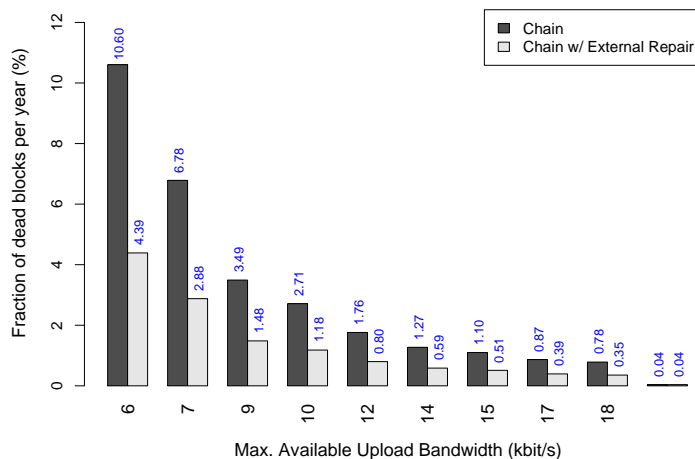
Figure 6: Comparison between the Chain policy with internal reconstruction and with external reconstruction. The fraction of block losses (see Remark 2) for different bandwidth limits is presented. There is an improvement of about 50% on the fraction of data losses.

A rough estimate of the gain in terms of reconstruction time can be given. In the internal reconstruction, local peers have to support $s + r - r_0$ uploads of pieces. However, when using the external reconstruction, they only have to support s uploads of pieces. As the local peers basically are the bottleneck of the reconstruction, the gains in terms of bandwidth and hence of reconstruction time are roughly $1 - s/(s - 1 + r - r_0)$. 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 of the local policies

(a small sub-network to monitor) 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 pieces of this block (of size $s + r$ for Buddy and Chain, and of size N for Global).
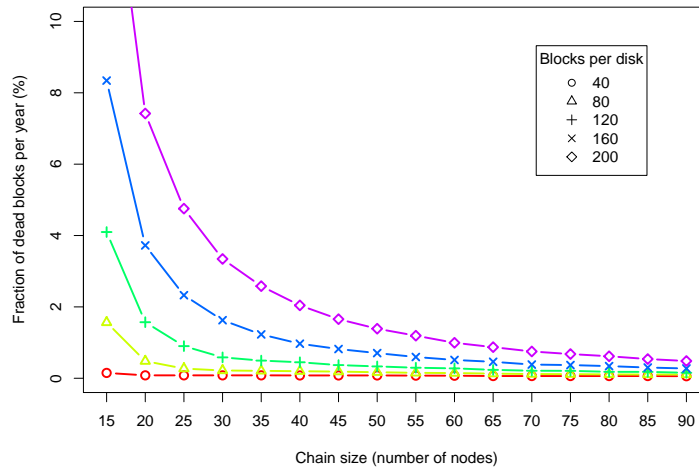

Figure 7: Study of the size of the block neighborhood. Fraction of block losses (see Remark 2) per year for different sizes of neighborhood and different number of fragments per disks.

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:

Table 3: Comparison of Replication and Erasure Codes when using the Chain placement. Number of dead blocks per year and avg. 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 $r = s * k$.

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 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. [23, 8, 5, 14]. It is shown in [25] 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. Rodrigues and Liskov, in [20], state that in high-churn systems, Erasure Codes may demand high bandwidth usage, which could be

impractical. This is not relevant to our study since we assume the use of storage "bricks" that stay turned on almost all the time. 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 $r = k * s$ 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 pieces 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. [23]. Thus, systems with Erasure Codes have less probability to lose data for the same amount of resources and realistic levels of redundancy.

5   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] S. Alouf, A. Dandoush, and P. Nain. Performance analysis of peer-to-peer storage systems. *Internation Teletraffic Congress (ITC), LNCS 4516*, 4516:642, 2007.

[2] R. Bhagwan, K. Tati, Y. chung Cheng, S. Savage, and G. M. Voelker. Total recall: System support for automated availability management. In *Proc. of NSDI*, pages 337–350, 2004.

[3] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. *SIGMETRICS Perform. Eval. Rev.*, 28(1):34–43, 2000.

[4] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *Proc. of NSDI*, pages 45–58, 2006.

[5] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of ACM SOSP*, 2001.

[6] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput. In *Proc. NSDI*, pages 85–98, San Francisco, California, 2004.

[7] J. R. Douceur and R. Wattenhofer. Competitive hill-climbing strategies for replica placement in a distributed file system. In *Proc. of DISC*, pages 48–62, 2001.

[8] J. R. Douceur and R. P. Wattenhofer. Large-scale simulation of replica placement algorithms for a serverless distributed file system. In *Proc. of MASCOTS*, pages 311–319, 2001.

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37:29–43, 2003.

[10] A. V. Goldberg and P. N. Yianilos. Towards an archival intermemory. In *Proc. of ADL Conf.*, page 147, USA, 1998.

[11] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: highly durable, decentralized storage despite massive correlated failures. In *Proc. of NSDI*, pages 143–158, 2005.

[12] M. Karlsson, M. Mahalingam, M. Karlsson, and M. Mahalingam. Do we need replica placement algorithms in content delivery networks. In *Proc. of Workshop on Web Content Caching and Dist.*, 2002.

[13] 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.

[14] Q. Lian, W. Chen, and Z. Zhang. On the impact of replica placement to the reliability of distributed brick storage systems. In *Proc. of ICDCS'05*, volume 0, pages 187–196, 2005.

[15] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proc. of PODC*, 2002.

[16] M. Luby, M. Mitzenmacher, M. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proc. ACM Symp. on Theory of computing*, pages 150–159, 1997.

[17] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proc. of ACM SIGMOD*, 1988.

[18] F. Picconi, B. Baynat, and P. Sens. Predicting durability in dhts using markov chains. In *Proc. of ICDIM*, volume 2, pages 532–538, Oct. 2007.

[19] S. Ramabhadran and J. Pasquale. Analysis of long-running replicated systems. In *Proc. of INFOCOM*, pages 1–9, 2006.

[20] R. Rodrigues and B. Liskov. High availability in dhts: Erasure coding vs. replication. In *Peer-to-Peer Systems IV*, pages 226–239. LNCS, 2005.

[21] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. volume 2218, pages 329–350, 2001.

[22] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proc. ACM SOSP*, pages 188–201, 2001.

[23] R. van Renesse. Efficient reliable internet storage. In *Workshop on Dependable Distributed Data Management*, October 2004.

[24] R. van Renesse and F. B. Schneider. Chain replication for supporting high throughput and availability. In *Proc. of OSDI*, pages 7–7, 2004.

[25] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proc. of IPTPS*, volume 2, pages 328–338, 2002.