Lifetime and availability of data stored on a P2P system: Evaluation of redundancy and recovery schemes

Abdulhalim Dandoush, Sara Alouf, Philippe Nain

Lifetime and availability of data stored on a P2P system: Evaluation of redundancy and recovery schemes

Abdulhalim Dandoush, Sara Alouf and Philippe Nain
adandoush@gmail.com, sara.alouf@inria.fr, philippe.nain@inria.fr
INRIA, B.P. 93, 06902 Sophia Antipolis Cedex, France

Abstract

This paper studies the performance of Peer-to-Peer storage and backup systems (P2PSS). These systems are based on three pillars: data fragmentation and dissemination among the peers, redundancy mechanisms to cope with peers churn and repair mechanisms to recover lost or temporarily unavailable data. Usually, redundancy is achieved either by using replication or by using erasure codes. A new class of network coding (regenerating codes) has been proposed recently. Therefore, we will adapt our work to these three redundancy schemes. We introduce two mechanisms for recovering lost data and evaluate their performance by modeling them through absorbing Markov chains. Specifically, we evaluate the quality of service provided to users in terms of durability and availability of stored data for each recovery mechanism and deduce the impact of its parameters on the system performance. The first mechanism is centralized and based on the use of a single server that can recover multiple losses at once. The second mechanism is distributed: reconstruction of lost fragments is iterated sequentially on many peers until that the required level of redundancy is attained. The key assumptions made in this work, in particular, the assumptions made on the recovery process and peer on-times distribution, are in agreement with the analysis in [12] and in [28] respectively. The models are thereby general enough to be applicable to many distributed environments as shown through numerical computations. We find that, in stable environments such as local area or research institute networks where machines are usually highly available, the distributed-repair scheme in erasure-coded systems offers a reliable, scalable and cheap storage/backup solution. For the case of highly dynamic environments, in general, the distributed-repair scheme is inefficient, in particular to maintain high data availability, unless the data redundancy is high. Using regenerating codes overcomes this limitation of the distributed-repair scheme. P2PSS with centralized-repair scheme are efficient in any environment but have the disadvantage of relying on a centralized authority. However, the analysis of the overhead cost (e.g. computation, bandwidth and complexity cost) resulting from the different redundancy schemes with respect to their advantages (e.g. simplicity), is left for future work.

keyword — Peer-to-Peer network, distributed storage system, performance evaluation, absorbing Markov chain, data availability, system engineering

1 Introduction

Conventional storage solutions rely on robust dedicated servers and magnetic tapes on which data are stored. These equipments are reliable, but they are also expensive and do not scale well. The growth of storage volume, bandwidth, and computational resources for PCs has fundamentally changed the way applications are constructed. Almost 10 years ago, a new network paradigm has been proposed where computers can build a virtual network (called overlay) on top of another network or an existing

architecture (e.g. Internet). This new network paradigm has been labeled peer-to-peer (P2P) distributed network. A peer in this paradigm is a computer that play the role of both supplier and consumer of resources.

This P2P model has proved to be an alternative to the Client/Server model and a promising paradigm for Grid computing, file sharing, voice over IP, backup and storage applications. Some of the recent efforts for building highly available storage or backup system based on the P2P paradigm include OceanStore [25], CFS [8], Total Recall [5], [39] and UbiStorage [37]. Wuala [39], is an example of P2P distributed storage system, with more than 100 million stored files, where a node can simultaneously be supplier and client. Wuala guarantees very high data availability (theoretically with probability 99.9% the data is available). Each new user is immediately allowed to use $1GB$ of free storage space. Moreover, users can buy more storage space and, possibly, share their own local disk spaces. The distributed architecture of Wuala is based on Chord [34]. The recovery process is triggered periodically and managed using a centralized authority and more precisely, is carried out by a few Super-Nodes. Although scalable and economically attractive compared to traditional storage/backup systems, these P2P systems pose many problems such as reliability, data availability and confidentiality.

We can distinguish between backup and storage systems. P2P backup systems aim to provide long data lifetime without constraints on the data availability level or the reconstruction time. For this reason, the backup system designers are interested in the permanent departures of peers rather than the intermediate disconnections, on the contrary to storage systems, even if the disconnections durations were long.

1.1 Redundancy schemes

P2PSS is considered today one of the important applications where benefits of data redundancy and network coding have emerged [16]. A significant research activity has been devoted to evaluating the fundamental tradeoff between the storage space and the amount of data required to repair data.

In a P2P network, peers are free to leave and join the system at any time. As a result of the intermittent availability of peers, redundant data is inserted into the system to ensure high availability of the stored data. Existing systems achieve redundancy either by replication, where there are two replication levels, or by erasure codes (e.g. [31]). The two replication levels are as follows:

- *The whole-file-level replication scheme.* A file f is replicated r times over r different peers (as in PAST [32]) so that the tolerance against failures or peers departure is equal to r as the system maintains $r+1$ copies of the f. The ratio $1/\{r+1\}$ defines the useful storage space in the system. Hereafter, we will refer to this replication scheme as *replication*.

- *The fragment-level replication scheme.* This scheme consists of dividing the file f into s equally sized fragments, and then make r copies of each of them and place one fragment copy per peer, as in CFS [8].

The erasure code (EC) scheme consists of dividing the file f into b equally sized blocks (say S_B bits). Each block of data D is partitioned into s equally sized fragments (say $F_{EC} = S_B/s$ bits) to which, using one of the erasure codes scheme (e.g. [31]), r redundant fragments are added of the same size. Recovering any fragment (if it is lost) or adding a new redundant fragment of a given block of data requires the download of any other s fragments out of the available fragments of that block (downloading again the size of the original block S_B). Therefore, for each stored block of data, the tolerance against failures or peers departure is equal to r. The useful storage space in the system is defined by the ratio $s/(s+r)$. OceanStore [25], Total Recall [5] , Wuala [39], and UbiStorage [37] are some examples of

existing P2P systems that use erasure coding mechanisms to provide some level of system reliability and data availability.

For the same amount of redundancy, erasure codes provide higher availability of data than replication [38, 3].

A new class of codes, so-called regenerating codes (RC) has been proposed recently in [15]. RC can be considered a generalization of erasure code (EC), which reduces the communication cost of EC by slightly increasing the storage cost. The size of fragments in RC is larger of that in EC. In [15], the authors consider in *Theorem 1, p. 5* a simple scheme in which they require that any s fragments (the minimum possible) can reconstruct the original block of data. All fragments have equal size $F_{RC} = \theta * S_B$. A newcommer produces a new redundant fragment by connecting to any s nodes and downloading $\theta S_B/s$ bits from each. In this theorem, the authors assume that the source node of the block of data will store initially n fragments of size θS_B bits on n storage nodes. In addition, newcommers arrive sequentially and each one connects to an arbitrary k-subset of previous nodes (including previous newcommers). They define $\theta_c := \frac{s}{s^2 - s + 1}$ to be, in the worth case, the lower bound on the minimal amount of data that a newcomer must download. The worth case is occured when a data collector (client) need to recover the original block of data from only newcomers. In general, if $\theta \geq \theta_c$ there exists a linear network code so that all data collectors can reconstruct the considered block by downloading s fragments from any s nodes. So, using this simple scheme of RC, adding a new redundant fragment of a given block requires a new peer to download $1/s$ percent of s stored fragments ($\theta S_B/s$ of each) so that the new peer regenerates one random linear combination of the parts of fragments already downloaded. Until the time of writing this paper, the regenerating codes is not yet used in any P2P system. However, we will show through numerical results that RC is a promissing redundancy scheme for P2P storage and backup systems. An interesting survey on the main problems in distributed storage applications with network coding can be found in [17]

1.2 Recovery mechanisms and policies

However, using redundancy mechanisms without repairing lost data is not efficient, as the level of redundancy decreases when peers leave the system. Consequently, P2P storage/backup systems need to compensate the loss of data by continuously storing additional redundant data onto new hosts.

Systems may rely on a central authority that reconstructs fragments when necessary; these systems will be referred to as *centralized-recovery systems*. Alternatively, secure agents running on new hosts can reconstruct by themselves the data to be stored on the hosts disks. Such systems will be referred to as *distributed-recovery systems*. A centralized server can recover at once multiple losses of the same document in the centralized-recovery scheme. In the distributed case, each new host—thanks to its secure agent—recovers only one loss per document.

Regardless of the recovery mechanism used, two repair policies can be enforced. In the *eager* policy, when the system detects that one host has left the network, it immediately initiates the reconstruction of the lost data, and stores it on a new peer upon recovery. This policy is simple but makes no distinction between permanent departures that need to be recovered, and transient disconnections that do not.

Having in mind that connections may experience temporary, as opposed to permanent failures, one may want to deploy a system that defers the repair beyond the detection of a first loss of data. This alternative policy, so-called *lazy*, inherently uses less bandwidth than the eager policy. However, it is obvious that an extra amount of redundancy is necessary to mask and to tolerate host departures for extended periods of time.

The aim of this paper is to develop mathematical models to evaluate fundamental performance metrics

(data lifetime and availability) of P2PSS. Our contributions are as follows:

- Analysis of centralized and distributed recovery mechanisms.
- Proposition of a general model that captures the behavior of both eager and lazy repair policies and the three replication schemes, and accommodates both temporary and permanent disconnections of peers.
- Numerical investigation using realistic parameters values to support the mathematical models and to compare erasure codes with regenerating codes.
- Guidelines on how to engineer a P2P backup or storage system in order to satisfy given requirements.

In the following, Section 2 reviews related work and Section 3 introduces the assumptions and notation used throughout the paper. Sections 5 and 6 are dedicated to the modeling of the centralized- and distributed-recovery mechanism, respectively. In Section 8, we provide some numerical results showing the performance of the centralized and decentralized schemes while either erasure codes or regenerating codes are used. Section 9 concludes the paper.

2 Related Work and Motivation

Although the literature on the architecture and file system of distributed backup and storage systems is abundant, most of these systems are configured statically to provide durability and/or availability with only a cursory understanding of how the configuration will impact overall performance. Some systems allow data to be replicated and cached without constraints on the storage overhead or on the frequency at which data are cached or recovered. These yield to waste of bandwidth and storage volume and do not provide a clear predefined durability and availability level. Hence, the importance of the thorough evaluation of P2P storage systems before their deployment.

There have been recent modeling efforts focusing on the performance analysis of P2P backup and storage systems in terms of data durability and availability such as [30, 1] and [9]. However, in all these models, findings and conclusions rely on the assumption that the peers availability and the recovery process are exponentially distributed with some parameters for the sake of simplification or for the lack of works characterizing their distribution under realistic settings and assumptions.

In [36] a general queuing models for P2P service systems is developed. In modeling P2P storage systems as a queueing system, the study assumes the availability of the file is not an issue thanks to redundancy. Then, it does not consider neither the permanent departure nor the recovery process. The key performance problem would be whether there is sufficient bandwidth to support the download requests. Moreover, the authors assume server online/offline times and file length to follow exponential distributions.

Most of systems that take data availability into account do so in a basic way. The general approach is to leverage one single parameter, the mean peer availability in the system [40].

The question about how to detect or define permanent failures and when trigger a recovery process is typically addressed by the use of timeouts [22, 40]. After a given timeout, a system parameter, a node is declared as failed and a repair is triggered. Authors in [22] propose an approach to replica management based on the availability history of hosts. The approach is achieved by an adaptive per-node timeout, instead of relying on a system-level timeout. On one hand, advanced timeout are often computed with Markov models as in [40] using some assumptions such as homogeneous behavior, or memoryless exponential distributions that are not verified in certain systems [24, 28]. One the other

hand, such an approach cannot guarantee that more than or equal to s fragments (required fragments to regenerate data) are present in the system and we do not have any idea about the waiting time in a state where less than s fragments are available.

Recently, Martalo et. al in [26], provide an interesting analysis for a distributed storage architecture based on a DHT-based overlay like Wuala project. They propose a proactive sporadic recovery strategy, according to which a document is regenerated each time a client or a peer downloads it in the system. The authors aimed to propose a replacement of the Wuala-based system , where resource maintenance is periodic and centralized. However, regenerating documents and thus redundant information within the proposed scheme, e.g. each time clients download documents, is not fair for both the popular and non-popular documents. For the popular documents, the proposed scheme is expensive in terms of the storage overhead but guarantee high availability as Wula-like system. Where for the non-popular documents, the scheme may lead to loss the data temporarily or permanently. Moreover, we can expect the performance for the private documents where only few users can access them. However this scheme can be good for the bandwidth utilization.

Characterizing peers availability both in local and wide area environments has been the focus of [28]. In this paper, Nurmi, Brevik and Wolski investigate three sets of data, each measuring machine availability in a different setting, and perform goodness-of-fit tests on each data set to assess which out of four distributions best fits the data. They have found that a hyper-exponential model fits more accurately the machine availability durations than the exponential, Pareto, or Weibull distribution.

To understand how the recovery process could be better modeled, we performed a packet-level simulation analysis of the download and the recovery processes in erasure-coded systems; cf. [12]. We found that the download time of a fragment of data located on a single peer follows approximately an exponential distribution. We also found that the recovery time essentially follows a hypo-exponential distribution with many distinct phases.

In light of the conclusions of [28], namely, that machine availability is modeled with a hyper-exponential distribution, and building on the findings of [12] we will propose in this paper a more general and accurate models that are valid under different distributed environments. Thus, the models presented in Sections 5 and 6 can be seen as a generalization of those presented in [10] and has little in common with [1]. Although [1, 10] are not valid under different systems and can not give a very accurate results, they are very simple and give a good background to easiely understand this paper.

3 System Description and Assumptions

In the following, we will distinguish the *peers*, which are computers where data is stored and which form a storage system, from the *users* whose objective is to retrieve the data stored in the storage system.

We consider a distributed *storage* system which peers randomly join and leave. The following assumptions on the P2PSS design will be enforced throughout the report:

- A block of data D is partitioned into s equally sized fragments to which, using erasure codes or regenerating code, r redundant fragments are added. The case of replication-based redundancy is equally captured by this notation, after setting $s = 1$ and letting the r redundant fragments be simple replicas of the unique fragment of the block. This notation—and hence our modeling—is general enough to study both replication-based and erasure code-based storage systems.
- Mainly for privacy issues, a peer can store at most one fragment of any data D.
- We assume the system has perfect knowledge of the location of fragments at any given time, e.g. by using a Distributed Hash Table (DHT) or a central authority.

- The system keeps track of only the *latest* known location of each fragment.
- Over time, a peer can be either *connected* to or *disconnected* from the storage system. At reconnection, a peer may or may not still store its fragments. We denote by p the probability that a peer that reconnects still stores its fragments.
- The number of connected peers at any time is typically much larger than the number of fragments associated with D, i.e., $s + r$. Therefore, we assume that there are always at least $s + r$ connected peers—hereafter referred to as *new* peers—which are ready to receive and store fragments of D.

We refer to as *on-time* (resp. *off-time*) a time-interval during which a peer is always connected (resp. disconnected). During a peer's off-time, the fragments stored on this peer are momentarily unavailable to the users of the storage system. At reconnection, and according to the assumptions above, the fragments stored on this peer will be available only with a *persistence* probability p (and with probability $1 - p$ they are lost). In order to improve data availability and increase the reliability of the storage system, it is therefore crucial to recover from losses by continuously monitoring the system and adding redundancy whenever needed.

We will investigate the performance of the two different repair policies: the *eager* and the *lazy* repair policies. Recall that in the lazy policy, the repair is delayed until the number of unavailable fragments reaches a given threshold, denoted k. We must have $k \leq r$ since D is lost if more than r fragments are missing from the storage system. Both repair policies can be represented by the threshold parameter $k \in \{1, 2, \ldots, r\}$, where k can take any value in the set $\{2, \ldots, r\}$ in the lazy policy and $k = 1$ in the eager policy.

Any repair policy can be implemented either in a centralized or a distributed way. In the following description, we assume that the system misses k fragments so that lost fragments have to be restored.

In the centralized implementation, a central authority will: (1) download *in parallel* s fragments from the peers which are connected, (2) reconstruct at once all the unavailable fragments, and (3) upload the reconstructed fragments *in parallel* onto as many new peers for storage. The central authority updates the database recording fragments locations as soon as all uploads terminate. Step 2 executes in a negligible time compared to the execution time of Steps 1 and 3 and will henceforth be ignored in the modeling.

In the distributed implementation, a secure agent on one new peer is notified of the identity of *one* out of the k unavailable fragments for it to reconstruct it. Upon notification, the secure agent (1) downloads s fragments (or s parts of s fragments if RC is used) of D from the peers which are connected to the storage system, (2) reconstructs the specified fragment and stores it on the peer's disk; (3) subsequently discards the s downloaded fragments, in the case of erasure code, so as to meet the privacy constraint that only one fragment of a block of data may be held by a peer. This operation iterates until less than k fragments are sensed unavailable and stops if the number of missing fragments reaches $k - 1$. The recovery of one fragment lasts mainly for the execution time of Step 1. We will thus consider the recovery process to end when the download of the last fragment (out of s) is completed.

In both implementations, once a fragment is reconstructed, any other copy of it that "reappears" in the system due to a peer reconnection is simply ignored, as only one location (the newest) of the fragment is recorded in the system. Similarly, if a fragment is unavailable, the system knows of only one disconnected peer that stores the unavailable fragment.

Given the system description, data D can be either *available*, *unavailable* or *lost*. Data D is said to be available if any s fragments out of the $s + r$ fragments can be downloaded by the users of the P2PSS. Data D is said to be unavailable if *less than* s fragments are available for download, however the missing fragments to complete D are located at a peer or a central authority on which a recovery process is ongoing. Data D is said to be lost if there are *less than* s fragments in the system including the fragments

involved in a recovery process. We assume that, at time $t = 0$, at least s fragments are available so that the document is initially available.

We now introduce the assumptions considered in our models.

Assumption 1: (off-times) We assume that successive durations of off-times of a peer are independent and identically distributed (iid) random variables (rvs) with a common exponential distribution function with parameter $\lambda > 0$.

Assumption 1 is in agreement with the analysis in [30].

Assumption 2: (on-times) We assume that successive durations of on-times of a peer are iid rvs with a common hyper-exponential distribution function with n phases; the parameters of phase i are $\{p_i, \mu_i\}$, with p_i the probability that phase i is selected and $1/\mu_i$ the mean duration of phase i. We naturally have $\sum_{i=1}^{n} p_i = 1$.

Assumption 2, with $n > 1$, is in agreement with the analysis in [28]; when $n = 1$, it is in agreement with the analysis in [30].

Assumption 3: (independence) Successive on-times and off-times are assumed to be independent. Peers are assumed to behave independently of each other.

Assumption 4: (download/upload durations) We assume that successive download (resp. upload) durations of a fragment are iid rvs with a common exponential distribution function with parameter α (resp. β). We further assume that concurrent fragments downloads/uploads are not correlated.

Assumption 4 is supported by our findings in [12]. As already mentioned, the fragment download/upload time was found to follow approximately an exponential distribution. As for the concurrent downloads/uploads, we have found in simulations that these are weakly correlated and close to be "independent" as long as the total workload is equally distributed over the active peers. There are two main reasons for the weak correlation between concurrent downloads/uploads as observed in simulations: (i) the good connectivity of nowadays core networks and (ii) the asymmetry in peers upstream and downstream bandwidths, as on average, a peer tends to have higher downstream than upstream bandwidth [33, 19]. So, as the bottleneck would be the upstream capacity of peers, the fragment download times are close to be iid rvs.

A consequence of Assumption 4 is that each of the block download time and the durations of the centralized and the distributed recovery processes is a rv following a hypo-exponential distribution [20]. Indeed, each of these durations is the summation of *independently* distributed exponential rvs (s for the block download and in the distributed scheme, and $s + k$ in the centralized scheme if k fragments are to be reconstructed) having each its own rate. This is a fundamental difference with [1] where the recovery process is assumed to follow an exponential distribution.

It is worth mentioning that the simulation analysis of [12] has concluded that in most cases the recovery time follows roughly a hypo-exponential distribution. This result is expected as long as fragments downloads/uploads are exponentially distributed and very weakly correlated.

Given Assumptions 1–4, the models developed in this report are more general and/or more realistic than those in [30, 1, 10]. Table 1 recapitulates the parameters introduced in this section. We will refer to s, r and k as the *protocol* parameters, p, λ and $\{p_i, \mu_i\}_{i=1,\ldots,n}$ as the *peers* parameters, and α and β as the *network* parameters.

Table 1: System parameters.

D	Block of data
s	Original number of fragments for each block of data
r	Number of redundant fragments
k	Threshold of the recovery process
p	Persistence probability
λ	Rate at which peers rejoin the system
$\{p_i, \mu_i\}_{i=1,\ldots,n}$	Parameters of the peers failure process
α	Download rate of a piece of data (fragment)
β	Upload rate of a fragment in the centralized-repair scheme

4 Preliminaries and Notation

We will focus in this section on the dynamic of peers in the storage system. In particular, we are interested in computing the stationary distribution of peers. According to Assumptions 1–3 in the previous section, each time a peer rejoins the system, it picks its on-time duration from an exponential distribution having parameter μ_i with probability p_i, for $i \in [1..n]$. In other words, a peer can stay connected for a short time in a session and for a long time in another one.

This dynamicity can be modeled as a general queueing network with an arbitrary but finite number n of different classes of customers (peers) and an infinite number of servers. In this network, a new customer enters directly, with probability p_i, a server with a service rate μ_i. Define $PI(\vec{n} = (n_1, \ldots, n_n)) := \lim_{t\to\infty} P(N_1(t) = n_1, \ldots, N_n(t) = n_n)$ to be the joint distribution function of the number of customers of class $1, \ldots, n$ in steady-state (or, equivalently, the number of busy servers) where $N_i(t)$ is the number of peers of class i in the system at time t for $i = 1, \ldots, n$. We have the following known results [2, 23]:

$$PI(\vec{n}) = 1/G \prod_{i=1}^{n} \frac{\rho_i^{n_i}}{n_i!}$$

where $\rho_i = \lambda p_i/\mu_i$ is the rate at which work enters class i and G is the normalizing constant.

$$G = \sum_{\vec{n}\in \mathrm{N}^n} \prod_{i=1}^{n} \frac{\rho_i^{n_i}}{n_i!}$$

Denote the expected number of customers of class i in the system by $E[n_i]$ where

$$E[n_i] = \sum_{\vec{n}\in \mathrm{N}^n} n_i PI(\vec{n}) = \rho_i \prod_{l=1}^{n} e^{\rho_l}, \qquad \text{for } i = 1, \ldots, n$$

.

For later use, we will compute the probability of selecting a *new* peer in phase i, denoted by $R(i)$, or equivalently the percentage of the connected peers in phase i as follows:

$$R(i) = \frac{E[n_i]}{\sum_{l=1}^{n} E[n_l]} = \frac{\rho_i}{\sum_{l=1}^{n} \rho_l} = \frac{p_i/\mu_i}{\sum_{l=1}^{n} p_l/\mu_l} \tag{1}$$

We introduce as well functions S and f such that for a given n-tuple $\vec{a} = (a_1, \ldots, a_n)$, $S(\vec{a}) := \sum_{i=1}^{n} a_i$ and $f_i(\vec{a}) := a_i/S(\vec{a})$.

We conclude this section by a word on the notation: a subscript "c" (resp. "d") will indicate that we are considering the centralized (resp. distributed) recovery scheme. The notation $\vec{e}_j^{\,i}$ refers to a *row* vector of dimension j whose entries are null except the i-th entry that is equal to 1; the notation $\vec{1}_j$ refers

to a *column* vector of dimension j whose each entry is equal to 1; and the notation $\vec{0}$ refers to a null *row* vector of appropriate dimension. $\mathbb{1}\{A\}$ is the characteristic function of event A. The notation $[a]^+$ refers to $\max\{a, 0\}$. The set of integers ranging from a to b is denoted $[a..b]$. Given a set of n rvs $\{B_i(t)\}_{i\in[1..n]}$, $\vec{B}(t)$ denotes the vector $(B_1(t), \ldots, B_n(t))$ and $\vec{\mathbf{B}}$ denotes the stochastic process $\{\vec{B}(t),\, t \geq 0\}$.

5 Centralized Repair Systems

In this section, we address the performance of P2PSS using the centralized-recovery scheme.

We will focus on a single block of data D, and pay attention only to peers storing fragments of this block. At any time t, the state of a block D can be described by both the number of fragments that are available for download and the state of the recovery process. When triggered, the recovery process goes first through a "download phase" (fragments are downloaded from connected peers to the central authority) then through an "upload phase" (fragments are uploaded to new peers from the central authority).

More formally, we introduce n-dimensional vectors $\vec{X}_c(t)$, $\vec{Y}_c(t)$, $\vec{Z}_c(t)$, $\vec{U}_c(t)$, and $\vec{V}_c(t)$, where n is the number of phases of the hyper-exponential distribution of peers on-times durations, and a $5n$-dimensional vector $\vec{W}_c(t) = (\vec{X}_c(t), \vec{Y}_c(t), \vec{Z}_c(t), \vec{U}_c(t), \vec{V}_c(t))$. Vectors $\vec{Y}_c(t)$ and $\vec{Z}_c(t)$ describe the download phase of the recovery process whereas $\vec{U}_c(t)$ and $\vec{V}_c(t)$ describe its upload phase. The formal definition of these vectors is as follows:

- $\vec{X}_c(t) := (X_{c,1}(t), \ldots, X_{c,n}(t))$ where $X_{c,l}(t)$ is a $[0..s+r]$-valued rv denoting the number of fragments of D stored on peers that are in phase l at time t.
- $\vec{Y}_c(t) := (Y_{c,1}(t), \ldots, Y_{c,n}(t))$ where $Y_{c,l}(t)$ is a $[0..s-1]$-valued rv denoting the number of fragments of D being downloaded at time t to the central authority from peers in phase l (one fragment per peer).
- $\vec{Z}_c(t) := (Z_{c,1}(t), \ldots, Z_{c,n}(t))$ where $Z_{c,l}(t)$ is a $[0..s]$-valued rv denoting the number of fragments of D hold at time t by the central authority and whose download was done from peers in phase l (one fragment per peer). Observe that these peers may have left the system by time t.
- $\vec{U}_c(t) := (U_{c,1}(t), \ldots, U_{c,n}(t))$ where $U_{c,l}(t)$ is a $[0..s+r-1]$-valued rv denoting the number of (reconstructed) fragments of D being uploaded at time t from the central authority to new peers that are in phase l (one fragment per peer).
- $\vec{V}_c(t) := (V_{c,1}(t), \ldots, V_{c,n}(t))$ where $V_{c,l}(t)$ is a $[0..s+r-1]$-valued rv denoting the number of (reconstructed) fragments of D whose upload from the central authority to new peers that are in phase l has been completed at time t (one fragment per peer).

Given the above definitions, we necessarily have $Y_{c,l}(t) \leq X_{c,l}(t)$ for $l \in [1..n]$ at any time t. The number of fragments of D that are available for download at time t is given by $S(\vec{X}_c(t))$ (recall the definition of the function S in Section 3). Given that s fragments of D need to be downloaded to the central authority during the download phase of the recovery process, we will have (during this phase) $S(\vec{Y}_c(t)) + S(\vec{Z}_c(t)) = s$, such that $S(\vec{Y}_c(t)), S(\vec{Z}_c(t)) \in [1..s-1]$. Once the download phase is completed, the central authority will reconstruct at once all missing fragments, that is $s + r - S(\vec{X}_c(t))$. Therefore, during the upload phase, we have $S(\vec{U}_c(t)) + S(\vec{V}_c(t)) = s + r - S(\vec{X}_c(t))$. Observe that, once the download phase is completed, the number of available fragments, $S(\vec{X}_c(t))$, may well decrease to 0 with peers all leaving the system. In such a situation, the central authority will reconstruct $s + r$ fragments of D. As soon as the download phase is completed $\vec{Y}_c(t) = \vec{0}$ and $S(\vec{Z}_c(t)) = s$. The end of the upload phase is also

the end of the recovery process. We will then have $\vec{Y}_c(t) = \vec{Z}_c(t) = \vec{U}_c(t) = \vec{V}_c(t) = \vec{0}$ until the recovery process is again triggered.

According to the terminology introduced in Section 3, at time t, data D is *available* if $S(\vec{X}_c(t)) \geq s$, regardless of the state of the recovery process. It is *unavailable* if $S(\vec{X}_c(t)) < s$ but $S(\vec{Z}_c(t))$—the number of fragments hold by the central authority—is larger than $s - S(\vec{X}_c(t))$ *and* at least $s - S(\vec{X}_c(t))$ fragments out of $S(\vec{Z}_c(t))$ are different from those $S(\vec{X}_c(t))$ fragments available on peers. Otherwise, D is considered to be *lost*. The latter situation will be modeled by a single state a.

If a recovery process is ongoing, the exact number of *distinct* fragments of D that are in the system—counting both those that are available and those hold by the central authority—may be unknown due to peers churn. However, we are able to find a lower bound on it, namely,

$$b(\vec{X}_c(t), \vec{Y}_c(t), \vec{Z}_c(t)) := \sum_{l=1}^{n} \max\{X_{c,l}(t), Y_{c,l}(t) + Z_{c,l}(t)\}.$$

In fact, the uncertainty about the number of distinct fragments is a result of peers churn. That said, this bound is very tight and most often gives the exact number of distinct fragments since peers churn occurs at a much larger time-scale than a fragment download. In our modeling, we consider an unavailable data D to become lost when the bound b takes a value smaller than s. Observe that, if the recovery process is not triggered, then $b(\vec{X}_c(t), \vec{0}, \vec{0}) = S(\vec{X}_c(t))$ gives the exact number of distinct fragments.

The system state at time t can be represented by the $5n$-dimensional vector $\vec{W}_c(t)$. Thanks to the assumptions made in Section 3, the multi-dimensional process $\vec{\mathbf{W}}_c := \{\vec{W}_c(t), t \geq 0\}$ is an absorbing homogeneous continuous-time Markov chain (CTMC) with a set of transient states $\mathcal{T}_c$ representing the situations when D is either available or unavailable and a single absorbing state a representing the situation when D is lost. As writing $\mathcal{T}_c$ is tedious, we will simply say that $\mathcal{T}_c$ is a subset of $[0..s+r]^n \times [0..s-1]^n \times [0..s]^n \times [0..s+r-1]^n \times [0..s+r-1]^n$. The elements of $\mathcal{T}_c$ must verify the constraints mentioned above.

Without loss of generality, we assume that $S(\vec{X}_c(0)) \geq s$. The infinitesimal generator has the following canonical form

$$\begin{array}{c} \\ \mathcal{T}_c \\ a \end{array} \begin{array}{c} \begin{array}{cc} \mathcal{T}_c & a \end{array} \\ \left(\begin{array}{c|c} \vec{Q}_c & \vec{R}_c \\ \hline \vec{0} & 0 \end{array}\right) \end{array}$$

where $\vec{R}_c$ is a non-zero column vector of size $|\mathcal{T}_c|$, and $\vec{Q}_c$ is $|\mathcal{T}_c|$-by-$|\mathcal{T}_c|$ matrix. The elements of $\vec{R}_c$ are the transition rates between the transient states $\vec{w}_c \in \mathcal{T}_c$ and the absorbing state a. The diagonal elements of $\vec{Q}_c$ are each the total transition rate out of the corresponding transient state. The other elements of $\vec{Q}_c$ are the transition rates between each pair of transient states. The non-zero elements of $\vec{R}_c$ are, for $S(\vec{y}_c) \in [1..S(\vec{x}_c)]$ and $S(\vec{z}_c) = s - S(\vec{y}_c)$,

$$r_c(\vec{x}_c, \vec{0}, \vec{0}, \vec{0}, \vec{0}) = \sum_{l=1}^{n} x_{c,l}\mu_l, \qquad \text{for} \quad S(\vec{x}_c) = s.$$

$$r_c(\vec{x}_c, \vec{y}_c, \vec{z}_c, \vec{0}, \vec{0}) = \sum_{l=1}^{n} y_{c,l}\mu_l \cdot \mathbb{1}\{b(\vec{x}_c, \vec{y}_c, \vec{z}_c) = s\}, \qquad \text{for} \quad S(\vec{x}_c) \in [1..s].$$

Let us proceed to the definition of the non-zero elements of $\vec{Q}_c$.

The case when a peer leaves the system There are seven different situations in this case. In the first situation, either the recovery process has not been triggered or it has but no download has been completed yet. In both the second and third situations, the download phase of the recovery process is ongoing and at least one download is completed. However, in the second situation, the departing peer does not affect the recovery process (either it was not involved in it or its fragment download is

completed), unlike what happens in the third situation. In the third situation, a fragment download is interrupted due to the peer's departure. The central authority will then immediately start downloading a fragment from another available peer that is uniformly selected among all available peers not currently involved in the recovery process. The fourth situation arises when a peer leaves the system at the end of the download phase. The fifth situation occurs when an available fragment becomes unavailable during the upload phase. The sixth situation occurs when a peer, to which the central authority is uploading a fragment, leaves the system. The last situation arises because of a departure of a peer to which the central authority has completely uploaded a reconstructed fragment. Note that the uploaded fragment was not yet integrated in the available fragments. This is caused by the fact that the central authority updates the database recording fragments locations as soon as all uploads terminate. To overcom any departure or failure that occurs in the context of one of the last three situations, the central authority has then to upload again the given fragment to a new peer. A new selected peer would be in phase m with probability $R(m)$ for $m \in [1..n]$. The elements of $\vec{Q}_c$ corresponding to these seven situations are, for $l \in [1..n]$ and $m \in [1..n]$,

$$
\begin{aligned}
&q_c((\vec{x}_c, \vec{0}, \vec{0}, \vec{0}, \vec{0}), (\vec{x}_c - \vec{e}_n^{\,l}, \vec{0}, \vec{0}, \vec{0}, \vec{0})) = x_{c,l}\mu_l, \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [s+1..s+r]. \\
&q_c((\vec{x}_c, \vec{y}_c, \vec{z}_c, \vec{0}, \vec{0}), (\vec{x}_c - \vec{e}_n^{\,l}, \vec{y}_c, \vec{z}_c, \vec{0}, \vec{0})) = [x_{c,l} - y_{c,l}]^+\mu_l, \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [s..s+r-1], \quad S(\vec{y}_c) \in [1..s-1], \quad S(\vec{z}_c) = s - S(\vec{y}_c); \\
&\qquad \text{or} \quad S(\vec{x}_c) \in [2..s-1], \quad S(\vec{y}_c) \in [1..S(\vec{x}_c)-1], \quad S(\vec{z}_c) = s - S(\vec{y}_c). \\
&q_c((\vec{x}_c, \vec{y}_c, \vec{z}_c, \vec{0}, \vec{0}), (\vec{x}_c - \vec{e}_n^{\,l}, \vec{y}_c - \vec{e}_n^{\,l} + \vec{e}_n^{\,m}, \vec{z}_c, \vec{0}, \vec{0})) = \frac{y_{c,l}\mu_l[x_{c,m} - y_{c,m} - z_{c,m}]^+}{\sum_{i=1}^{n}[x_{c,i} - y_{c,i} - z_{c,i}]^+}, \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [s..s+r-1], \quad S(\vec{y}_c) \in [1..s-1], \quad S(\vec{z}_c) = s - S(\vec{y}_c); \\
&\qquad \text{or} \quad S(\vec{x}_c) \in [2..s-1], \quad S(\vec{y}_c) \in [1..S(\vec{x}_c)-1], \quad S(\vec{z}_c) = s - S(\vec{y}_c).
\end{aligned}
$$

$$
\begin{aligned}
&q_c((\vec{x}_c, \vec{0}, \vec{z}_c, \vec{0}, \vec{0}), (\vec{x}_c - \vec{e}_n^{\,l}, \vec{0}, \vec{z}_c, \vec{0}, \vec{0})) = x_{c,l}\mu_l, \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [1..s+r-1], \quad S(\vec{z}_c) = s. \\
&q_c((\vec{x}_c, \vec{0}, \vec{z}_c, \vec{u}_c, \vec{v}_c), (\vec{x}_c - \vec{e}_n^{\,l}, \vec{0}, \vec{z}_c, \vec{u}_c + \vec{e}_n^{\,m}, \vec{v}_c)) = x_{c,l}\mu_l R(m), \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [1..s+r-2], \quad S(\vec{z}_c) = s, \quad S(\vec{u}_c) \in [1..s+r-S(\vec{x}_c)-1], \\
&\qquad\qquad S(\vec{v}_c) = s + r - S(\vec{x}_c) - S(\vec{u}_c). \\
&q_c((\vec{x}_c, \vec{0}, \vec{z}_c, \vec{u}_c, \vec{v}_c), (\vec{x}_c, \vec{0}, \vec{z}_c, \vec{u}_c - \vec{e}_n^{\,l} + \vec{e}_n^{\,m}, \vec{v}_c)) = u_{c,l}\mu_l R(m), \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [1..s+r-2], \quad S(\vec{z}_c) = s, \quad S(\vec{u}_c) \in [1..s+r-S(\vec{x}_c)-1], \\
&\qquad\qquad S(\vec{v}_c) = s + r - S(\vec{x}_c) - S(\vec{u}_c), \quad l \neq m. \\
&q_c((\vec{x}_c, \vec{0}, \vec{z}_c, \vec{u}_c, \vec{v}_c), (\vec{x}_c, \vec{0}, \vec{z}_c, \vec{u}_c + \vec{e}_n^{\,m}, \vec{v}_c - \vec{e}_n^{\,l})) = v_{c,l}\mu_l R(m), \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [1..s+r-2], \quad S(\vec{z}_c) = s, \quad S(\vec{u}_c) \in [1..s+r-S(\vec{x}_c)-1], \\
&\qquad\qquad S(\vec{v}_c) = s + r - S(\vec{x}_c) - S(\vec{u}_c).
\end{aligned}
$$

The case when a peer rejoins the system Recall that the system keeps trace of only the latest known location of each fragment. As such, once a fragment is reconstructed, any other copy of it that "reappears" in the system due to a peer reconnection is simply ignored, as only one location (the newest) of the fragment is recorded in the system. Similarly, if a fragment is unavailable, the system knows of only one disconnected peer that stores the unavailable fragment. In the following, only relevant reconnections are considered. For instance, when the recovery process is in its upload phase, any peer that rejoins the system does not affect the system state since all fragments have been reconstructed and are being uploaded to their new locations.

There are three situations where reconnections may be relevant. In the first, either the recovery process has not been triggered or it has but no download has been completed yet. In both the second and third situations, the download phase of the recovery process is ongoing and at least one download is

completed. However, in the third situation, there is only one missing fragment, so when the peer storing the missing fragments rejoins the system, the recovery process aborts.

The elements of $\vec{Q}_c$ corresponding to these three situations are, for $l \in [1..n]$ and $S(\vec{z}_c) = s - S(\vec{y}_c)$

$$\begin{aligned}
&q_c((\vec{x}_c, \vec{0}, \vec{0}, \vec{0}, \vec{0}), (\vec{x}_c + \vec{e}_n^{\,l}, \vec{0}, \vec{0}, \vec{0}, \vec{0})) = p_l(s + r - S(\vec{x}_c))p\lambda, \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [s..s + r - 1]. \\
&q_c((\vec{x}_c, \vec{y}_c, \vec{z}_c, \vec{0}, \vec{0}), (\vec{x}_c + \vec{e}_n^{\,l}, \vec{y}_c, \vec{z}_c, \vec{0}, \vec{0})) = p_l(s + r - S(\vec{x}_c))p\lambda, \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [s..s + r - 2], \quad S(\vec{y}_c) \in [1..s - 1]; \\
&\qquad \text{or} \quad S(\vec{x}_c) \in [1..s - 1], \qquad S(\vec{y}_c) \in [1..S(\vec{x}_c)]. \\
&q_c((\vec{x}_c, \vec{y}_c, \vec{z}_c, \vec{0}, \vec{0}), (\vec{x}_c + \vec{e}_n^{\,l}, \vec{0}, \vec{0}, \vec{0}, \vec{0})) = p_l \; p\lambda, \\
&\qquad \text{for} \quad S(\vec{x}_c) = s + r - 1, \qquad S(\vec{y}_c) \in [1..s - 1].
\end{aligned}$$

The case when one download is completed during the recovery process When a recovery process is initiated, the system state verifies $S(\vec{x}_c) \in [s..s+r-k]$ and $\vec{y}_c = \vec{z}_c = \vec{u}_c = \vec{v}_c = \vec{0}$. The central authority selects s peers out of the $S(\vec{x}_c)$ peers that are connected to the system and initiates a fragment download from each. Among the s peers that are selected, i_l out of s would be in phase l, for $l \in [1..n]$. Let $\vec{i} = (i_1, \ldots, i_n)$. We naturally have $0 \leq i_l \leq x_{c,l}$, for $l \in [1..n]$, and $S(\vec{i}) = s$. This selection occurs with probability

$$g(\vec{i}, \vec{x}_c) := \frac{\prod_{l=1}^{n} \binom{x_{c,l}}{i_l}}{\binom{S(\vec{x}_c)}{s}}.$$

The probability that the first download to be completed out of s was from a peer in phase l is equal to $f_l(\vec{i}) = i_l/s$ (recall the definition of f in Section 3). Similarly, when the number of ongoing downloads is $\vec{y}_c$, the probability that the first download to be completed out of $S(\vec{y}_c)$ was from a peer in phase l is equal to $f_l(\vec{y}_c) = y_{c,l}/S(\vec{y}_c)$.

The two possible transition rates in such situations are, for $l \in [1..n]$, $m \in [1..n]$ and $S(\vec{z}_c) = s - S(\vec{y}_c)$,

$$\begin{aligned}
&q_c((\vec{x}_c, \vec{0}, \vec{0}, \vec{0}, \vec{0}), (\vec{x}_c, \vec{i} - \vec{e}_n^{\,l}, \vec{e}_n^{\,l}, \vec{0}, \vec{0})) = s\alpha \; g(\vec{i}, \vec{x}_c) \; f_l(\vec{i}), \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [s..s + r - k], \quad i_m \in [0..x_{c,m}], \quad S(\vec{i}) = s. \\
&q_c((\vec{x}_c, \vec{y}_c, \vec{z}_c, \vec{0}, \vec{0}), (\vec{x}_c, \vec{y}_c - \vec{e}_n^{\,l}, \vec{z}_c + \vec{e}_n^{\,l}, \vec{0}, \vec{0})) = S(\vec{y}_c)\alpha \; f_l(\vec{y}_c), \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [s..s + r - 1], \quad S(\vec{y}_c) \in [1..s - 1]; \\
&\qquad \text{or} \quad S(\vec{x}_c) \in [1..s - 1], \qquad S(\vec{y}_c) \in [1..S(\vec{x}_c)].
\end{aligned}$$

The case when one upload is completed during the recovery process When the download phase is completed, the system state verifies $S(\vec{z}_c) = s$ and $\vec{y}_c = \vec{u}_c = \vec{v}_c = \vec{0}$. The central authority selects $s + r - S(\vec{x}_c)$ *new* peers that are connected to the system and initiates a (reconstructed) fragment upload to each. Among the peers that are selected, i_l out of $s+r-S(\vec{x}_c)$ would be in phase l, for $l \in [1..n]$. Let $\vec{i} = (i_1, \ldots, i_n)$. We naturally have $0 \leq i_l \leq s + r - S(\vec{x}_c)$, for $l \in [1..n]$, and $S(\vec{i}) = s + r - S(\vec{x}_c)$. This selection occurs with probability

$$h(\vec{i}, \vec{x}_c) := \binom{s + r - S(\vec{x}_c)}{i_1, i_2, \ldots, i_n} \prod_{l=1}^{n} R(l)^{i_l}$$

where the multinomial coefficient has been used. For $l \in [1..n]$ and $S(\vec{z}_c) = s$, we can write

$$
\begin{aligned}
&q_c((\vec{x}_c, \vec{0}, \vec{z}_c, \vec{0}, \vec{0}), (\vec{x}_c, \vec{0}, \vec{z}_c, \vec{i} - \vec{e}_n^{\,l}, \vec{e}_n^{\,l})) = S(\vec{i})\beta \; h(\vec{i}, \vec{x}_c) \; f_l(\vec{i}), \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [0..s+r-2], \quad \vec{i} \in [0..s+r-S(\vec{x}_c)]^n, \quad S(\vec{i}) = s+r-S(\vec{x}_c). \\
&q_c((\vec{x}_c, \vec{0}, \vec{z}_c, \vec{u}_c, \vec{v}_c), (\vec{x}_c, \vec{0}, \vec{z}_c, \vec{u}_c - \vec{e}_n^{\,l}, \vec{v}_c + \vec{e}_n^{\,l})) = S(\vec{u}_c)\beta \; f_l(\vec{u}_c), \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [0..s+r-2], \quad S(\vec{u}_c) \in [2..s+r-S(\vec{x}_c)-1], \\
&\qquad\qquad S(\vec{v}_c) = s+r-S(\vec{x}_c)-S(\vec{u}_c). \\
&q_c((\vec{x}_c, \vec{0}, \vec{z}_c, \vec{e}_n^{\,l}, \vec{v}_c), (\vec{x}_c + \vec{v}_c + \vec{e}_n^{\,l}, \vec{0}, \vec{0}, \vec{0}, \vec{0})) = \beta, \\
&\qquad \text{for} \quad S(\vec{x}_c) \in [0..s+r-2], \quad S(\vec{v}_c) = s+r-S(\vec{x}_c)-1. \\
&q_c((\vec{x}_c, \vec{0}, \vec{z}_c, \vec{0}, \vec{0}), (\vec{x}_c + \vec{e}_n^{\,l}, \vec{0}, \vec{0}, \vec{0}, \vec{0})) = R(l)\beta, \\
&\qquad \text{for} \quad S(\vec{x}_c) = s+r-1.
\end{aligned}
$$

Note that $\vec{Q}_c$ is not an infinitesimal generator since elements in some rows do not sum up to 0. Those rows correspond to the system states where only s distinct fragments are present in the system. The diagonal elements of $\vec{Q}_c$ are

$$
q_c(\vec{w}_c, \vec{w}_c) = -r_c(\vec{w}_c) - \sum_{\vec{w}'_c \in \mathcal{T}_c - \{\vec{w}_c\}} q_c(\vec{w}_c, \vec{w}'_c), \quad \text{for} \quad \vec{w}_c \in \mathcal{T}_c.
$$

For illustration purposes, we depict in Fig. 1 some of the transitions of the absorbing CTMC when $n = 2$, $s = 3$, $r = 1$, and $k = 1$.

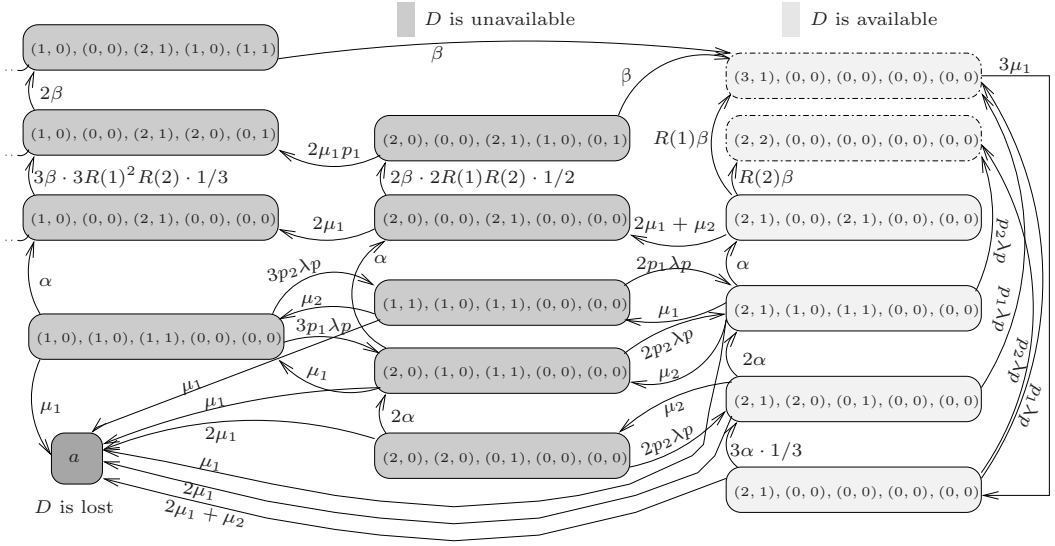


Figure 1: Some transitions of the Markov chain $\vec{\mathbf{W}}_c$ when $n = 2$, $s = 3$, $r = 1$, and $k = 1$.

5.1 Data Lifetime

This section is devoted to the analysis of the lifetime of D. It will be convenient to introduce sets

$$
\mathcal{E}_I := \{(\vec{x}_c, \vec{0}, \vec{0}, \vec{0}, \vec{0}) : \vec{x}_c \in [0..s+r]^n, S(\vec{x}_c) = I\} \text{ for } I \in [s..s+r].
$$

The set $\mathcal{E}_I$ consists of all states of the process $\vec{\mathbf{W}}_c$ in which the number of fragments of D currently available is equal to I and the recovery process either has not been triggered (for $I \in [s+r-k+1..s+r]$) or it has but no download has been completed yet (for $I \in [s..s+r-k]$). For any I, the cardinal of $\mathcal{E}_I$

is $\binom{I+n-1}{n-1}$ (think of the possible selections of $n-1$ boxes in a row of $I+n-1$ boxes, so as to delimit n groups of boxes summing up to I).

Introduce $T_c(\mathcal{E}_I) := \inf\{t > 0 : \vec{W}_c(t) = a | \vec{W}_c(0) \in \mathcal{E}_I\}$, the time until absorption in state a—or equivalently the time until D is lost—given that the initial number of fragments of D available in the system is equal to I. In the following, $T_c(\mathcal{E}_I)$ will be referred to as the *conditional block lifetime*. We are interested in the conditional probability distribution function, $P(T_c(\mathcal{E}_I) \leq t)$, and the conditional expectation, $\mathrm{E}[T_c(\mathcal{E}_I)]$, given that $\vec{W}_c(0) \in \mathcal{E}_I$ for $I \in [s..s+r]$.

From the theory of absorbing Markov chains, we can compute $P(T_c(\{\vec{w}_c\}) \leq t)$ where $T_c^h(\{\vec{w}_c\})$ is the time until absorption in state a given that the system initiates in state $\vec{w}_c \in \mathcal{T}_c$. We know that (e.g. [27, Lemma 2.2])

$$P(T_c(\{\vec{w}_c\}) \leq t) = 1 - \vec{e}_{|\mathcal{T}_c|}^{\,\mathrm{ind}(\vec{w}_c)} \cdot \exp\left(t\vec{Q}_c\right) \cdot \vec{1}_{|\mathcal{T}_c|}, \quad t > 0,\ \vec{w}_c \in \mathcal{T}_c \tag{2}$$

where $\mathrm{ind}(\vec{w}_c)$ refers to the index of state $\vec{w}_c$ in the matrix $\vec{Q}_c$. Definitions of vectors $\vec{e}_j^{\,i}$ and $\vec{1}_j$ were given at the end of Section 3. Observe that the term $\vec{e}_{|\mathcal{T}_c|}^{\,\mathrm{ind}(\vec{w}_c)} \cdot \exp\left(t\vec{Q}_c\right) \cdot \vec{1}_{|\mathcal{T}_c|}$ in the right-hand side of (2) is nothing but the summation of all $|\mathcal{T}_c|$ elements in row $\mathrm{ind}(\vec{w}_c)$ of matrix $\exp\left(t\vec{Q}_c\right)$.

Let $\pi_{\vec{x}_c}$ denote the probability that the system starts in state $\vec{w}_c = (\vec{x}_c, \vec{0}, \vec{0}, \vec{0}, \vec{0}) \in \mathcal{E}_I$ at time 0 given that $\vec{W}_c(0) \in \mathcal{E}_I$. We can write

$$\pi_{\vec{x}_c} := P\left(\vec{W}_c(0) = \vec{w}_c \in \mathcal{E}_I | \vec{W}_c(0) \in \mathcal{E}_I\right) = \binom{I}{x_{c,1}, \ldots, x_{c,n}} \prod_{l=1}^{n} R(l)^{x_{c,l}}. \tag{3}$$

Clearly $\sum_{\vec{w}_c \in \mathcal{E}_I} \pi_{\vec{x}_c} = 1$ for $I \in [s..s+r]$. Using (2) and (3) and the total probability theorem yields, for $I \in [s..s+r]$,

$$\begin{aligned} P(T_c(\mathcal{E}_I) \leq t) &= \sum_{\vec{w}_c \in \mathcal{E}_I} \pi_{\vec{x}_c}\ P\left(T_c(\{\vec{w}_c\}) \leq t\right) \\ &= 1 - \sum_{\vec{w}_c \in \mathcal{E}_I} \pi_{\vec{x}_c}\ \vec{e}_{|\mathcal{T}_c|}^{\,\mathrm{ind}(\vec{w}_c)} \cdot \exp\left(t\vec{Q}_c\right) \cdot \vec{1}_{|\mathcal{T}_c|}, \quad t > 0. \end{aligned} \tag{4}$$

We know from [27, p. 46] that the expected time until absorption given that the $\vec{W}_c(0) = \vec{w}_c \in \mathcal{T}_c$ can be written as

$$\mathrm{E}\left[T_c(\{\vec{w}_c)\}\right] = -\vec{e}_{|\mathcal{T}_c|}^{\,\mathrm{ind}(\vec{w}_c)} \cdot \left(\vec{Q}_c\right)^{-1} \cdot \vec{1}_{|\mathcal{T}_c|}, \qquad \vec{w}_c \in \mathcal{T}_c,$$

where the existence of $\left(\vec{Q}_c\right)^{-1}$ is a consequence of the fact that all states in $\mathcal{T}_c$ are transient [27, p. 45]. The conditional expectation of $T_c(\mathcal{E}_I)$ is then (recall that the elements of $\mathcal{E}_I$ are of the form $(\vec{x}_c, \vec{0}, \vec{0}, \vec{0}, \vec{0})$)

$$\begin{aligned} \mathrm{E}\left[T_c(\mathcal{E}_I)\right] &= \sum_{\vec{w}_c \in \mathcal{E}_I} \pi_{\vec{x}_c}\ \mathrm{E}\left[T_c(\{\vec{w}_c\})\right] \\ &= -\sum_{\vec{w}_c \in \mathcal{E}_I} \pi_{\vec{x}_c}\ \vec{e}_{|\mathcal{T}_c|}^{\,\mathrm{ind}(\vec{w}_c)} \cdot \left(\vec{Q}_c\right)^{-1} \cdot \vec{1}_{|\mathcal{T}_c|}, \quad \text{for } I \in [s..s+r]. \end{aligned} \tag{5}$$

5.2 Data Availability

In this section we introduce different metrics to quantify the availability of D. But first, we will study the time during which J fragments of D are available in the system given that there were initially I fragments. To formalize this measure, we introduce the following subsets of $\mathcal{T}_c$, for $J \in [0..s+r]$,

$$\mathcal{F}_J := \{(\vec{x}_c, \vec{y}_c, \vec{z}_c, \vec{u}_c, \vec{v}_c) \in \mathcal{T}_c : S(\vec{x}_c) = J\}$$

The set $\mathcal{F}_J$ consists of all states of process $\vec{\mathbf{W}}_c$ in which the number of fragments of D currently available is equal to J, regardless of the state of the recovery process. The subsets $\mathcal{F}_J$ form a partition of $\mathcal{T}_c$. We may define now

$$T_c(\mathcal{E}_I, \mathcal{F}_J) := \int_0^{T_c(\mathcal{E}_I)} \mathbb{1}\Big\{\vec{W}_c(t) \in \mathcal{F}_J | \vec{W}_c(0) \in \mathcal{E}_I\Big\}\, dt.$$

$T_c(\mathcal{E}_I, \mathcal{F}_J)$ is the total time spent by the CTMC in the set $\mathcal{F}_J$ before being absorbed in state a, given that $\vec{W}_c(0) \in \mathcal{E}_I$. Similarly, $T_c(\{\vec{w}_c\}, \{\vec{w}'_c\})$ is the total time spent by the CTMC in state $\vec{w}'_c$ before being absorbed in state a, given that $\vec{W}_c(0) = \vec{w}_c$. We know from [18, p. 419] that

$$\mathrm{E}\left[T_c(\{\vec{w}_c\}, \{\vec{w}'_c\})\right] = -\vec{e}^{\,\mathrm{ind}(\vec{w}_c)}_{|\mathcal{T}_c|} \cdot \left(\vec{Q}_c\right)^{-1} \cdot {}^t\vec{e}^{\,\mathrm{ind}(\vec{w}'_c)}_{|\mathcal{T}_c|}, \qquad \vec{w}_c, \vec{w}'_c \in \mathcal{T}_c \tag{6}$$

where ${}^t\vec{y}$ denotes the transpose of a given vector $\vec{y}$. In other words, the expectation $\mathrm{E}\left[T_c(\{\vec{w}_c\}, \{\vec{w}'_c\})\right]$ is the entry of matrix $\left(-\vec{Q}_c\right)^{-1}$ at row $\mathrm{ind}(\vec{w}_c)$ and column $\mathrm{ind}(\vec{w}'_c)$. Using (3) and (6), we derive for $I \in [s..s+r]$ and $J \in [0..s+r]$

$$\begin{aligned}
\mathrm{E}\left[T_c(\mathcal{E}_I, \mathcal{F}_J)\right] &= \sum_{\vec{w}'_c \in \mathcal{F}_J} \mathrm{E}\left[T_c(\mathcal{E}_I, \{\vec{w}'_c\})\right] \\
&= \sum_{\vec{w}_c \in \mathcal{E}_I} \sum_{\vec{w}'_c \in \mathcal{F}_J} \pi_{\vec{x}_c} \mathrm{E}\left[T_c(\{\vec{w}_c\}, \{\vec{w}'_c\})\right] \\
&= -\sum_{\vec{w}_c \in \mathcal{E}_I} \sum_{\vec{w}'_c \in \mathcal{F}_J} \pi_{\vec{x}_c} \vec{e}^{\,\mathrm{ind}(\vec{w}_c)}_{|\mathcal{T}_c|} \cdot \left(\vec{Q}_c\right)^{-1} \cdot {}^t\vec{e}^{\,\mathrm{ind}(\vec{w}'_c)}_{|\mathcal{T}_c|}.
\end{aligned} \tag{7}$$

We are now in position of introducing two availability metrics. The first metric, defined as

$$M_{c,1}(\mathcal{E}_I) := \mathrm{E}\left[\sum_{J=0}^{s+r} J\, \frac{T_c(\mathcal{E}_I, \mathcal{F}_J)}{T_c(\mathcal{E}_I)}\right], \quad \text{where } I \in [s..s+r],$$

can be interpreted as the expected number of fragments of D that are available for download—as long as D is not lost—given that I fragments are initially available. A second metric is

$$M_{c,2}(\mathcal{E}_I, m) := \mathrm{E}\left[\sum_{J=m}^{s+r} \frac{T_c(\mathcal{E}_I, \mathcal{F}_J)}{T_c(\mathcal{E}_I)}\right], \quad \text{where } I \in [s..s+r],$$

that we can interpret as the fraction of the lifetime of D when at least m fragments are available for download, given that I fragments are initially available. For instance, $M_{c,2}(\mathcal{E}_{s+r}, s)$ is the proportion of time when data D is *available* for users, given that $s+r$ fragments of D are initially available for download.

The expectations involved in the computation of the availability metrics are difficult to find in closed-form. Therefore, we resort to using the following approximation

$$\mathrm{E}\left[\frac{T_c(\mathcal{E}_I, \mathcal{F}_J)}{T_c(\mathcal{E}_I)}\right] \approx \frac{\mathrm{E}[T_c(\mathcal{E}_I, \mathcal{F}_J)]}{\mathrm{E}[T_c(\mathcal{E}_I)]}, \tag{8}$$

where the terms in the right-hand side have been derived in (7) and (5). The validation of this approximation is introduced in technical report [14]. With this approximation in mind, the two availability metrics become

$$M_{c,1}(\mathcal{E}_I) = \sum_{J=0}^{s+r} J\, \frac{\mathrm{E}[T_c(\mathcal{E}_I, \mathcal{F}_J)]}{\mathrm{E}[T_c(\mathcal{E}_I)]}, \quad \text{where } I \in [s..s+r], \tag{9}$$

$$M_{c,2}(\mathcal{E}_I, m) = \sum_{J=m}^{s+r} \frac{\mathrm{E}[T_c(\mathcal{E}_I, \mathcal{F}_J)]}{\mathrm{E}[T_c(\mathcal{E}_I)]}, \quad \text{where } I \in [s..s+r]. \tag{10}$$

6 Distributed Repair Systems

In this section, we model P2P storage systems that implement a distributed recovery mechanism, as described in Section 3. Unlike the centralized case, the distributed recovery process consists of only a download phase at the end of which the secure agent running on the new peer reconstructs a *single* fragment and stores it on the peer's disk.

To model the system, we introduce n-dimensional vectors $\vec{X}_d(t)$, $\vec{Y}_d(t)$, $\vec{Z}_d(t)$ and a $3n$-dimensional vector $\vec{W}_d(t) = (\vec{X}_d(t), \vec{Y}_d(t), \vec{Z}_d(t))$. Vectors $\vec{Y}_d(t)$ and $\vec{Z}_d(t)$ describe the recovery process. The formal definition of these vectors is as follows:

- $\vec{X}_d(t) := (X_{d,1}(t), \ldots, X_{d,n}(t))$ where $X_{d,l}(t)$ is a $[0..s+r]$-valued rv denoting the number of fragments of D stored on peers that are in phase l at time t. $\vec{X}_d(t)$ must verify $S(\vec{X}_d(t)) \in [s-1..s+r]$.
- $\vec{Y}_d(t) := (Y_{d,1}(t), \ldots, Y_{d,n}(t))$ where $Y_{d,l}(t)$ is a $[0..s-1]$-valued rv denoting the number of fragments of D being downloaded at time t to the secure agent from peers in phase l (one fragment per peer).
- $\vec{Z}_d(t) := (Z_{d,1}(t), \ldots, Z_{d,n}(t))$ where $Z_{d,l}(t)$ is a $[0..s-1]$-valued rv denoting the number of fragments of D hold at time t by the secure agent and whose download was done from peers in phase l (one fragment per peer). Observe that these peers may have left the system by time t.

As in the centralized case, $Y_{d,l}(t) \leq X_{d,l}(t)$ for $l \in [1..n]$ at any time t. The number of fragments of D that are available for download at time t is given by $S(\vec{X}_d(t))$. During the recovery process, $S(\vec{Y}_d(t)) + S(\vec{Z}_d(t)) = s$, such that $S(\vec{Y}_d(t)), S(\vec{Z}_d(t)) \in [1..s-1]$. Because the distributed scheme repairs fragments only one at a time, we have $S(\vec{X}_d(t)) \in [s-1..s+r]$. The end of the download phase is also the end of the recovery process. We will then have $\vec{Y}_d(t) = \vec{Z}_d(t) = \vec{0}$ until the recovery process is again triggered.

D is available when $S(\vec{X}_d(t)) \geq s$, unavailable when $S(\vec{X}_d(t)) = s-1$ and $b(\vec{X}_d(t), \vec{Y}_d(t), \vec{Z}_d(t)) \geq s$ (recall the lower bound on the number of distinct fragments in the system introduced in Section 5) and lost otherwise (situation modeled by a single absorbing state a).

According to the description and assumptions listed in Section 3, the state of data D at time t can be represented by $\vec{W}_d(t)$ and the multi-dimensional process $\vec{\mathbf{W}}_d := \{\vec{W}_d(t),\, t \geq 0\}$ is an absorbing homogeneous CTMC with a single absorbing state a. The set of transient states $\mathcal{T}_d$ is the set of elements of $[0..s+r]^n \times [0..s-1]^n \times [0..s-1]^n$ that verify the constraints mentioned above.

The analysis of the absorbing Markov chain $\vec{\mathbf{W}}_d$ that takes values in $\mathcal{T}_d \cup \{a\}$ is very similar to the analysis of $\vec{\mathbf{W}}_c$ in Section 5, we will then only sketch it. In particular, $\vec{R}_d$ and $\vec{Q}_d$ have similar definitions as $\vec{R}_c$ and $\vec{Q}_c$ after replacing the subscript "c" with the subscript "d" whenever needed. The non-zero elements of $\vec{R}_d$ are, for $S(\vec{y}_d) \in [1..s-1]$ and $S(\vec{z}_d) = s - S(\vec{y}_d)$,

$$
\begin{aligned}
r_d(\vec{x}_d, \vec{0}, \vec{0}) &= \sum_{l=1}^{n} x_{d,l}\mu_l, && \text{for} \quad S(\vec{x}_d) = s.\\
r_d(\vec{x}_d, \vec{y}_d, \vec{z}_d) &= \sum_{l=1}^{n} x_{d,l}\mu_l, && \text{for} \quad S(\vec{x}_d) = s-1.\\
r_d(\vec{x}_d, \vec{y}_d, \vec{z}_d) &= \sum_{l=1}^{n} y_{d,l}\mu_l \cdot \mathbb{1}\{b(\vec{x}_d, \vec{y}_d, \vec{z}_d) = s\}, && \text{for} \quad S(\vec{x}_d) = s.
\end{aligned}
$$

We next write the non-zero elements of $\vec{Q}_d$.

The case when a peer leaves the system

The elements of $\vec{Q}_d$ corresponding to the three possible situations are, for $l \in [1..n]$, $m \in [1..n]$, $S(\vec{y}_d) \in [1..s-1]$ and $S(\vec{z}_d) = s - S(\vec{y}_d)$,

$$\begin{aligned}
&q_d((\vec{x}_d, \vec{0}, \vec{0}), (\vec{x}_d - \vec{e}_n^{\,l}, \vec{0}, \vec{0})) = x_{d,l}\mu_l, \\
&\qquad \text{for} \quad S(\vec{x}_d) \in [s+1..s+r]. \\
&q_d((\vec{x}_d, \vec{y}_d, \vec{z}_d), (\vec{x}_d - \vec{e}_n^{\,l}, \vec{y}_d, \vec{z}_d)) = [x_{d,l} - y_{d,l}]^+ \mu_l, \\
&\qquad \text{for} \quad S(\vec{x}_d) \in [s..s+r-1]. \\
&q_d((\vec{x}_d, \vec{y}_d, \vec{z}_d), (\vec{x}_d - \vec{e}_n^{\,l}, \vec{y}_d - \vec{e}_n^{\,l} + \vec{e}_n^{\,m}, \vec{z}_d)) = \frac{y_{d,l}\mu_l[x_{d,m} - y_{d,m} - z_{d,m}]^+}{\sum_{i=1}^{n}[x_{d,i} - y_{d,i} - z_{d,i}]^+}, \\
&\qquad \text{for} \quad S(\vec{x}_d) \in [s..s+r-1].
\end{aligned}$$

The case when a peer rejoins the system

There are three situations in this case exactly like in Section 5. The elements of $\vec{Q}_d$ corresponding to these three situations are, for $l \in [1..n]$, $S(\vec{y}_d) \in [1..s-1]$ and $S(\vec{z}_d) = s - S(\vec{y}_d)$,

$$\begin{aligned}
&q_d((\vec{x}_d, \vec{0}, \vec{0}), (\vec{x}_d + \vec{e}_n^{\,l}, \vec{0}, \vec{0})) = p_l(s + r - S(\vec{x}_d))p\lambda, && \text{for } S(\vec{x}_d) \in [s..s+r-1]. \\
&q_d((\vec{x}_d, \vec{y}_d, \vec{z}_d), (\vec{x}_d + \vec{e}_n^{\,l}, \vec{y}_d, \vec{z}_d)) = p_l(s + r - S(\vec{x}_d))p\lambda, && \text{for } S(\vec{x}_d) \in [s-1..s+r-2]. \\
&q_d((\vec{x}_d, \vec{y}_d, \vec{z}_d), (\vec{x}_d + \vec{e}_n^{\,l}, \vec{0}, \vec{0})) = p_l\ p\lambda, && \text{for } S(\vec{x}_d) = s + r - 1.
\end{aligned}$$

The case when one download is completed during the recovery process

There are three situations in this case, following which download has been completed. If it is the first or any of the $s-2$ subsequent ones, then we obtain the two situations described in Section 5. The third situation occurs when the last download is completed, which is essentially the end of the recovery phase. The elements of $\vec{Q}_d$ corresponding to these three situations are, for $l \in [1..n]$ and $m \in [1..n]$,

$$\begin{aligned}
&q_d((\vec{x}_d, \vec{0}, \vec{0}), (\vec{x}_d, \vec{i} - \vec{e}_n^{\,l}, \vec{e}_n^{\,l})) = s\alpha\ g(\vec{i}, \vec{x}_d)\ f_l(\vec{i}), \\
&\qquad \text{for} \quad S(\vec{x}_d) \in [s..s+r-k], \quad i_l \in [0..x_{d,l}], \quad S(\vec{i}) = s. \\
&q_d((\vec{x}_d, \vec{y}_d, \vec{z}_d), (\vec{x}_d, \vec{y}_d - \vec{e}_n^{\,l}, \vec{z}_d + \vec{e}_n^{\,l})) = S(\vec{y}_d)\alpha\ f_l(\vec{y}_d), \\
&\qquad \text{for} \quad S(\vec{x}_d) \in [s-1..s+r-1], \quad S(\vec{y}_d) \in [2..s-1], \quad S(\vec{z}_d) = s - S(\vec{y}_d). \\
&q_d((\vec{x}_d, \vec{e}_n^{\,m}, \vec{z}_d), (\vec{x}_d + \vec{e}_n^{\,l}, \vec{0}, \vec{0})) = R(l)\alpha, \\
&\qquad \text{for} \quad S(\vec{x}_d) \in [s-1..s+r-1], \quad S(\vec{z}_d) = s - 1.
\end{aligned}$$

And last :

$$q_d(\vec{w}_d, \vec{w}_d) = -r_d(\vec{w}_d) - \sum_{\vec{w}'_d \in \mathcal{T}_d - \{\vec{w}_d\}} q_d(\vec{w}_d, \vec{w}'_d), \quad \text{for} \quad \vec{w}_d \in \mathcal{T}_d.$$

For illustration purposes, we depict in Fig. 2 some of the transitions of the absorbing CTMC when $n = 2$, $s = 4$, $r = 2$, and $k = 1$.

We can now derive closed-form expressions for the distribution of the conditional block lifetime, its expectation, and the two availability metrics, as was done in Section 5. $P(T_d(\mathcal{E}_I) \leq t)$, $\mathrm{E}[T_d(\mathcal{E}_I)]$, $\mathrm{E}[T_d(\mathcal{E}_I, \mathcal{F}_J)]$, $M_{d,1}(\mathcal{E}_I)$ and $M_{d,2}(\mathcal{E}_I, m)$ are given in (4), (5), (7), (9) and (10) respectively, after replacing the subscript "c" with the subscript "d." Alike for the centralized case, we will perform numerical computations as it is not tractable to explicitly invert $\vec{Q}_d$.

7 Discussion: Deploy and tune the P2P backup and storage protocol

In this section, we discuss some practical issues related to how can we use our theoretical framework to tune the key system parameters for fulfilling predefined data lifetime and/or availability requirements.

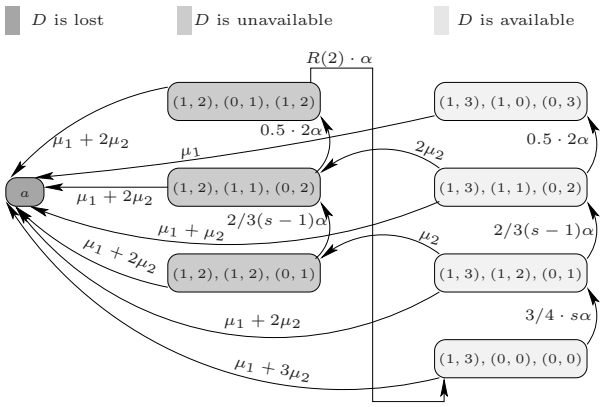


Figure 2: Some transition rates of the Markov chain $\vec{\mathbf{W}}_d$ when $n = 2$, $s = 4$, $r = 2$, and $k = 1$.

We saw in the previous sections that the performance metrics depend on the transition matrix $\vec{Q}$ which depends in turn on the *peers or network* parameters (p, λ, and $\{p_i, \mu_i\}_{i=1,\ldots,n}$), the *recovery process* parameters (α and β) and the *protocol* parameters (s, r and k).

Concerning the *peers or network* parameters, they can be set according to some measurements on the storage environment's peers that report the peers on-times, off-times durations or the disk failure rate such as the work of Nurmi, Brevik and Wolski [28]. The distribution of the *recovery process* and the values of its parameters (α, and β) depend on the block/fragment sizes and the upload/download capacities of peers, the work-load in the overlay network, and the inter-network connections capacities. To fit the distribution of the recovery process into an appropriate distribution and to estimate its parameter's values, one may do some simulations using for example our packet-level simulator presented in [11] or build on the flow-level simulation model introduced in [13] to simulate a large network. Another solution is to estimate the fragment/block download times using the log files of some P2P applications or FTP clients run on some peers involved in the P2P storage solution.

The *protocol* parameter s depends on the choice of the size of data blocks and fragments. Nowadays, block sizes in P2P *storage* systems are usually set to either 4MB, 8MB or 9MB and fragment sizes are set somewhere between 256KB and 1MB. A helpful factor to choose from these values can be the average size of the stored files in the system, so that the fragmentation overhead associated with the transmission of data is still negligible with respect to the files sizes. Concerning the two *key* parameters r and k, we compute numerically some contour lines (curves along which the function has constant values) of each of the performance metric functions studied in this paper as a function of r and k at desired values, and we report them in a figure.

After that, we select the operating point of the P2PSS that ensures the desired data lifetime, and availability for a reasonable storage overhead r/s and acceptable recovery threshold k. Intuitively, smaller threshold values enable smaller amounts of redundant data at the cost of higher bandwidth utilization. The trade-off here is between efficient storage use (small r) and efficient bandwidth use (large k).

8 Numerical Results

We have solved numerically the closed-form expressions (4), (5), (7), (9) and (10) using MATLAB with realistic values. The models presented in Sections 5 and 6 can be seen as a generalization of those presented in [10]. In particular, in [10] the number of phases of the hyper-exponential distribution of on-times is $n = 1$. This reduces the state-space of the Markov chain, and then solving the basic models

is much less time consuming than solving the general models presented in this paper. As a matter of curiosity, we will compare in this section the results obtained with all these models when considering an environment that is known to violate the exponential assumption on peers on-times made in [10]. This allows us to see whether the models in [10] are robust against a violation of this assumption and can justify or not the importance of the general models.

Once this question addressed, we evaluate the lifetime and availability of data stored on P2PSS running in different contexts. Throughout the numerical computations, we consider both centralized- and distributed-recovery implementations and both erasure-code and regenerating code based systems. Last, we illustrate how our models can be used to engineer storage systems and we discuss the impact of the blocks/fragments sizes on the performance.

8.1 Parameter values

Our mathematical models have been solved numerically using a set of parameters values.

Network parameters λ, $\{p_i, \mu_i\}_{i=1,\ldots,n}$ and p. We consider three sets of values that represent three different environments. These correspond to three data sets that have been studied in the literature. The sets *CSIL* and *Condor* have been collected by Nurmi, Brevik and Wolski [28]. The *CSIL* set reports uptime of machines in the Computer Science Instructional Laboratory (CSIL) at the University of California, Santa Barbara. As for the *Condor* set, it reports CPU idle times of peers in a Condor pool [7] at the University of Wisconsin, in other words, it reports the availability of peers to perform an external job (the Condor pool offers processing time to the whole Internet). This can be seen as the time during which a peer may participate in a storage system. The *All-pairs-ping* set has been obtained by Stribling [35] after the processing of ping requests between each pair of PlanetLab [29] nodes. Each node pings every other node roughly 4 times an hour. A 10-probes ping is considered successful only if at least one probe response was received.

The sets *CSIL* and *Condor* are best fit by a hyper-exponential distribution according to the analysis in [28], even though they report different flavors of peer "availability". An exponential distribution is found to "reasonably" fit the *All-pairs-ping* data set in [30]. The basic characteristics of the three data sets considered here and the corresponding values of the peers parameters are reported in Table 2. Out of the three mentioned scenarios, *Condor* experiences the highest dynamics environment. This behavior has been reported elsewhere concerning peers on the Internet. For instance, it has been observed in [4, 5] that on average peers join/leave the Internet 6.4 times per day and that sessions times are typically on the order of hundreds of minutes on average. In this paper, the *Condor* system will mirror the Internet context and *CSIL* and PlanetLab environments will mirror a stable environment such as local area or research laboratory networks where machines are usually highly available.

As an exponential distribution is found to "reasonably" fit the peers availability in the *All-pairs-ping* data-set, PlanetLab-like systems can be studied using the models presented in [10] while the *CSIL* and *Condor* contexts need the more general models developed in this paper. Justifying this last point is the objective of the next section.

The value of λ, or equivalently the mean off-time, has been set to have the same *peers availability* across all environments. This measure, given in row 16 of Table 2, is the probability of finding a peer connected or equivalently the percentage of on-times in a peer life cycle. We have set $p = 0.8$ in the *Condor* scenario as peers churn rate is very high and $p = 0.3$ otherwise, namely in the *CSIL* and *All-pairs-ping* scenarios. This is to reflect that disconnections in stable environments are likely due to software or hardware problems.

Protocol parameters s, r and k. Motivated by the discussion in 7, and for *CSIL*- and *Condor*-like systems, we will consider block sizes of 4MB and fragment sizes of 1MB and then $s = 4$. For PlanetLab

Table 2: Data sets characteristics and corresponding peers parameters values

Data set	*CSIL*	*Condor*	*All-pairs-ping*
Context	LAN	Internet	PlanetLab
Covered period	8 weeks	6 weeks	21 months
Number of peers	83	210	200–550
On-times distribution	H_3 [28] (best fit)	H_2 [28] (best fit)	Exp. [30] (reasonable)
On-times parameters			
p_1	0.464	0.592	1
p_2	0.197	0.408	—
p_3	0.339	—	—
$1/\mu_1$ (hours)	250.3	0.094	181
$1/\mu_2$ (hours)	1.425	3.704	—
$1/\mu_3$ (hours)	33.39	—	—
Mean on-time (hours)	127.7	1.567	181
Mean off-time (hours)	48	0.522	61
Percentage of on-times	0.727	0.75	0.750
Persistence probability p	0.3	0.8	0.3

context, we considered block sizes of 8MB and fragment sizes of 1MB and then $s = 8$. In the *CSIL* scenario where peers churn is low, we vary the redundancy r from 1 to $1.5s = 6$. In the high dynamic scenario (*Condor*), we vary the redundancy r from 1 to $3s = 12$ (resp. to $1.5s = 6$) when the recovery is distributed (resp. centralized). In all the considered scenarios, we vary the threshold k from 1 to r.

Recovery process parameters α and β. Fragments download/upload times depend on the upload/download capacities of the peers and the central authority when needed . The measurement study [33] of P2P *file sharing* systems, namely Napster and Gnutella, shows that 78% of the users have downstream bottleneck of at least 100 Kbps. Furthermore, 50% of the users in Napster and 60% of the users in Gnutella use broadband connections (Cable, DSL, T1 or T3) having rate between 1Mbps and 3.5Mbps. Moreover, a recent experimental study [19] on P2P VoIP and file sharing systems shows that more than 90% of users have upstream capacity between 30 Kbps and 384 Kbps, where the downstream is of the order of some Mbps (like Cable/ADSL). Based on the two mentioned studies, we assume the peers' upload capacity that is dedicated to a single connection to be either 384 Kbps or 150 Kbps and the total upload capacity of the central authority to be 1.5 Mbps. Hence, considering fragments of size 1MB, we obtain $1/\alpha = 22$ or 56 seconds and $1/\beta = 6$ seconds.

Depending on this discussion and on our simulation resultas presented in [12], we consider that $1/\alpha = 56$ second for SCIL context and 88 seconds for Condor and erasure codes (EC) based systems and 23 seconds for Condor and regenrating code (RC) based systems. When RC is enforced, the recovery process is faster due to the fact that the size of the downloaded fragments is smaller. Recall that in EC a new peer downloads s fragments of size 1MB, where in RC it downloads s pieces of size $\geq \frac{s}{s^2 - s + 1} * S_B/s = 0.308$MB in order to download a fragment of size $\geq s * 0.308 = 1.232$MB. We considered fragments of size of 1.35MB for the regenerating codes.

8.2 Discussion and comparison with the models of [10]

As mentioned previously, the models presented in [10] are a special case of the models developed here, namely when the number of phases of the hyper-exponential distribution of on-times is $n = 1$. To

avoid any ambiguity, the models presented in [10] will be referred to as "basic" models whereas the ones developed in Section 5 and 6 will be referred to as "general" models. Because of the reduced state-space, solving the basic models is much less time consuming than solving the general models. The basic models describe well PlanetLab-like environments. However, one question remains: do they model any environment?

To answer this question, we deliberately select a scenario in which peers have been identified to have a non-exponential on-times distribution, namely the *Condor* scenario, and evaluate the lifetime of a block of data D using both models and compare the results. In [28], a 2-stage hyper-exponential distribution is found to best fit the *Condor* data set, but the authors identify as well the parameter of the exponential distribution that best fits the same data.

Table 3 reports the expected data lifetime obtained with the distributed-recovery implementation for $s = 4$, $1/\lambda = 0.522$ hour, $1/\alpha = 22$ seconds, $p = 0.8$ and different amounts of redundancy r and recovery thresholds k. Results provided by the general model with $1/\mu_1 = 0.094$ hours, $1/\mu_2 = 3.704$ hours, $p_1 = 0.592$ and $p_2 = 1 - p_1$ are in column 3; those given by the basic model with $1/\mu = 1.543$ hours (best exponential fit found in [28]) and $1/\mu = 1.567$ (first moment of the H_2 distribution) can be found in columns 4 and 6 respectively. The relative error between $\mathrm{E}[T_d(\mathcal{E}_{s+r})]$ (general model; column 3) and $\mathrm{E}[T_d(s+r)]$ (basic model [10]; columns 4 and 6) are reported in columns 5 and 7.

Table 3 reveals that the basic model returns substantially different results than those of the general model. Since the distribution of peers on-times is hyper-exponential in the *Condor* scenario, the results obtained through the general model are the correct ones.

We conclude that the basic model presented in [10] does not capture the essence of the system performance when peers on-times are not exponentially distributed. *Henceforth, we will use the basic model in scenarios with the* All-pairs-ping *characteristics, and the general model in scenarios with the characteristics of either* CSIL*, or* Condor.

8.3 Performance analysis

We have solved numerically (4), (5), (7), (9) and (10) given that all $s + r$ fragments of D are initially available, considering either *Condor* or *CSIL* context, and either the centralized or distributed recovery scheme. Results are reported partially in Tables 4 and 5. Results in PlanetLab (*All-pairs-ping*) context (when $n = 1$) while an erasure code in enforced can be found in Table 6.

It appears that, whichever the scenario or the recovery mechanism considered, the expected data lifetime increases roughly exponentially with r and decreases with an increasing k. Regardless of the

Table 3: Expected data lifetime (expressed in hours) in a *Condor* scenario using a distributed-recovery scheme. Comparison between $\mathrm{E}[T_d(\mathcal{E}_{s+r})]$ (general model) and $\mathrm{E}[T_d(s+r)]$ (basic model [10]).

$s = 4$		H_2 fit [28]	Exponential fit [28]		equating 1st moments	
		$\mathrm{E}[T_d(\mathcal{E}_{s+r})]$ (in Hours)	$\mathrm{E}[T_d(s+r)]$	error	$\mathrm{E}[T_d(s+r)]$	error
$k = 1$	$r = 2$	1.283	0.78	-39.2%	1.017	-20.73%
	$r = 4$	4.2	3.453	-17.79%	4.09	-2.62%
	$r = 6$	12.62	14.04	11.25%	14.44	14.42%
$k = 2$	$r = 2$	0.46	0.492	6.96%	0.633	37.61%
	$r = 4$	2.31	2.34	1.3%	2.74	18.61%
	$r = 6$	4.488	10.464	133.16%	10.732	139.13%

Table 4: Expected lifetime in Condor context for centralized or distributed repair scheme and EC or RC are enforced

Condor context		$\mathrm{E}[T(\mathcal{E}_{s+r})]$ (in days)		
	recovery	cent.	dist.	dist.
	redundancy	EC	EC	RC
	$1/\alpha$ in sec.	88	88	23
	$1/\beta$ in sec.	6.3	—	—
$k=1$	$r=2$	0.365	5.34e-02	1.607
	$r=4$	20.769	0.175	34.7
	$r=5$	129.551	0.305	144.15
	$r=6$	730.132	0.526	533.22
	$r=10$	—	3.933	—
	$r=12$	—	9.558	—
$k=2$	$r=2$	0.123	1.92e-02	0.43
	$r=4$	6.955	9.62e-02	8.21
	$r=5$	45.901	0.187	37.81
$k=4$	$r=4$	0.222	1.86e-02	0.33

Table 5: Expected lifetime and first availability metric in CSIL context for distributed repair scheme and EC

CSIL context		$\mathrm{E}[T(\mathcal{E}_{s+r})]$ (in months)		$M_1(\mathcal{E}_{s+r})$	
	$s=4$	distributed repair		distributed repair	
		$1/\alpha = 22$ sec	$1/\alpha = 56$ sec	$1/\alpha = 22$ sec	$1/\alpha = 56$ sec
$k=1$	$r=2$	3.102	1.022	5.972	5.945
	$r=4$	209.67	24.29	7.929	7.853
$k=2$	$r=2$	0.248	0.134	5.026	5.023
	$r=4$	27.34	5.29	6.973	6.93
$k=4$	$r=4$	0.219	0.117	5.09	5.119

context considered, the distributed scheme, while an erasure code in enforced, yields a significantly smaller expected data lifetime than the centralized scheme for the same redundancy mechanism, especially when peers churn rate is high; cf. columns 3-4 in Table 4. Observe how the use of regenerating codes improves very well the performance of the system even in dynamic context; cf. columns 4-5 and 3-5. This is due to the fact that each new peer in the RC downloads the size of one fragment unlike the case of EC where each peer downloads the size of the whole block of data in order to recover one fragment. However, in a stable context like SCIL, EC with the distributed repair scheme provides a good performance as shown in Table 5.

We conclude that *when peers churn rate is high, only the centralized repair scheme can be efficient if erasure code is used as redundancy mechanism, as long as the storage overhead is kept reasonable small (that is $r/s \leq 2$).* As the distributed repair scheme is more scalable than the centralized one, it will be a good implementation choice in large networks where hosts have a good availability. Regenerating codes scheme is very promissing for the storage objective in dynamic context even with the distributed repair scheme.

Table 6: PlanetLab context: Expected lifetime and first availability metric while an erasure code in enforced

PlanetLab context		Expected data lifetime E$[T(s+r,0)]$ (in months)		Expected number of fragments $M_1(s+r,0)$	
	$s=8$	cent. repair	dist. repair	cent. repair	dist. repair
$k=1$	$r=2$	0.32	0.11	7.81	8.04
	$r=4$	2.15	1.05	11.01	8.68
	$r=6$	17.12	7.61	13.18	9.80
	$r=8$	262.16	46.24	15.11	12.12
$k=2$	$r=4$	0.81	0.37	10.34	8.19
	$r=6$	6.95	3.20	12.76	9.25
	$r=8$	110.03	23.34	14.72	11.37
$k=4$	$r=8$	13.33	4.34	13.77	9.81

Setting the system's key parameters.

We illustrate now how our models can be used to set the system parameters r and k such that predefined requirements on data lifetime and availability are fulfilled. We assume the recovery mechanism is distributed and the context is similar to CSIL. We have picked two contour lines of each of the performance metrics studied in this paper and report them in Fig. 3 that is done using the general model and Fig. 4 that is done using the basic model.

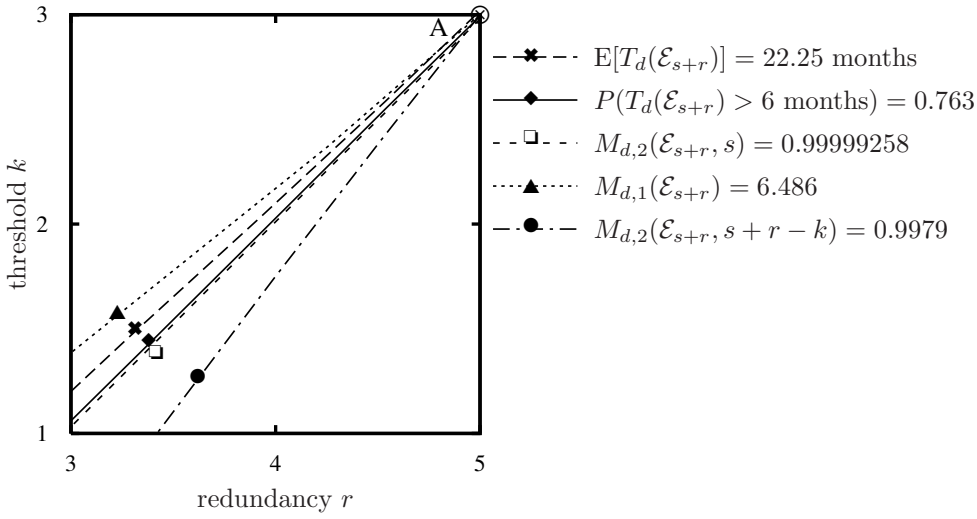
Consider Now point A in Fig. 4 which corresponds to $r=6$, $k=1$ and $s=8$. Selecting this point as the operating point of the P2P2P storage system ensures (roughly) the following: given that each data is initiated with $s+r$ available fragments, then (i) the expected data lifetime is 18 months; (ii) only 11% of the stored data would be lost after 3 months; (iii) as long as D is not lost, 13 fragments of D are expected to be in the system; (iv) during 99.7% of its lifetime, D is available for download; and (v) during 80% of the lifetime of D, at least $s+r-k=13$ fragments of D are available for download in the system. Observe that the storage overhead, r/s, is equal to 0.75.

Consider point A (resp. B) in Fig. 3 which corresponds to $r=5$ and $k=3$ (resp. $k=2$). Recall that $s=4$ (for both points). Selecting point A (resp. B) as the operating point of the P2PSS ensures the following: given that each data is initiated with $s+r=9$ available fragments, then (i) the expected data lifetime is 22.25 (resp. 188.91) months; (ii) 23.7% (resp. 3.13%) of the stored data would be lost after six months; (iii) as long as D is not lost, 6.486 (resp. 7.871) fragments of D are expected to be available in the system; (iv) during 99.9992% (resp. 99.9999%) of its lifetime, D is available for download; and (v) during 99.79% (resp. 99.7%) of the lifetime of D, at least $s+r-k=6$ (resp. $s+r-k=7$) of its fragments are available. Observe that the storage overhead, r/s, is 1.25 for both operating points and it is the *lazy* policy that is enforced ($k>1$). Observe how the performance metrics improve when k is decreased, even by one. However, this incurs more bandwidth use because the recovery will be more frequently triggered.

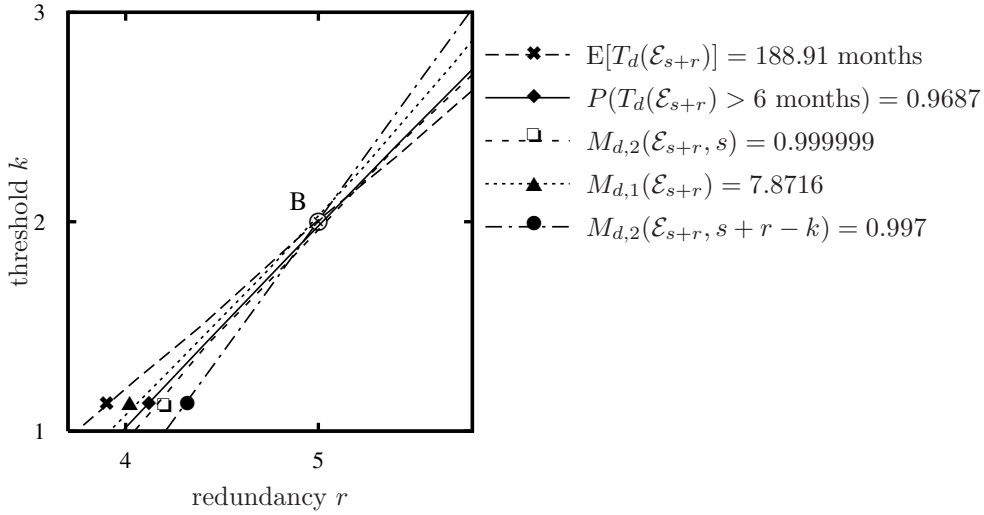
Impact of the size of blocks/fragments, the parameter s and α. Given the size of data D, a larger size of fragments translates into a smaller s and a larger expected fragment download time $1/\alpha$. We have computed all pairs (r,k) with $s=8$ and $s=16$ that ensure $P(T_c(\mathcal{E}_{s+r})>3 \text{ months})=0.89$ in the PlanetLab context, i.e., only 11% of the total data would be lost after 3 months. In particular, operating points $r=6$ and $k=1$ with $s=8$, and $r=12$ and $k=7$ with $s=16$ satisfy the above requirement, and additionally yield the same storage overhead (namely, 0.75). But, and this is important, the former

point invokes the recovery process much more often (and potentially unnecessarily) than the latter point, suggesting that *large fragments size reduces the efficiency of the recovery mechanism.* This observation should be moderated by the fact that fragments size when $s = 8$ is twice their size when $s = 16$, yielding a different bandwidth usage per recovery. Although the performance of the system seems to be better when the number of fragments increases, due to decrease their sizes, each fragment adds some coordination and control overhead.

In the same direction, we observe from Table 5 that when the expected fragment download time $1/\alpha$ becomes smaller (roughly the half, from 56 to 22 sec.), the expected data lifetime increases roughly exponentially. Most P2PSS and other P2P application, e.g. P2P video streaming and file sharing, use small fragment size. So, few IP packets are sent for each fragment. This can minimize the packetization delay.



(a) Settings of point A: $s = 4$, $r = 5$ and $k = 3$



(b) Settings of point B: $s = 4$, $r = 5$ and $k = 2$

Figure 3: Contour lines of performance metrics (CSIL context, distributed repair).

Numerical computations Complexity. It is evident that he cost of computing the solution of the

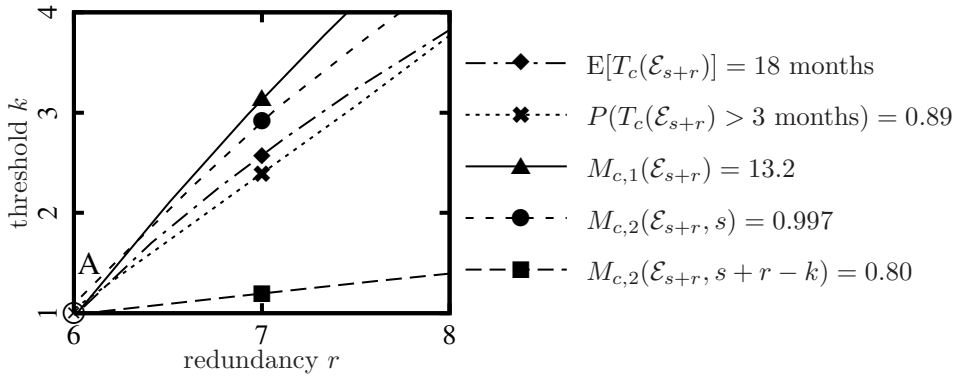


Figure 4: Contour lines of performance metrics (PlanetLab context, centralized repair, Basic Model $n = 1$).

absorbing CTMC numerically depends in particular on the values of n, s and r that will determine the finite number of the MC states. The closed-form expressions of the performance metrics were solved using multi-threaded structured programming using MATLAB. Some of them runs on one single machine with the following principal characteristics: multi-threaded processor Intel(R) Core(TM)2 Duo of 2.66GHz, 4GB RAM + 4GB swap running Fedora Core 5. The execution times varies between some hours and some days. From the analysis in Section [5] and [6]. We found that the set of transient states $\mathcal{T}_c$ is a subset of $[0..s+r]^n \times [0..s-1]^n \times [0..s]^n \times [0..s+r-1]^n \times [0..s+r-1]^n$, and $\mathcal{T}_d$ is the set of elements of $[0..s+r]^n \times [0..s-1]^n \times [0..s-1]^n$ that verify the constraints mentioned in the Sections. Therefor, the state-space size, and hence the computational cost, of the absorbing CTMC is polynomial in the values of s and r and an exponential in the number of n. Fortunately, the value of s in most practical system is less than 10 and r takes usually from (0.5 – 3) times of s (we provided enough details about the parameter values in the new submission in Section [7]). By analyzing real traces of different network topologies and settings, as explained in [28], we found that values of n vary between 1 and 3. Thus, the computational complexity is still reasonable.

Fortunately, the benefits of the P2P paradigm is not limited to storage application. Thanks to the open source P2P cloud computing project [6, 21] that is developed at INRIA, we could reduce the computation time needed for some experiments from two weeks when run them on one machine to less than one day using ProActive (usually resources of about 5 machines in the cloud were allocated for the experiments)

9 Conclusion

We have proposed general analytical models for evaluating the performance of two approaches for recovering lost data in distributed storage systems and three redundancy schemes. Numerical computations have been performed to illustrate several issues of the performance. We conclude that, using our theoretical framework, it is possible to tune and optimize the system parameters for fulfilling predefined requirements. We find that, in stable environments such as local area or research laboratory networks where machines are usually highly available, the distributed-repair scheme offers a reliable, scalable and cheap storage/backup solution regardless the redundancy scheme. This is in contrast with the case of highly dynamic environments, where the distributed-repair scheme is inefficient with erasure codes as long as the storage overhead is kept reasonable. P2PSS with centralized-repair scheme are efficient in any

environment but have the disadvantage of relying on a centralized authority. Regenerating codes scheme is very promissing for P2PSS applications. However, the analysis of the overhead cost (e.g. computation, bandwidth and complexity cost) resulting from the different redundancy schemes with respect to their advantages (e.g. simplicity), is left for future work.

References

[1] S. Alouf, A. Dandoush, and P. Nain. Performance analysis of peer-to-peer storage systems. In *Proc. of 20th International Teletraffic Congress (ITC)*, volume 4516 of *LNCS*, pages 642–653, Ottawa, Canada, 17–21 June 2007.

[2] F. Baskett, K.M. Chandy, R.R Muntz, and F.G Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2):248–260, 1975.

[3] R. Bhagwan, D. Moore, S. Savage, and G.M. Voelker. Replication strategies for highly available peer-to-peer storage. In *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 153–158. Springer, 2003.

[4] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proc. of 2nd IPTPS*, Berkeley, California, February 2003.

[5] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G.M. Voelker. Total Recall: System support for automated availability management. In *Proc. of ACM/USENIX NSDI '04*, pages 337–350, San Francisco, California, March 2004.

[6] Denis Caromel. Keynote lecture proactive parallel suite: Multi-cores to clouds to autonomicity. In *IEEE 5th International Conference on Intelligent Computer Communication and Processing, 2009. ICCP 2009.*, 2009.

[7] Condor: High throughput computing. `http://www.cs.wisc.edu/condor/`, 2007.

[8] F. Dabek, F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of ACM SOSP '01*, pages 202–215, Banff, Canada, October 2001.

[9] O. Dalle, F. Giroire, J. Monteiro, and S. Pérennes. Analysis of failure correlation impact on peer-to-peer storage systems. In *Proc. of 9th International Conference on Peer-to-Peer Computing (P2P09)*, Seattle, USA, September 8-–11 2009. To appear.

[10] A. Dandoush, S. Alouf, and P. Nain. Performance analysis of centralized versus distributed recovery schemes in P2P storage systems. In *Proc. of IFIP/TC6 Networking 2009*, volume 5550 of *LNCS*, pages 676–689, Aachen, Germany, May 11–15 2009.

[11] A. Dandoush, S. Alouf, and P. Nain. A realistic simulation model for peer-to-peer storage systems. In *Proc. of 2nd International ICST Workshop on Network Simulation Tools (NSTOOLS09), in conjunction with the 4th International Conference (VALUETOOLS'09)*, Pisa, Italy, October 19 2009.

[12] A. Dandoush, S. Alouf, and P. Nain. Simulation analysis of download and recovery processes in P2P storage systems. In *Proc. of 21st International Teletraffic Congress (ITC)*, Paris, France, September 2009.

[13] A. Dandoush and A. Jean-Marie. Flow-level modeling of parallel download in distributed systems. In *Third International Conference on Communication Theory, Reliability, and Quality of Service (CTRQ), 2010*, pages 92–97, June 2010. Best Paper Award.

[14] Abdulhalim Dandoush, Sara Alouf, and Philippe Nain. Lifetime and availability of data stored on a P2P system: Evaluation of recovery schemes. Technical Report RR-7170, INRIA Sophia Antipolis, January 2010.

[15] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. In *Proc. of 26th IEEE Conference on Computer Communications (INFOCOM)*, Anchorage, Alaska, USA, May 6–12 2007.

[16] A.G. Dimakis, V. Prabhakaran, and K. Ramchandran. Decentralized erasure codes for distributed networked storage. *IEEE Trans. Inform. Theory, 2006,*, 52, 2006.

[17] A.G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A survey on network codes for distributed storage. In *Proc. IEEE 99*, pages 476–489. IEEE Computer Society, 2011.

[18] C. Grinstead and J. Laurie Snell. *Introduction to Probability.* American Mathematical Society, 1997.

[19] A. Guha, N. Daswani, and R. Jain. An experimental study of the skype peer-to-peer VoIP system. In *Proc. of 5th IPTPS*, Santa Barbara, California, February 2006.

[20] P. Harrison and S. Zertal. Queueing models of RAID systems with maxima of waiting times. *Performance Evaluation Journal*, 64(7-8):664–689, August 2007.

[21] INRIA. Proactive parallel suite. http://proactive.activeeon.com/index.php.

[22] A. Kermarrec, E. L. Merrer, G. Straub, and A. V. Kempen. Availability-based methods for distributed storage systems. In *In 31st IEEE International Symposium on Reliable Distributed Systems*, pages 151 – 160. IEEE Computer Society, 2012.

[23] H. Kobayashi and B. L. Mark. On queuing networks and loss networks. In *Proc. 1994 Annual Conference on Information Sciences and Systems*, Princeton, NJ, March 1994.

[24] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *In Proc. of the IEEE International Symposiumon on Cluster Computing and the Grid*, 2010.

[25] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proc. of ACM ASPLOS*, pages 190–201, Boston, Massachusetts, November 2000.

[26] M. Martalo, M. Amoretti, M. Picone, and G. FerrariSporadic. decentralized resource maintenance for p2p distributed storage networks. *Parallel Distributed Computer, 2014,*, 74, 2014.

[27] M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models. An Algorithmic Approach.* John Hopkins University Press, Baltimore, 1981.

[28] D. Nurmi, J. Brevik, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *Proc. of Euro-Par 2005*, volume 3648 of *LNCS*, pages 432–441, Lisbon, Portugal, August 2005.

[29] PlanetLab. An open platform for developing, deploying, and accessing planetary-scale services. `http://www.planet-lab.org/`, 2007.

[30] S. Ramabhadran and J. Pasquale. Analysis of long-running replicated systems. In *Proc. of IEEE Infocom*, Barcelona, Spain, April 2006.

[31] I.S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of SIAM*, 8(2):300–304, June 1960.

[32] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. of ACM SOSP '01*, pages 188–201, Banff, Canada, October 2001.

[33] S. Saroiu, P.K. Gummadi, and S.D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of Multimedia Computing and Networking (MMCN)*, San Jose, Cailfornia, January 2002. Best Paper Award.

[34] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-to-Peer lookup service for Internet applications. In *In Proc. of ACM SIGCOMM*, pages 149–160, San Diego, California, August 2001.

[35] J. Stribling. PlanetLab - All Pairs Pings. `http://pdos.csail.mit.edu/~strib/pl_app`, 2005.

[36] L. Taoyu, C. Minghua, C. Dah-Ming, and C. Maoke. Queuing models for peer-to-peer systems. In *Proceedings of the 8th International Conference on Peer-to-peer Systems*, IPTPS'09, pages 4–4, Berkeley, CA, USA, 2009. USENIX Association.

[37] UbiStorage. `http://http://www.ubistorage.com`.

[38] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proc. of IPTPS '02, Cambridge, Massachusetts*, volume 2429 of *Lecture Notes in Computer Science*, pages 328–337, March 2002.

[39] Wuala. The wuala project. http://www.wuala.com.

[40] Z. Yang, Y. Dai, and Z. Xiao. Exploring the costavailability tradeoff in p2p storage systems. In *In ICPP'09: Proceedings of the 2009 International Conference on Parallel Processing*, pages 429–436. IEEE Computer Society, 2009.