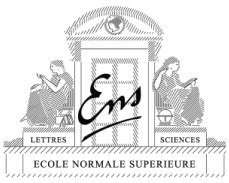
Surveying Different Placement Policies in P2P Storage Systems

Stéphane Caron

September 10, 2009



Abstract

Peer-to-peer systems are foreseen as an efficient solution to achieve reliable data storage at low cost. To deal with common P2P problems such as peer failures or churn, such systems encode the user data into redundant fragments and distribute them among peers. The way they distribute it, known as *placement policy*, has a significant impact on their behavior and reliability.

In this report, after a brief state-of-the-art of the technology used in P2P storage systems, we compare three different placement policies: two of them *local*, in which the data is stored in logical peer neighborhoods, and on of them *global* in which fragments are parted at random among the different peers. For each policy, we give either Markov Chain Models to efficiently compute the Mean Time To Data Loss (which is closely related to the probability to lose data) or approximations of this quantity under certain assumptions.

We also attempt to give lower bounds on P2P storage systems introducing the BIG system, in which we consider information globally. We propose various ways to compute a bound on the probability to lose data, in relation with parameters such as the peer failure rate of the peer bandwidth, though we do not provide a satisfactory algorithm yet.

Contents

1 Introduction to P2P Storage Systems

With information playing an essential role in today's societies, data storage has become a very important issue. Peer-to-peer storage systems provide an efficient, reliable and scalable solution to this problem. In this paper, we will focus on a particular category of such systems, namely *Brick Storage Systems*, where 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.

1.1 Data Redundancy

1.1.1 Redundancy Computation

The key concept of P2P Storage Systems is to distribute redundant data among peers, insuring that, even if some of them come to die, the system will still be able to recover the data it stores. There are multiple ways to compute redundant data. The simplest one is just to copy it multiple times on different peers of the system, as e.g. done in [GGL03]. However, a more efficient way to do that (see e.g. [WK02]) is to use *erasure correcting codes* such as Reed-Solomon or Tornado [LMS+97], where one can choose the level of redundancy he wants. In practice, when it receives a *block* of user data, the system:

- splits it into s fragments;
- adds r redundancy fragments;
- sends the $s+r$ fragments to different peers.

The interest of these codes is that one can reconstruct the initial block using *any* s fragments among the $s+r$ that are stored. Let us illustrate this by summarizing how the Reed-Solomon encoding works. Here we represent fragments as integer numbers sharing the same binary representation, so we have s integers $d_1,\dots,d_s$ and want r redundancy numbers $c_1,\dots,c_r$ from it. Here is how we define (c_i):

$$\begin{matrix} & & \overbrace{\qquad\qquad\qquad}^{s} & \\ s\ \left\{\right. & \left.\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ 1 & 2 & \dots & s \\ 1 & 4 & \dots & s^2 \\ \vdots & \vdots & \dots & \vdots \\ 1 & 2^{r-1} & \dots & s^{r-1} \end{pmatrix}\right. & \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_s \end{pmatrix} = \begin{pmatrix} d_1 \\ \vdots \\ d_s \\ c_1 \\ \vdots \\ c_r \end{pmatrix} \end{matrix} \quad (1)$$

What's interesting with the left matrix M is that any subset of s lines from it makes a free family of vectors (that's why we completed the canonical basis with a Vandermonde family). We can hence see fragment losses among $d_1,\dots,d_s,c_1,\dots,c_r$ as line deletions in M. While we have more than s fragments, we can chose a subvector $(e_1,\dots,e_s)$ from the remaining fragments, select the associated rows in M and invert the subsequent square matrix M'. According to equation 1, we then have:

$$M'^{-1}\begin{pmatrix} e_1 \\ \vdots \\ e_s \end{pmatrix} = \begin{pmatrix} d_1 \\ \vdots \\ d_s \end{pmatrix}$$

This is how we compute back the original data. Anyway, this is only possible if matrix M' is nonsingular. To ensure that this is the case, we should compute in a field $\mathbf{K}$, but we cannot compute in $\mathbf{R}$ with infinite precision, and computing in $\mathbf{Q}$ can be costly. This is why common implementations of the Reed-Solomon encoding work in a finite field $\mathbf{GF}(2^w)$, where w is the machine word size. Working in such fields is easy to implement (see [Pla97] for details).

1.1.2 Data Repair

Data redundancy is not sufficient in itself to guarantee that the system won't ever lose data, hence the need for a mechanism of "data repair" ensuring that the redundancy level (i.e., the number of redundant fragments) is sufficient. Basically, when there is not enough redundancy for a block, some peer retrieves s fragments, computes back the original data and re-inserts the block into the system.

The main parameter in this process is the threshold, in terms of redundancy level, at which we start reconstructing the data. Let us denote it by r_0: as soon as it has $< r_0$ redundant fragments, the systems starts a block's reconstruction. When $r_0 = r - 1$, i.e., when reconstruction starts as soon as the first fragment is lost, we say that the reconstruction policy is *eager*; otherwise we talk about *saddle* reconstruction.

Remark. *We see that an important parameter for us is the mean number $f(i)$ of bits we need to read in order to repair i lost bits, as it has direct implications on the bandwidth usage. In this sense, Reed-Solomon encoding is costly because it has $f(i) = s$ for all i, while bare replication does better with $f(i) = i$.*

1.2 Distributed Hash Tables

So, we have redundant data we want to place among different peers. Each peer has an identifier, therefore a first approach would be to store somewhere a table giving for each block the identifiers of the peers on which its fragments are stored. Anyway this is a centralized approach: we would like some decentralized data structure to do the same with reasonable space and time complexities.

Distributed Hash Tables (DHTs) tackle this issue. They map peer identifiers (e.g. hashing of their IP address) onto a ring $\mathbf{Z}/2^m\mathbf{Z}$ modulo 2^m for some $m \in \mathbf{N}$. Blocks are mapped onto the ring as well using appropriate hash functions. Then, the rule is:

Fact. *A peer is in charged of all blocks whose identifiers are between its own identifier and the one of its predecessor on the ring.*

Hence, when we want to store/retrieve some block, we just have to look up the peer in charge of it in the DHT – which we can do in $O(\log N)$, where N is the number of peers, as we will see – and then ask him where the fragments are/ should be stored.

Essential features of the DHTs is that they are decentralized, scalable, and allow good trade-offs between *peers degrees* (i.e., the number of other peers each brick in the system

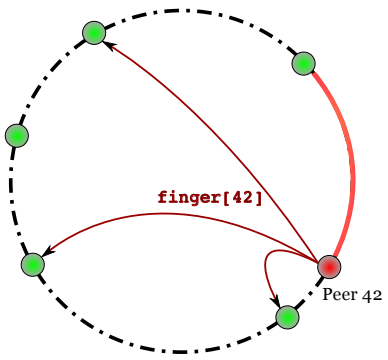


Figure 1: Example Chord DHT Ring.

knows) and *route length* (i.e., the number of peers one has to visit before finding the one in charge of some block). For example, in Chord and Pastry DHTs, both of them are $O(\log N)$.

1.2.1 Finding the peer in charge of a block

Let us consider the Chord approach. In such a DHT, peer identified by k stores a table `finger[k]` of m identifiers ($m \sim \log N$): the one of his successor, of the peer 2 ranks after him in the peers list, then 4 ranks after him, ..., and finally 2^{m-1} ranks after him (figure 1 shows an example of a peer with his links, red arcs corresponding to block IDs peer 42 is in charge of). Then, the strategy to find the peer in charge of an identifier id is to get closer to him *by default*, excluding half of the possible identifiers at each step. This is possible using the `finger` tables, as illustrated by the following pseudo-code:

Algorithm 1 PEER(k).FINDSUCCESSOR(id)

```
if id ∈ ]k, k.successor] (or peer k is alone) then
    return k.successor
else
    for i = m down to 1 do
        if finger[k][i] ∈ ]k, id[ then
            return PEER(i).FINDSUCCESSOR(id)
        end if
    end for
end if
```

As we exclude at least half possible peer identifiers at each step, the route length with such an algorithm is $O(\log N)$. Besides, maintenance cost for Chord DHTs is only $O(\log^2 N)$; anyway, we won't get any further in implementation details (peer insertions, departures, etc.); for details, see e.g. [SMK+01] or [RD01a].

1.2.2 Conclusion

DHTs provide us with a distributed, efficient and scalable way to retrieve the location of a block's fragments among the peers of the system. Now that we have described this aspect

of the system, we will focus on a higher-level layer where we know for each block where its fragments are.

1.3 Placement Policies

The insertion of a new block's fragments into the systems raises an important question: how should these fragments be parted among the different peers ? Should it be totally at random ? Not necessarily, since recent studies (e.g. [GMP09] or [LCZ05]) showed that the *random placement strategy* has its drawbacks. We should then investigate other placement strategies.

We saw that peer IDs are distributed among the ring of a DHT: we will interpret the proximity between these identifiers as a notion of peer neighborhood. This leads us to the three data placement policies we will study thereafter:

Random (or Global) Policy. Block's fragments are sent to $s+r$ peers chosen uniformly at random among the N peers. The peer in charge of the block is also selected among all peers of the system.

Chain Policy. Block's fragments are sent to $s+r$ consecutive peers on the ring, starting from the peer in charge of the block, which is selected uniformly at random among all peers of the system.

Buddy Policy. Peers are grouped into independent clusters of size $s+r$, and a block's fragments are sent to all peers of a cluster chosen uniformly at random.

The two later policies are called *local* ones because their placement classes* are sets of neighboring peers. The number of placement classes is far greater in a global policy than in a local one (here it is $\binom{N}{s+r}$ for the random policy, while $\sim N$ for the two others). As a consequence:

- *local policies are more resistant to simultaneous failures*: if there are $s+r$ concurrent failures happening at the same time, they will hit with the same probability any of the $\binom{N}{s+r}$ subsets of peers, and so the probability to hit a local placement class is $\sim N/\binom{N}{s+r}$ (while it is almost 1 for a global policy);
- *global policies provide more repair bandwidth*: with a local placement policy, if some peer dies, only its neighbors can participate in the upload part of the reconstruction of the fragments it stored. This can have a considerable impact on repair times, e.g. if a peer with its disk full comes to die. Global policies avoid this issue with a better balance of the upload bandwidth among all the available peers.

For a survey of these arguments concerning the Random and Chain policies, see [LCZ05].

1.4 Studying P2P Storage Systems

As we want to compare different storage systems, we need to measure the behavior of such systems, and especially their reliability. The main criterion we will focus on is the risk of losing any data stored in the system, which we will quantify as follows.

*i.e., the different ways to part a block's fragments according to the policy

1.4.1 Mean Time To Data Loss

The *Mean Time To Data Loss* (MTTDL) of the storage system is the mean value of the first time at which some block stored in the system is lost.

In practice, we want it to be as large as possible. Anyway, the computation of this measure is not straightforward, and as an intermediate result we may also consider the mean time before a *given* block is lost, which we'll also call "Mean Time To Data Loss" (of the block). In case of ambiguity, we'll write MTTDL_{obj} and MTTDL_{sys} the MTTDLs respectively of the block and of the system.

We also happen to compute the Mean Time Between two consecutive Data Losses (MTBDL) instead of the MTTDL. In this case, we consider that blocks are re-inserted in the system immediately after they are lost, so the amount of data stored is constant, and also that dead peers are immediately replaced with new, empty ones. Then, if the MTTDL is long enough[†], this model admits a *stationary* behavior with MTBDL ≈ MTTDL.

1.4.2 Related Work

Most existing systems use a local placement policy, e.g. *CFS* [DKK+01], *PAST* [RD01b] (which is based on a Pastry DHT [RD01a]) of *TotalRecall* [BTC+04], but others like *OceanStore* [KBC+00] or *GFS* [GGL03] follow a Global approach.

In [LCZ05], the authors study the impact of the Global and Chain policies (which they call random and sequential placement) on the MTTDL, but in the case of systems using bare replication. They highlight drawbacks of both policies and propose a compromise between.

In [CCLZ07], Chen et al. consider a switch-topology-aware model of peers connectivity: they group peers into clusters, themselves connected to a global switch, and suggest a good placement strategy is to distribute replicas globally (among all peers) while doing local repairs (new replicas being sent to the cluster of the peer in charge of the reconstruction). Anyway, in the long run such a strategy leads to nothing but a Buddy configuration.

Table 1: Summary of the notations used throughout the article.

NOTATION	MEANING	EXAMPLE VALUE
N	# of peers in the system	10^5
B	# of blocks in the system	10^6
s	# of fragments in the initial block	9
r	# of redundancy fragments	6
r_0	reconstruction threshold	2
MTBF	peer mean time between failures	3 years
θ	mean repair time	1 day
α	peer failure frequency (i.e., MTBF^{-1})	10^{-8} Hz
γ	repair frequency (i.e., θ^{-1})	10^{-4} Hz
τ	time step for discrete models	1 hour

[†] that is if $\text{MTTDL} \gg T$ where T is the mean time it takes for the system to go from his initial state (all blocks have $s+r$ fragments) to his most probable state

2 Buddy Policy

In this placement policy, the N peers are grouped into c clusters of size $s+r$. The notion of independence is then easy to figure out: a block's state (number of remaining fragments) is fully described by the state of the cluster it is stored in, and all clusters are independent.

We first give an analytical expression of the MTTDL in the simple the case where the placement policy is eager and the reconstruction takes unit time step.

We then focus on saddle reconstruction with reconstruction time θ as a parameter, providing Markov Chain Models to compute the MTTDL with different degrees of approximation. We will check the validity of these approximations and analyse the impact of the different parameters on the behavior of the system.

2.1 Eager reconstruction in unit time

Let us consider a given cluster C and the event [more than $r+1$ failures occur in C] (which is equivalent to a block death since the reconstruction is eager). We have:

$$p := \mathbf{P}[\text{lose data stored in } C] = \sum_{j=r+1}^{s+r} \binom{s+r}{j} \alpha^j (1-\alpha)^{s+r-j} \tag{2}$$

We can then see the clusters as c independent Bernoulli variables with probability p to fail. Hence, the probability to fail during a given time step is $\Pi = 1-(1-p)^c$. If the average number of cluster failures $cp \ll 1$, we have $\Pi \approx cp$ and thus $\text{MTTDL}_{sys} \approx \text{MTTDL}_{obj}/c$[‡].

Since the ratio between two consecutive terms in sum 2 is $\leq (s+r)\alpha$, we can bound its tail by a geometric series and see that is $O((s+r)\alpha)$. Then, given that $(s+r)\alpha \ll 1$, we can approximate $p \approx \binom{s+r}{r+1}\alpha^{r+1}$, which leads us to a simplified expression of the MTTDL:

$$\text{MTTDL} \approx \frac{\text{MTBF}^{r+1}}{c\binom{s+r}{r+1}} \tag{3}$$

2.2 Saddle reconstruction

We shall now consider the more general case where the reconstruction policy is saddle and the reconstruction time θ may be greater than τ.

To compute the MTTDL of *a block*, we model its evolution in the system by a discrete time Markov chain with discrete state space. One state of the chain represents its number of remaining redundancy fragments (e.g. in state i it has $s+i$ fragments available in the system) while transitions $i \to j$ model its evolution during one time step, with failures corresponding to $i > j$ and repairs to $j = r$, $i < r$.

There is also a "dead" state † which means that we lost the block because at some time there were $< s$ fragments remaining. If we know the stationary distribution π of the chain, $\pi[\dagger]$ gives us the block loss frequency and thus the MTBDL of a single object, which we assimilate to MTTDL_{obj}.

[‡]The exact relation between MTTDL_{sys} and Π is given by $\text{MTTDL}_{sys} = \frac{1-\Pi}{\Pi}$.

2.2.1 Approximative Markov Chain

In the first model given in figure 2, we assume failures and repairs happen independently according to a Poisson distribution.

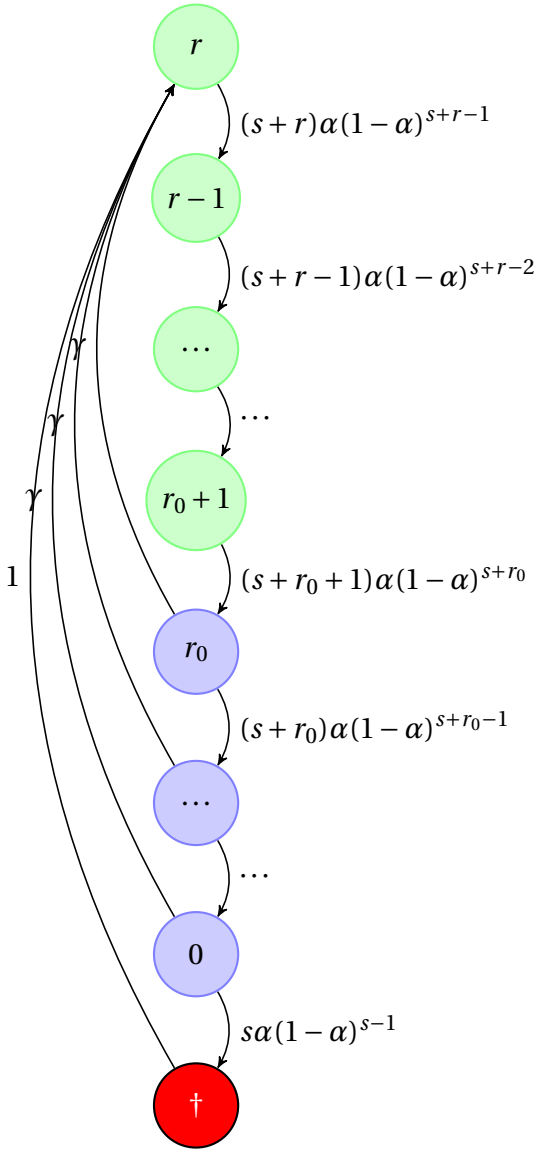


Figure 2: Block's Markov Chain Model for the Buddy policy.

We distinguish three state classes:

- **Non-critical states:** enough fragments, no repair;
- **Critical states:** too few fragments, reconstruction in progress;
- **Dead state:** block cannot be reconstructed.

Downward transitions correspond to peer failures. Since $\alpha \ll 1$, we assume there cannot be multiple failures per time step (this amounts to $\tau \ll$ MTBF). As individual peer failures

follow a Bernoulli distribution, the probability to lose one peer among the $s+i$ active ones in the cluster is $(s+i)\alpha(1-\alpha)^{s+i-1}$. As to repairs (upward transitions), we assume they are not affected by concurrent peer failures (i.e., mean repair time is parameter θ and repairs follow a Poisson distribution), at least while we do not reach the dead state †.

Computing the MTTDL. From the definition of the stationary distribution π of our chain, we find that:

$$\begin{aligned}
\pi(i) &= \left(1+\frac{1}{s+i}\right)(1-\alpha)\pi(i+1) \quad (i \in [\![r_0+1, r-1]\!]) \\
\pi(i) &= \frac{(s+i+1)(1-\alpha)}{s+i+\frac{\gamma}{\alpha(1-\alpha)^{s+i-1}}}\pi(i+1) \quad (i \in [\![0, r_0]\!]) \\
\pi(\dagger) &= s\alpha(1-\alpha)^{s-1}\pi(0)
\end{aligned}$$

Hence, for $i \neq r$, we can define $a_i := \pi(i+1)/\pi(i)$. Given that $\sum_i \pi(i) = 1$, simple linear combinations lead us to:

$$\begin{cases}
\pi(\dagger) + \left(1+\sum_{i=0}^{r-1}\prod_{j=i}^{r-1} a_j\right)\pi(r) &= 1 \\
\pi(\dagger) - \left(s\alpha(1-\alpha)^{s-1}\prod_{i=0}^{r-1} a_i\right)\pi(r) &= 0
\end{cases}$$

These relations make it easy to compute the stationary distribution π, and therefore the MTTDL of the system.

2.2.2 Simplified Chain

Though computable, these results don't give us any idea of the behaviour of the MTTDL and its dependence on the different parameters. Let us then consider an even simpler chain where all failures have the same probability, i.e., $(s+i)\alpha(1-\alpha)^{s+i-1} \longrightarrow \delta$. If we take δ greater than the previous probabilities, this model is pessimistic.

From our calculus of the previous section, we still have $\pi(i) = a_i \cdot \pi(i+1)$ with

$$a_i = \begin{cases}
1 & \text{for } i \in [\![r_0+1, r-1]\!] \\
\frac{\delta}{\gamma+\delta} & \text{for } i \in [\![0, r_0]\!] \\
\delta & \text{for } i = \dagger
\end{cases}$$

Let $\rho := \frac{\delta}{\gamma+\delta}$. The linear relations between $\pi(\dagger)$ and $\pi(r)$ become:

$$\begin{cases}
\pi(\dagger) + \left(r - r_0 + \sum_{i=0}^{r_0}\rho^{i+1}\right)\pi(r) &= 1 \\
\pi(\dagger) - \delta\rho^{r_0+1}\pi(r) &= 0
\end{cases}$$

Which leads us to

$$\pi(\dagger) = \frac{\delta\rho^{r_0+1}}{\delta\rho^{r_0+1} + r - r_0 + \sum_{i=0}^{r_0}\rho^{i+1}} \approx \frac{\delta}{r-r_0}\left(\frac{\delta}{\gamma}\right)^{r_0+1}$$

as by hypothesis we have $\gamma \gg \delta$ and $\delta \ll 1$. In fine,

$$\text{MTTDL} \approx (r-r_0)\cdot\text{MTBF}\cdot\left(\frac{\text{MTBF}}{\theta}\right)^{r_0+1} \tag{4}$$

This expression highlights several relations between the MTTDL and different parameters of the system: the reconstruction threshold $r - r_0$, the MTBF and the reconstruction time θ. Actually, MTTDL is only linearly dependent on $r - r_0$ while the ratio γ/δ between the reconstruction and failure rates is elevated to the power $r_0 + 1$. This suggests that ensuring $\gamma \gg \delta$ is one of the main issues to address in order to make the system reliable.

2.2.3 Complete Chain

Our MCM (Markov Chain Model) made the assumption that τ was small enough so that only one peer failure occurred during one time step; considering multiple failures per time step just makes the computation harder. Anyway, we also implemented a complete MCM, adding transitions $(r+i) \xrightarrow{\delta(i,j)} (r+i-j)$ with $\delta(i,j) = \binom{s+r+i}{j} \alpha^j (1-\alpha)^{s+r+i-j}$. The only difference is that, instead of the calculus we gave in 2.2.1, we used SAGE (see [S+09]) to compute the eigenvector of the transition matrix corresponding to the stationary distribution π.

2.2.4 Comparison of the three models

Figures 3 and 4 show some results of these three models for different values of the system parameters. We see that the simplified model (here we took $\delta = (s+r)\alpha$) is very pessimistic while the two others match well, at least in terms of orders of magnitude. However, approximation (4) seems coherent when $\gamma \gg \alpha$, as all models then have the same behaviour (though the simplified one differs from the other by a consequent multiplicative factor).

Parameter	α	γ	s	r	r_0
Default value	10^{-6} Hz	10^{-2} Hz	15	9	3

Table 2: Parameters for plots 3, 4, 5 and 6.

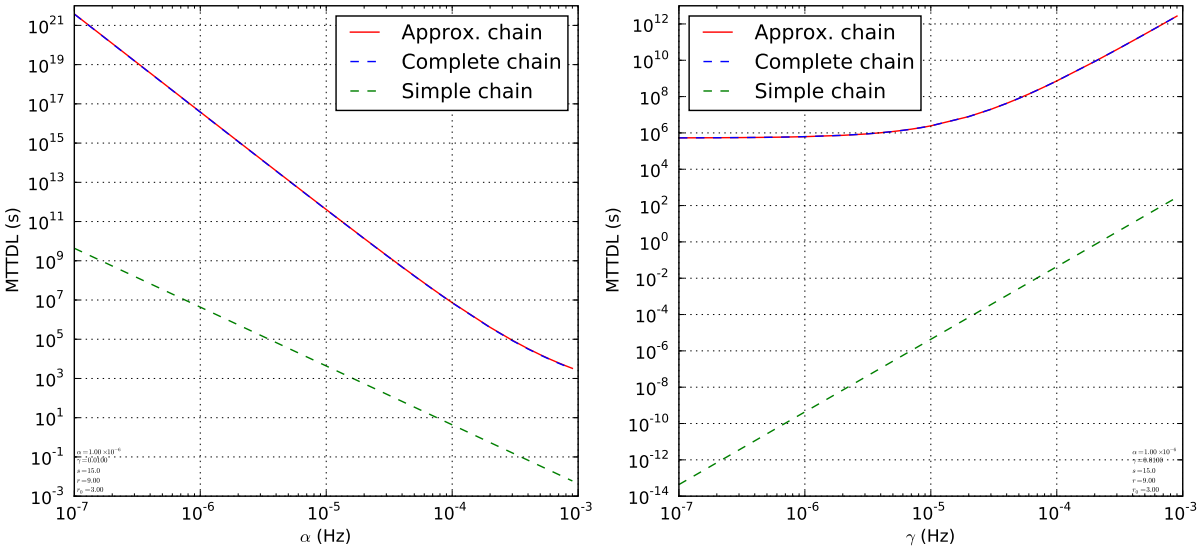


Figure 3: Behavior of the MTTDL with α and γ.

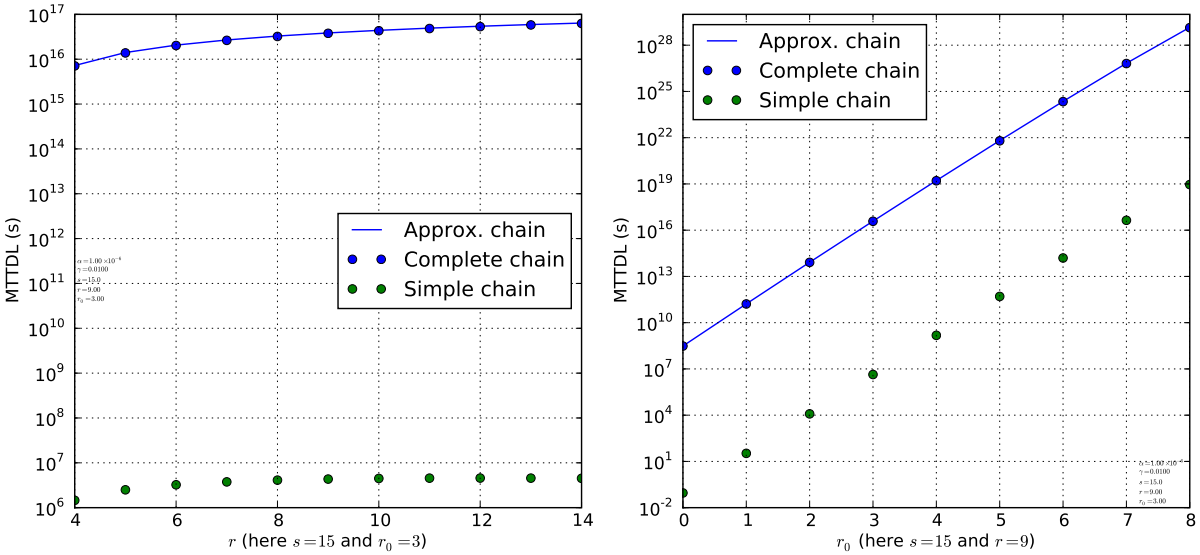


Figure 4: Behavior of the MTTDL with r and r_0.

2.3 Simulation checks

We ran simulations in order to survey the validity of the Poisson approximation (i.e., only one peer failure at a time). We made each parameter varoes in a range of common values and surveyed the impact on the relative variation

$$\Delta = \frac{\left|\text{MTTDL}_{approx} - \text{MTTDL}_{complete}\right|}{\text{MTTDL}_{complete}}.$$

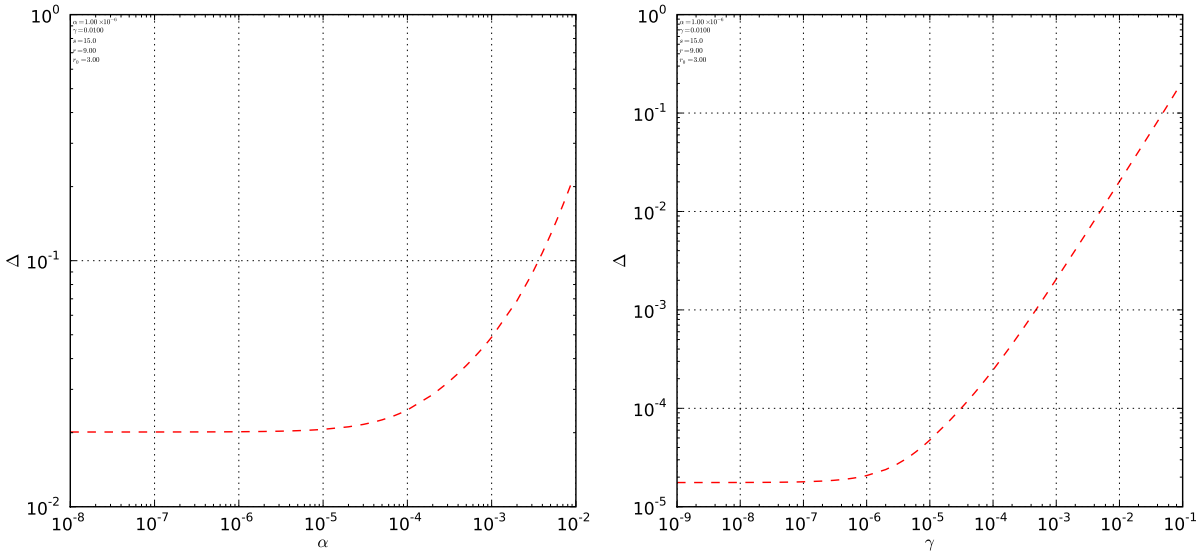


Figure 5: Validity of the approximation when α and γ vary.

We observed that the approximation is valid under the following conditions:

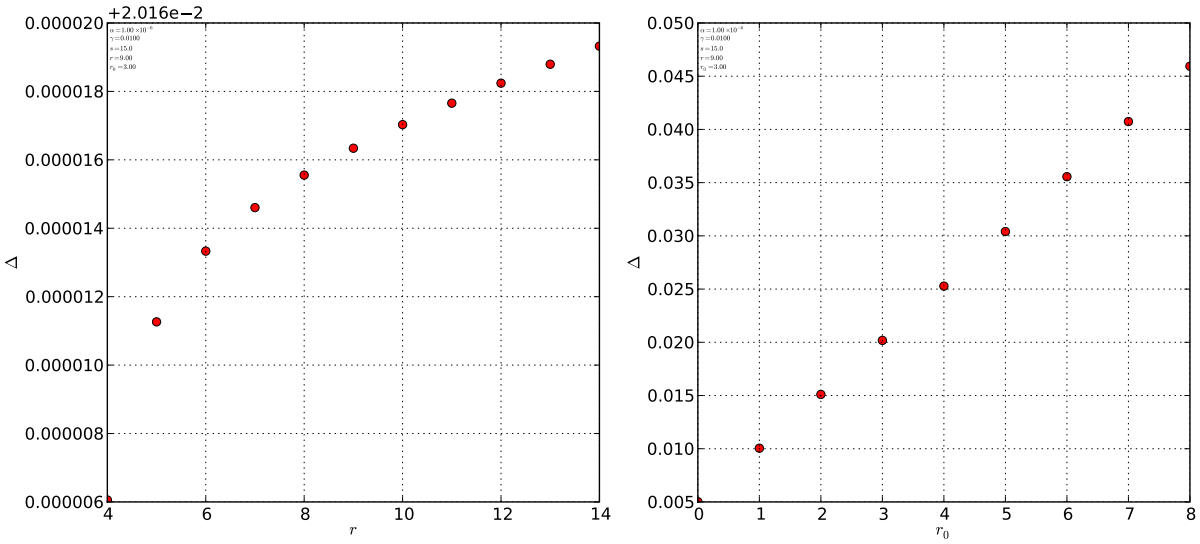


Figure 6: Validity of the approximation when r and r_0 vary.

- $(s+r)\alpha \ll 1$, which is necessary for a Poisson approximation of our Bernoulli process;
- γ *up to three or four orders of magnitude greater than* α, the approximation being very good for long repair times ($\gamma \ll \alpha$ or $\gamma \sim \alpha$): otherwise the relative variation is non-negligible, which we can explain by the fact that repairs are so fast that only multiple concurrent failures can manage to kill a block.

Anyway, s, r and r_0 do not have significant impact on Δ in the small domain in which they take their values (usually $[\![0,20]\!]$).

3 Chain Policy

In this policy, block's fragments are dispatched into "chains" of $s+r$ *consecutive* peers on the identifiers ring. Hence, a block dies when at least $r+1$ peers die among $s+r$ consecutive ones: we denote by "syndrome" such an event and focus on the existence of syndromes to characterize blocks' deaths.

First, we survey the eager reconstruction policy under the "one time step reconstruction" approximation, providing both MCM and analytical expression of the MTTDL. We then show how one can generalize these results to any reconstruction time θ. Anyway we don't handle the saddle case in the present paper; we will do so in some future work.

3.1 Eager reconstruction in unit time step

3.1.1 Markov Chain Approach

Our approach here is to consider a "snapshot" of the system at a given time step. We survey all the consecutive chains on the ring by a Markov process which jumps into its absorbing state when it encounters a "dead" chain (more than $r+1$ dead peers). We then make the hypothesis that *every chain contains at least one block*, which implies that absorption in the Markov chain coincides with detection of a block loss.

We can see a state of the chain as a node $(b_1,\ldots,b_{s+r})$ of a De Bruijn graph representing the states of the peers in the current chain, 1 standing for "dead" and 0 for "available". When we transit from state $(b_1,\ldots,b_{s+r})$ to $(b_2,\ldots,b_{s+r},b)$, we draw state b of the next peer on the ring: since the system's state is memoryless (dead peers are renewed at each time step), $b=1$ (resp. $b=0$) with probability α (resp. $1-\alpha$). If we reach a state $(b_1,\ldots,b_{s+r})$ with $\sum b_i > r$, we transit to the absorbing state of the chain.

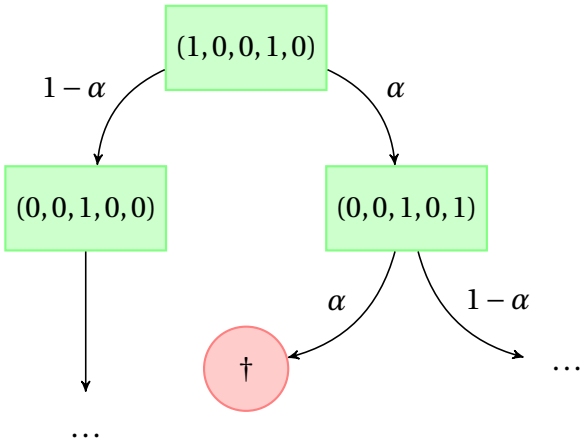


Figure 7: Sample part of the chain for $s+r=5$ and $r+1=3$.

3.1.2 Number of states

With this first approach, the size of the state space is $\#S = 1+\sum_{i=0}^{r}\binom{s+r}{i}$, as all states with more than r deads are mapped into the absorbing one. Actually we can reduce this state

space furthermore.

Lemma 1. *One can find a Markov chain with stationary distribution π having the same $\pi(\dagger)$ and such that:*

$$\#S = 1 + \sum_{i=0}^{r} \binom{s+r}{i} - \sum_{k=1}^{r} \sum_{j=0}^{k-1} \binom{s+k-1}{j} \tag{5}$$

Proof. One of the dead peers in the chain is meaningful if and only if it can be present in some following chain containing at least $r+1$ deads. For example, in the state $(1,0,\dots,0)$, 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 $r+1$ dead peers. In this sense, states $(0,\dots,0)$ and $(1,0,\dots,0)$ are equivalent and we can merge them.
Let's suppose we have k dead peers in the current chain: we miss $r+1-k$ dead peers to make a dead chain; hence, a dead peer in the current chain will have incidence iff 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 deads in a frame of size $s+r$.
Thus, among all the states with k dead peer, only those where all failures are in the tail of size $s+k-1$ are meaningful. As to the others, the first failures don't matter and we can forget them. This merging algorithm leads us to state space size (5) : in a nutshell, we forget all chains with k failures and less than k dead peers in the tail of size $s+k-1$. □

3.1.3 Computing the MTTDL

Let P and Q denote the transition matrices of respectively the complete chain and the subchain 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 starting from any state (suffices to sum its rows, see [GS06] for details).

Anyway, this time to absorption t is not exactly the MTTDL since $N-(s+r)$ chain steps correspond to one time step (we survey the whole ring). Hence, $\lfloor t/N \rfloor$ gives us the expected number of *time* steps before we reach the absorbing state, which is, this time, the MTTDL we are looking for.

3.1.4 Analytical Approximation of the MTTDL

Under the assumption that α is "small enough" (we'll see how much), we can derive an analytical expression of the MTTDL. Let us begin with two lemmas:

Lemma 2. *The probability to have two distinct syndromes is negligible compared to the probability to have only one and bounded by*

$$\mathbf{P}\left[\exists \textit{ two distinct syndromes} \mid \exists \textit{ a syndrome}\right] < \frac{\alpha N(s+r)\cdot(\alpha(s+r))^{r-1}}{r!} \tag{6}$$

Proof. The probability for a syndrome to begin at a given peer (the beginning of a syndrome being considered as his first dead peer) 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

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

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'd 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 3. *The probability to have more than $r+1$ dead peers in a given syndrome is negligible and bounded by*

$$\mathbf{P}\left[\exists > r+1 \textit{ dead peers} \mid \exists \geq r+1 \textit{ peers}\right] < \alpha(s+r) \tag{7}$$

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

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

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:

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

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

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

Indeed, since there is only one syndrome, the events $\left[\text{syndrome begins at peer } i\right]$ are exclusives. Here p is the probability for the syndrome to begin at a given peer, which we saw in proof of lemma 2. Given lemma 3, we can approximate it by $\binom{s+r-1}{r} \alpha^{r+1} (1-\alpha)^{s-1}$, which leads us too:

$$\text{MTTDL} \approx \frac{\text{MTBF}^{r+1}}{N\binom{s+r-1}{r}} \tag{8}$$

One may notice that this is the same formula as (3) in the Buddy case with $c = N\frac{r+1}{s+r}$.

3.1.5 Behavior of the MTTDL

Simulations led with common values of the parameters suggest that approximation (8) succeeds in describing the behavior of the MTTDL, as e.g. depicted by figure 8.

Parameter	α	γ	s	r	r_0
Default value	10^{-5} Hz	10^{-2} Hz	7	3	2

Table 3: Parameters for plots 8 and 9.

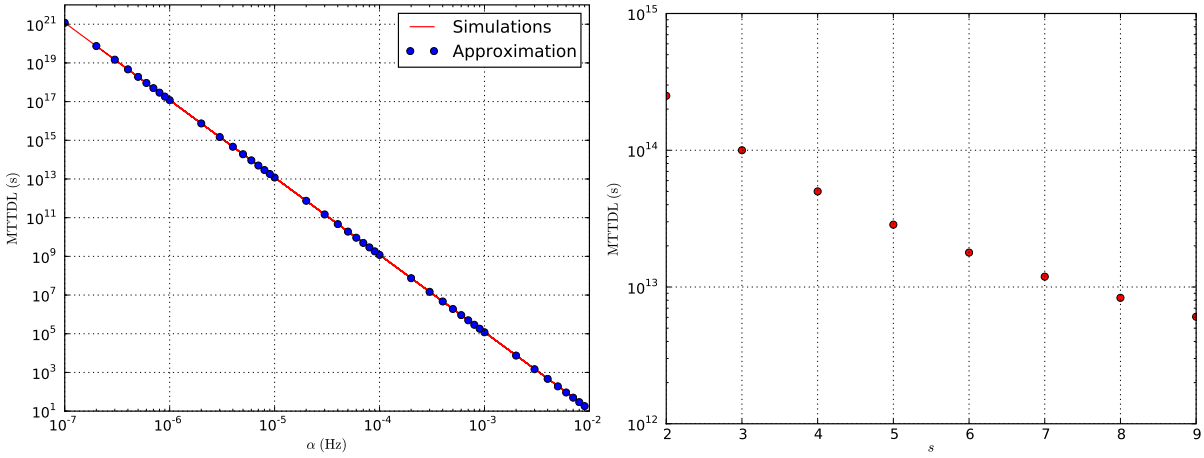


Figure 8: Behavior of the MTTDL when α and s vary.

3.1.6 Validity of the approximation

We've been able to compare the approximation with the exact results given by the MCM in cases where space size (5) was low enough (roughly $s < 15$ and $r < 5$), see figure 9 for sample values. Numerical results suggested formula (8) was a good approximation for $\alpha < 10^{-3}$, s having little influence (and r almost none) on the relative variation Δ between simulation and approximation

3.2 Reconstruction time θ as a parameter

Let us denote by η the probability for a peer to be available (i.e., not under reconstruction) at a given time. Given our failure model, we can compute η using an independent Markov chain.

3.2.1 Reconstruction Chain

Let us consider the MCM given in figure 10. It is easy to compute the stationary distribution of such a chain. From stationarity we infer $\pi(i) = (1-\alpha)\pi(i+1) \Rightarrow \pi(i) = (1-\alpha)^{\theta-i}\pi(\theta)$ and $\pi(0) = (1-\alpha)^{\theta-1}\pi(\theta)$. Then, from $\sum \pi(i) = 1$ we can express $\pi(\theta)$ and get a closed expression for $\eta := \pi(0)$:

$$\eta = \frac{1}{1 + \frac{\alpha(1-\alpha^{\theta})}{(1-\alpha)^{\theta+1}}}$$

For $\theta = 1$, $\eta = 1-\alpha$ and we fall back to the previous case.

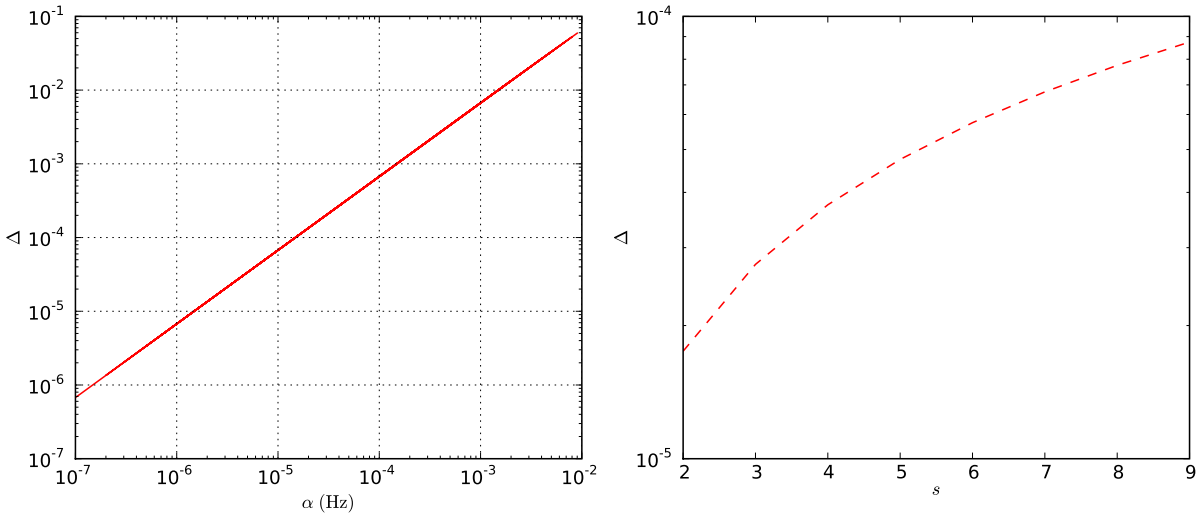


Figure 9: Impact of α and s on the relative variation Δ.

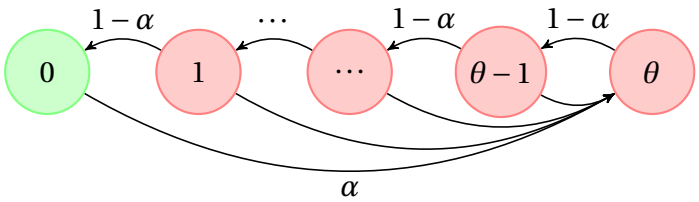


Figure 10: MCM for block reconstruction.

3.2.2 Computing the MTTDL

Even though peers now have more than two possible states due to the repair time, we can still follow our approach of taking a "snapshot" of the system. Indeed, instead of checking whether a peer dies at this time step or not, we can just check whether it is available: if more than $r+1$ peers are unavailable at the same time, even if they are under repair, a block is dead. Hence, we can still use the same MCM: it suffices to replace α by η in our computations.

4 Global Policy

In the Global policy, block's fragments are parted between $s+r$ peers chosen uniformly at random. We also make the assumption that *blocks are inserted independently* into the system, which we will use in subsequent calculus of the MTTDL.

4.1 Eager reconstruction in unit time step

4.1.1 MTTDL calculation

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:

$$\#F = \sum_{j=r+1}^{i} \binom{i}{j}\binom{N-i}{s+r-j} \tag{9}$$

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 p for the chosen class to be in F is:

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

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 $(1-p)^B$. We then come back to the global probability to lose data considering different failure scenarii:

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

Which eventually gives us the MTTDL of the system:

$$\text{MTTDL}^{-1} \approx \sum_{i=r+1}^{N}\binom{N}{i}\alpha^i(1-\alpha)^{N-i}\left[1-\left(1-\frac{\sum_{j=r+1}^{i}\binom{i}{j}\binom{N-i}{s+r-j}}{\binom{N}{s+r}}\right)^B\right] \tag{10}$$

4.1.2 MTTDL approximation

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:

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

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 4. *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: $A_{i+1} = A_i + \binom{i}{r}\binom{N-(i+1)}{s-1}$. □

The ratio between two consecutive terms of sum (10) is:

$$\rho := \frac{\Delta_{i+1}}{\Delta_i} = \frac{\alpha}{1-\alpha}\frac{N-i+1}{i+1}\frac{\Gamma_{i+1}}{\Gamma_i} \approx \alpha N \cdot \frac{\Gamma_{i+1}}{i\Gamma_i} \tag{11}$$

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

$$\Delta_i \approx B\binom{s+r}{r+1}(\alpha N)^{i-(r+1)}\alpha^{r+1}(1-\alpha)^{N-i} \tag{12}$$

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 (12). □

Lemma 6. *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 $\Gamma_i \approx B\left[2(s+r)-3\right]\left(\frac{i}{N}\right)^2$. Hence we see that $\frac{\Gamma_{i+1}}{\Gamma_i} \approx \left(1+\frac{1}{i}\right)^2 \approx 1$, equation (11) leading us to $\rho \approx \alpha N/i$. □

Lemma 7. *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 (11) giving $\rho \leq \alpha N / i$. □

So, where do we go with all these lemmata? Let us remember that, in the systems we consider, we have either $\alpha N \ll 1$ or $\alpha N \sim 1$, i.e., we don't want the mean number of peer failures per time step to get to high (otherwise we can not get a high MTTDL). Hence, lemmas 6 and 7 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 5 can provide us with a stronger result. We must point out that this is a "practical lemma" making many assumption associated with typical numerical values of the system parameters. This being said, under these assumptions, equation (12) 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:

$$\text{MTTDL} \approx \frac{\text{MTBF}^{r+1}}{B\binom{s+r}{r+1}} \tag{13}$$

One should confront it to equations (3) and (8). Here we can see that, under our strong hypotheses taken from usual numerical values of the parameters, the system behaves *as if all blocks were independent.*

5 The BIG system

5.1 Brief Presentation of the BIG system

The BIG system aims at providing lower bounds valid for any P2P storage system, e.g. on the minimum bandwidth needed to maintain redundant data. It is called "BIG" because it conceives its information as a unique, huge block of data.

In this model, we want to be able to store $I_0 N$ bits of information in the system (on average I_0 per peer). Each peer devotes an amount b of his bandwidth to data repair, and peers fail with rate α according to a Poisson process. At time t, the system stores $I_t N$ bits of information (with $I_t > I_0$ since we need to introduce redundancy to avoid data loss), and at each time step it looses a number Z of peers, where Z is distributed as follows:

$$\mathbf{P}[Z = k] = \binom{N}{k} \left(1 - e^{-\alpha\tau}\right)^k \left(e^{-\alpha\tau}\right)^{N-k} \tag{14}$$

We make the assumption that *all peers store the same amount of information.* Moreover, at each time step new fragments are generated: since the bandwidth provisioned by each peer is b, the number of bits created is at most $b\tau N$. Finally, the amount of information stored in this model is driven by the following equation:

$$NI_{t+1} = NI_t + b\tau N - ZI_t \tag{15}$$

If $\alpha\tau N \ll 1$, we can replace Z by a Bernoulli variable which value is 1 with probability $\beta = \alpha\tau N$. Let $\delta = b\tau$, and let us relax our integer model to the set of real numbers: for $x \in \mathbb{R}^+$, the mass at Nx is sent to $N(x+\delta)$ with probability $1-\beta$ and to $(N-1)X + \delta N$ with probability β. Thus, the continuous stationary distribution $f(x)$ of the system satisfies the following functional equation:

$$f(x) = (1-\beta) f(x-\delta) + \beta f\left(\frac{x-\delta}{1-\frac{1}{N}}\right) \tag{16}$$

Now we would like to compute this distribution in order to ensure that the mass $\int_0^{I_0} f(x)\mathrm{d}x$, which is the probability to lose data, is below a certain threshold. This would give us lower bounds on the MTTDL for all P2P storage systems having certain values of parameters α, b, etc.

5.2 Moments of the Distribution

Equation (16) makes it easy to compute the moments $\int x^n f(x)\mathrm{d}x$:

$$\mu_n := \int x^n f(x)\mathrm{d}x = (1-\beta)\int (x+\delta)^n f(x)\mathrm{d}x + \beta \int \left(\left(1-\frac{1}{N}\right)x + \delta\right)^n f(x)\mathrm{d}x$$

Using the binomial theorem, one can part terms in x^n from those in $x^i, i < n$, coming to the relation between μ_n and its predecessors:

$$\mu_n = \frac{\sum_{i=0}^{n-1} \binom{n}{i} \mu_i \delta^{n-i} \left((1-\alpha) + \alpha(1-\epsilon)^i\right)}{\alpha\left(1-(1-\epsilon)^n\right)} \tag{17}$$

Where $\epsilon = \frac{1}{N}$. Hence, we know how to compute easily all the moments of the distribution, and we would like to compute f back. On $\mathbb{R}^+$, this is know as Stieltjes moment problem, and we don't know if there is unicity of the underlying distribution f.

Another approach would be to say we cannot have more than $I_{max}N$ bits of data in the system, so f is definite on $[0, I_{max}]$. Then, the problem of computing f from its moments, known as the Hausdorff moment problem, admits various solutions (see e.g. [IPPT03]). But in this case, the computation of moments μ_n is not as simple as in relation (17) and introduces new unknown quantities (mass of the tail $\int_{I_{max}-\delta}^{I_{max}} f(x)\mathrm{d}x$).

Finally, though it may still be possible to compute the distribution efficiently from its moments, we didn't take this approach to its end.

5.3 Computing the Distribution

We considered approaching distribution f by a discrete one. First, we set a bounded interval $[a, b]$ where we want to study f, and subdivide it into several bins $b_1 = [a, a_1]$, $b_2 = [a_1, a_2]$, $\ldots$, $b_M = [a_{M-1}, b]$. Then, we try to approach f by a simple function $\widehat{f}$ constant on every bin b_i. Assuming bins are small enough, we iterate the following process:

Algorithm 2 Mass redistribution

$\widehat{f} \leftarrow$ actual distribution
$\widehat{g} \leftarrow$ nul distribution
for each bin b_i **do**
  $x \leftarrow$ middle of b_i
  $x_1 \leftarrow x + \delta$
  $x_2 \leftarrow (1-\epsilon)x + \delta$
  $\widehat{g}(\mathrm{bin}(x_1)) += (1-\beta) \times \widehat{f}(x)$
  $\widehat{g}(\mathrm{bin}(x_2)) += \beta \times \widehat{f}(x)$
end for
return $\widehat{g}$

Since we can see mass redistribution (15) as a Markov process, assuming that our discretized model is ergodic, we expect $\widehat{f}$ to converge to an approximation of f. Anyway, this straightforward process is not efficient enough and we had to introduce several tweaks to keep computation feasible, including:

Multiple iterations per redistribution. Algorithm 5.3 applies one iteration of the Markov process to the discrete vector $\widehat{f}$. We found that simulating multiple iterations per step lead to more efficient algorithms in practice.

Centering the interval on the mean. Since the main problem we encountered was the need for a high number of bins (e.g. the diameter of the bins must at least be $\sim \delta$), we considered computing only the central part of the distribution.

Let $\mu := \int xf(x)\mathrm{d}x$: if we take $a = (1 - m\epsilon)\mu$ and $b = (1 + m\epsilon)\mu$ with $\beta^m < 10^{-42}$, the probability to go from state μ to state a is $\leq \beta^m$. This way we were able to set bounds a and b closer to the mean μ (thus reducing the number of steps needed) while keeping sufficient numerical precision.

Yet we still didn't come to satisfying results: though computable, the distributions we got were sometimes significantly different while presenting similar system parameters. We will investigate on this furthermore.

Remark. *One of the issues in dealing with non-observable probabilities (e.g.* $< 10^{-20}$*) is that the bare study of simple trajectories is unfruitful.*

Future Work. *We think we found an expression of the distribution after k failures that would lead us to a lower bound on the bandwidth necessary to maintain the information stored in the system with high probability (e.g.* $> 1 - 10^{-20}$*).*

6 Conclusion

In this paper, we study three placement policies: Global placement on the one hand, Chain and Buddy placement on the other hand. We show that they admit similar approximations of their MTTDLs (cf. equations (3), (8) and (13)). For the local policies we also propose Markov Chain Models admitting various degrees of approximation.

We subsequently attempt to give lower bounds on P2P storage systems with the BIG system, a model in which information is conceived globally as a big block of data. We succeed in characterizing the probability distribution of this system, but do not actually provide an efficient and satisfactory algorithm to compute it. Further improvements could come for convex optimization algorithms.

References

[BTC+04] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: System support for automated availability management. In *In Proc. of NSDI*, pages 337–350, 2004.

[CCLZ07] M. Chen, W. Chen, L. Liu, and Z. Zhang. An analytical framework and its applications for studying brick storage reliability. *26th IEEE International Symposium on Reliable Distributed Systems*, pages 242–252, 2007.

[DKK+01] Frank Dabek, Frans M. Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, volume 35, pages 202–215, New York, NY, USA, December 2001. ACM Press.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *19th ACM Symposium on Operating Systems Principles*, October 2003.

[GMP09] Frédéric Giroire, Julien Monteiro, and Stéphane Pérennes. P2p storage systems: How much locality can they take? 2009.

[GS06] Charles M. Grinstead and Laurie J. Snell. *Grinstead and Snell's Introduction to Probability*. American Mathematical Society, version dated 4 july 2006 edition, 2006.

[IPPT03] Pierluigi Novi Inverardi, Giorgio Pontuale, Alberto Petri, and Aldo Tagliani. Hausdorff moment problem via fractional moments. *Appl. Math. Comput.*, 144(1):61–74, 2003.

[KBC+00] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. Oceanstore: an architecture for global-scale persistent storage. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, volume 28, pages 190–201. ACM Press, December 2000.

[LCZ05] Qiao Lian, Wei Chen, and Zheng Zhang. On the impact of replica placement to the reliability of distributed brick storage systems. *Proc. of ICDCS'05*, 0:187–196, 2005.

[LMS+97] Michael Luby, Michael Mitzenmacher, Amin Shokrollahi, Daniel Spielman, and Volker Stemann. Practical loss-resilient codes. In *In Proceedings of the 29th annual ACM Symposium on Theory of Computing*, pages 150–159, 1997.

[Pla97] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[RD01a] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–?, 2001.

[RD01b] Anthony Rowstrong and Perer Druschel. Pdf storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, volume 35, pages 188–201. ACM Press, 2001.

[S+09] W. A. Stein et al. *Sage Mathematics Software (Version 3.3)*. The Sage Development Team, 2009. `http://www.sagemath.org`.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[WK02] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 328–338, 2002.