Analysis and Optimization of Peer-to-Peer Storage and Backup Systems

Abdulhalim Dandoush

UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE LINFORMATION ET DE LA COMMUNICATION

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice - Sophia Antipolis

Montion : **Informatique**

présentée et soutenue par

Abdulhalim Dandoush

Analysis and optimization of peer-to-peer storage/backup systems

Thèse dirigée par **Philippe Nain** et **Sara Alouf**

Soutenue le 29 Mars 2010

Jury:

Président :	Walid	DABBOUS	INRIA Sophia Antipolis, France
Directeur :	Philippe	NAIN	INRIA Sophia Antipolis, France
Co-Directeur :	Sara	ALOUF	INRIA Sophia Antipolis, France
Rapporteurs :	Phuoc	TRAN-GIA	University of Wuerzburg, Allemagne
	Emilio	LEONARDI	Politecnico di Torino, Italie
Examinateurs :	Sébastien	CHOPLIN	Ubistorage, France
	Fabrice	LE FESSANT	INRIA Saclay, France
Membre invité :	Alain	JEAN-MARIE	LIRMM, INRIA and CNRS, France

Thèse

L'analyse et l'optimisation des systèmes de stockage de données dans les réseaux pair-à-pair

Analysis and optimization of peer-to-peer storage/backup systems

Abdulhalim DANDOUSH

April 2010

Résumé

Cette thèse évalue les performances de systèmes de stockage de données sur des réseaux de pairs. Ces systèmes reposent sur trois piliers: la fragmentation des données et leur dissémination chez les pairs, la redondance des données afin de faire face aux éventuelles indisponibilités des pairs et l'existence d'un mécanisme de recouvrement des données perdues ou temporairement indisponibles. Nous modélisons deux mécanismes de recouvrement des données par des chaînes de Markov absorbantes. Plus précisément, nous évaluons la qualité du service rendu aux utilisateurs en terme de longévité et de disponibilité des données de chaque mécanisme. Le premier mécanisme est centralisé et repose sur l'utilisation d'un serveur pour la reconstruction des donnée perdus. Le second est distribué : la reconstruction des fragments perdus met en oeuvre, séquentiellement, plusieurs pairs et s'arrête dès que le niveau de redondance requis est atteint. Les principales hypothèses faites dans nos modèles sont validées soit par des simulations soit par des traces réelles recueillies dans différents environnements distribués. Pour les processus de téléchargement et de recouvrement des données nous proposons un modèle de simulation réaliste qui est capable de prédire avec précision le comportement de ces processus mais le temps de simulation est long pour de grands réseaux. Pour surmonter cette restriction nous proposons et analysons un algorithme efficace au niveau flux. L'algorithme est simple et utilise le concept de (min-max). Il permet de caractériser le temps de réponse des téléchargements en parallèle dans un système de stockage distribué.

Mots-cles: systèmes pair-à-pair, évaluation de performance, chaîne de Markov absorbante, approximation champ moyen, simulation au niveau paquet, simulation au niveau flux.

ABSTRACT

This thesis characterizes the performance of peer-to-peer storage systems in terms of the delivered data lifetime and data availability. Two schemes for recovering lost data are modeled and analyzed: the first is centralized and relies on a server that recovers multiple losses at once, whereas the second is distributed and recovers one loss at a time. For each scheme, we propose simple Markovian models where the availability of peers is exponentially distributed, and more elaborate models where the latter is hyper-exponentially distributed. Our models equally apply to many distributed environments as shown through numerical computations. These allow to assess the impact of each system parameter on the performance. In particular, we provide guidelines on how to tune the system parameters in order to provide desired lifetime and/or availability of data. The key assumptions made in the models are validated through intensive packet-level simulations or real traces collected from different distributed environments. In fact, we propose a realistic simulation model implemented on the Network Simulator (NS-2) for both download and recovery processes. Although this simulator can accurately predict the behaviour of the latter processes while considering the impact of several constraints such as the heterogeneity of peers and the the underlying network topologies, this simulator requires however relatively long time. To overcome this scalability limitation, we propose and analyze an algorithm, we called the "progressive-filling flow-level algorithm" or PFFLA. The algorithm is efficient in time and quite simple and uses the concept of "Progressive-Filling" (or max-min fairness), hence the name. The validation of this algorithm consists in characterizing the *distribution* of the response time of parallel downloads in a distributed storage system, through simulations.

Keywords: Peer-to-Peer systems, performance evaluation, absorbing Markov chain, mean-field approximation, packet-level simulation, fow-level simulation.

ACKNOWLEDGMENTS

This thesis is the result of some years of research that has been done since I came to MAESTRO's group. Over time, I have met and worked with great people. It is a pleasure to convey my gratitude to them all in my humble acknowledgment.

First and foremost I wish to thank my supervisors, Philippe NAIN (Leader of the MAESTRO Team-Project at INRIA) and Sara ALOUF (Associate researcher at INRIA), whose expertise, understanding and patience, enriched me in many domains (e.g. Stochastic process, queueing theory, fluid approximation). I appreciate their vast knowledge and skills in many research and non-research areas. I will never forget their assistance in analyzing problems, understanding results, writing articles and reports. For sure, I am grateful to them for giving me the opportunity to work on this exciting research topic and the freedom to carry it out.

I would like to thank Alain Jean-Marie (Research director at INRIA and LIRMM) for taking time out from his busy schedule to guide me finishing the last chapter of my thesis. His invaluable advices on simulating parallel downloads and his exceptional insights into engineering and queueing systems greatly enriched my knowledge and my skills in different directions (e.g. programming, generating traffic). Thanks to him I learned how to formulate and describe distributed algorithms.

These dear three supervisors have provided me unflinching encouragement and support in various ways and in several critical moments. Thanks God for give me the opportunity to work with them.

I gratefully acknowledge Giovanni Neglia (Associate researcher at INRIA) for the helpful discussions on several issues related to my thesis and on general subjects in our small life. I will miss him a lot, he is someone special out of my friends.

Personal thanks to all my dear colleagues at MAESTRO, PLANETE and MASCOTTE Teams-Projects for their loving kindness and for the amusing time shared together, in particular, to Ahmad Al Hanbali, Mouhamad Ibrahim, Danil, Giuseppe, Saed, Ricardo, Utku, Vincenzo, Konstantin, Alonso, Amar, Alberto, Imed, M. Jaber, B. Ben-Romdhane, Faker, Amin Ismail, J. Monteiro, D. Mazauric and E. Isnard.

I am very thankful to our nice and lovely assistant Ephie Deriche for her help, kindness and the never forgotten amazing discussions during the daily lunch.

Special thanks to Fabrice Huet, Florin and Fabien of the OASIS Team-Project for their help in the Proactive tool. This tool helped me to do a huge computations in a short time.

Great thanks to INRIA that provides a very nice and rich environment of research which is one of the best in all Europe (from my experience) and maybe in the world.

Many thanks to all my teachers, doctors, professors and friends who taught me or let me teach with them or discussed with me or corrected me such as Y. Lecourtier, C. Barakat, W. Dabbous, E. Altman, M. Syska, J.-C. Bermond, F. Baude, A. Legout, F. Giroire, O. Dalle, J. Moulierac, S. Pérennes, E. Lety, E. Isnard, S. Choplin, P. Tran-Gia, and K. Pawlikowski.

I am very thankful to my dear professors, teachers and assistants in Syria such as Radwan Dandah, Ahmad Sakr Ahmad, Talal el Ateki, Kassem Kabalan, Jamal khalifah, Hassen el Ahmad, Adnan Meatrmawi, Moustafa Dalila, Haitham el Radwan, Zoheir Wakkaf, Halah Mahmoudi, Mariam Saii, Solafa Salameh, Moustafa Fawal and all the staff of the Computer Sciences, Electronic and Telecommunication Faculties in the Tishreen University (Lattakia).

I would also like to thank all my family (in particular, my father Hammoud and mother Houriah) for the support they provided me through my entire life. I last and not least acknowledge my wife Soheir and my nice daughter Nour Houriah, without whose love and encouragement, I would not have finished this thesis. I always thanks God who let me meet Soheir. She is a great women.

Abdulhalim DANDOUSH
INRIA Sophia Antipolis and University of Nice-Sophia Antipolis, France

CONTENTS

FIGURES

TABLES

1

Background, Motivation and Related Work

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 plays the role of both supplier and consumer of resources, in contrast to the traditional client-server model where only servers supply, and computers consume. Applications that use this distributed network provide enhanced scalability and service robustness as all the connected computers or peers provide some services. Peers in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network. As already mentioned, each peer receives/provides a service from/to other peers through the overlay network; examples of such services are computing (sharing the capacity of its central processing unit), data upload (sharing its bandwidth capacity), data storage (sharing its free storage space), as well as support to locate resources, services and other peers.

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. In fact, file sharing is the dominant P2P application on the Internet (see [66, 55, 41, 38, 24, 25]), allowing users to easily contribute, search and obtain content. P2P file sharing applications received a special interest from users thanks to the increasing popularity of the mp3 musical file format since 1991 and to the ability to share videos and films for free.

To provide a proper background, we will briefly describe, in Section 1.1, the basic taxonomy of P2P overlay network. For completeness and in view of the high popularity of P2P file sharing applications, we will introduce some of them as examples of the overlay architectures even if they are not the object of further study in this thesis. The techniques of P2P backup and storage systems will be introduced then with some existing examples in Section 1.2. Section 1.3 overviews the related works and motivations. Last, Section 1.4 introduces our contribution and presents the organization of this thesis.

1.1 P2P overlay architectures

Based on how the peers in the overlay network are linked to each other on top of the physical network topology, and on how services are shared and located, we can classify the P2P networks into two general topologies: unstructured and structured network.

1.1.1 Unstructured P2P network

Unstructured P2P networks organize peers or nodes into a random graph topology and use floods or random walks to discover data stored by overlay nodes. In other words, nodes connect themselves to the overlay without taking care of their neighbors IDs or names. This approach supports arbitrarily complex queries and it does not impose any constraints on the overlay topology or on data placement.

In general, three topologies of the unstructured architecture can be distinguished. First, there are fully distributed P2P systems, like the original **Gnutella** protocol [41], where all peers are completely equal and there is no central authority. As soon as a peer joins the system, it establishes several connections with peers, called neighbors. To search an entity in the system, a peer sends a query to its neighbors. If a neighbor stores the requested entity, it replies to the requester. Otherwise, it forwards the query to its own neighbors, and so on until a given depth. This depth is similar to time-to-live (TTL) of packets in IP networks. This type of search is called flooding. However, the cost of flooding the network increases linearly with the number of nodes which limits the system scalability if the depth is high. In addition, their is no guarantee on the response time, in particular, for the non-popular files.

Second, hybrid peer-to-peer systems, like **Kazaa** [55], use the concept of supernodes: nodes that handle indexing and caching blocks of data through small set of peers. Supernodes are dynamically elected depending on the available bandwidth capacity and processing power. Therefore, all queries are initially forwarded to supernodes to get served. Hence, discovery time is reduced in comparison with fully decentralized systems. There is no unique point of failure as the case of centralized peer-to-peer systems (explained below) and there is no need

to route messages by flooding as the case of fully distributed systems. A better implementation of Gnutella relies on supernodes.

Third, centralized peer-to-peer systems, like **Napster** [66], rely on a central server for indexing functions and for bootstrapping the entire system. In fact, Napster was the first P2P file sharing systems and it is one of the pioneers of digital music. Although considered a P2P system, this model follows the standard client-server paradigm because it uses a central server to maintain the directories of the shared files stored on the system's peers, to locate resources and route requests between peers. However, downloading occurs in a P2P manner; peers connect to each other to download pieces of data. This topology suffers from the single point of failure problem (the central authority) and does not scale very well.

One of the enormously studied centralized P2P file sharing systems is **BitTorrent** [12]. It relies on the terms *seeds, leechers, torrents, trackers and peers/pieces selection algorithms*. Files are split into fixed-size fragments stored initially on the first publisher which is called a *seed* and its availability in the network allows other users, called *leechers*, to connect and begin to receive pieces of different fragments of the file. Once a peer has a complete fragment of the seed, BitTorrent allows it to become a source (server) for that portion of the file. To share a document, the seed first creates a *torrent* file that contains metadata about the document to be shared and about the computer that coordinates the file distribution; the *tracker*. Peers that want to download the document must first obtain its torrent file, and connect to the specified tracker, which gives them a list of other peers that contain fragments of the document. Throughout the transfer, each computer will query the tracker, telling it how much it has downloaded and uploaded. In fact, finding torrent files and the single point of tracker failure are the major problems in the design of BitTorrent. But, once the torrent file is located and the corresponding tracker is available, BitTorrent provides better performance than the other file sharing protocols thanks to the multiple (parallel) download mechanism of pieces of requested data and due to its fragments and peers (uploaders and downloaders) selection algorithms, in particular the rarest first, optimistic unchoking, and choking algorithms. However, some new BitTorrent clients (e.g. recent Azureus clients [4]), have support for multible distributed trackers in a structured way to improve the resources look-up phase and to overcome the problem of trackers failures that may let the system to be unavailable. In other words, new BitTorrent implementations are moving to work over a structured P2P architecture.

1.1.2 Structured P2P network

In structured P2P networks, nodes are assigned uniform random nodeIds (node identifier) from a large identifier space. Data items or objects are assigned unique identifiers called "keys", selected from the same identifier space. Chord [89], Tapestry [101] and Pastry [83] use a circular identifier space of n-bit integers modulo 2^n ($n = 160$ for Chord and Tapestry, $n = 128$

for Pastry). Content Addressable Network (CAN) [77] uses a d-dimensional cartesian identifier space, with 128-bit nodeIds that define a point in the space. If n is big, each "key" is dynamically mapped by the overlay to a unique active node with a very high probability. This node is called the *key's root* or the *node responsible for the "key"*. To deliver messages efficiently to the root, each node maintains a routing table consisting of the nodeIds and IP addresses of the nodes to which the local node maintains overlay links. Messages are forwarded across overlay links to nodes whose nodeIds are progressively closer to the key in the identifier space. Structured P2P systems use consistent hash function (e.g. SHA-1 [87, 86]) to assign a global address or identifier space to all nodes and all keys. Consistent hash functions are hash functions with some additional advantageous properties, i.e., they let nodes join and leave the system with minimal disruption [54]. Unlike unstructured P2P networks, the main concept in structured networks is key-based routing. Key-based routing means that a set of keys is associated with "values" (addresses of the contents) in the address space. Structured P2P networks are usually considered as distributed hash table (DHT) which is a distributed dictionary in which every entry is composed of a "key" and an associated "value" that indicates the location of the content of that key.

It is proven that the cost of a look-up in most DHT-like systems grows only logarithmically in the number of nodes in the system, and that this system provides a good data balance as with high probability each node is responsible for $1/N$ of the identifier space, where N is the total number of nodes in the system. Structured P2P networks ensures that any peer can efficiently route a request to some peer that has the desired data object, even if the data object is rare. Ensuring efficient routing can be achieved in unstructured P2P systems only for the popular files. However, in a very dynamic and unstable environment, it is hard and costly to maintain a structured network.

In P2P applications, three data organization schemes can be considered. First, a "key" can be the identifier of the whole file of data (hashing value of its name or title or contents) as in PAST [84], a persistent global storage utility that has been built using Pastry [83] overlay network. Second, in other P2P applications, files are fragmented into equally sized fragments, and a "key" in this data organization is the identifier of a fragment of data of a file as the case of CFS [27], a Cooperative File System that has been built using Chord [89] to provide storage services. A third data organization consists of dividing files into equally sized blocks of data, each block of each file is fragmented into many equally sized fragments, and the "key" in this last scheme will be the identifier of a block of data as the case of UbiStorage [92], a P2P backup system. Each of these data organization schemes has its advantages and disadvantages. System objectives, data download time, availability, and the system implementation and design issues may favor one scheme over the others in a particular scenario.

Let us proceed to the evolution of P2P Backup and Storage Systems.

1.2 P2P backup and storage systems

Parallel to the evolution of P2P file sharing systems, P2P backup or/and storage systems have been developed. They are less popular systems because they are not dedicated exclusively for sharing music or videos and because people do not trust P2P for storing their private data. For later use, we will define two important metrics: data availability and data lifetime or durability.

Over time, a peer or a node can be either *connected* to or *disconnected* from the storage system. We refer to as *on-time* (resp. *off-time*) a time-interval during which a peer is always connected (resp. disconnected). During a given time, we can represent the node availability by the percentage of the sum of on-time durations over that time. So at any point in that time, a peer can be available with some probability. During a peer's off-time, the data items stored on that peer are momentarily unavailable to the users of the system. In consequence, any data item can be available at any time with some other probability related to node's availability that store this data item. To be able to download a data item, a node that stores a complete copy of it or an enough number of nodes that store all its distinct fragments must be active (connected to the system) for some time.

Some data items (or fragments of them) can be lost from the system due to permanent departure of some nodes or disk failures. We define data lifetime as the time until the moment at which the data is considered to be lost; can not be downloaded or reconstructed completely, given that it was initially available completely. So, before to lose the data, data can be available or not available temporarily but it is durable (not lost definitely).

We can distinguish between backup and storage systems. P2P backup systems aim to provide long data lifetime without constraints on the recovery or the reconstruction time. In other words, data must be durably stored but not necessarily immediately available for download on the contrary to storage systems. For this reason, backup systems designers are interested in the permanent departures of peers rather than the intermediate disconnections even if the disconnections durations are long. In this thesis, we will provide guidelines on how to engineer both P2P backup and storage systems in order to satisfy given data lifetime and/or availability requirements.

Some of the recent efforts for building highly available and durable systems based on the P2P paradigm include Intermemory [45, 21], Freenet [24], OceanStore [59], CFS [27], PAST [84], Farsite [14, 1], Total Recall [10], Wua.la [99] and Allmydata [2]. Although these storage systems are scalable, tolerant against unexpected catastrophes and economically attractive compared to traditional client/server systems, they pose many problems such as reliability, confidentiality and availability.

In these systems, peers are free to leave and join the system at any time. As a result of the

intermittent availability of peers, ensuring high availability of the stored data is an interesting and challenging problem. To ensure data reliability and availability in such dynamic systems, redundant data is inserted into the system. Redundancy can be achieved either by replication or by using erasure codes.

However, using redundancy mechanisms without recovering lost data is not efficient, as the level of redundancy decreases when peers leave the system. Consequently, P2P storage systems need to compensate the loss of data by continuously storing additional redundant data onto new peers. We denote by new peer, an available peer that does not store already redundant information of the considered data (e.g. a given block).

In the following sections, we will introduce the redundancy and recovery mechanisms, the two key techniques of any P2P backup and storage system.

1.2.1 Redundancy mechanisms

Redundancy is a key mechanism in any reliable or storage system to ensure some level of reliability and to increase data availability and durability. Concerning data storage, it was first used in 1987 in the Redundant Arrays of Inexpensive Disks (RAID) systems [72]. RAID systems allow computers to achieve high levels of storage reliability from inexpensive and less reliable hard disk components, by arranging the devices into arrays for redundancy. Redundancy provides fault tolerance, so that all or part of the data stored in the array can be recovered in the case of disk failure. In fact, there are three schemes in RAID to manage the stored data: (i) replication or mirroring over more than one disk, (ii) striping, the splitting of data across more than one disk, (iii) and use of the technique of error-correcting code (ECC) [49]. The basic idea is to combine two or more physical hard disks into a single logical unit and based on how data is managed (splitted, coded or replicated over disks), we can distinguish between seven levels of RAID systems from RAID-0 up to RAID-6. For example, in RAID-1, the whole data or a hard disk is mirrored without any coding or splitting over a second disk.

The cost typically associated with redundancy is a reduction of disk capacity available to the users, since the implementations require either a duplication of the entire data set, or an error-correcting code (ECC) [49], also known as a forward error correction (FEC) in information theory.

There is a wide range of mechanisms available for producing redundant representations of data. However, in the context of P2P backup and storage systems, we will distinguish between two major mechanisms, replication and erasure coding (a case of FEC).

Replication

There are two replication levels used in P2P backup and storage systems:

- *The whole-file-level replication scheme*. A file f is replicated r times over r different peers (as in PAST [84]) so that the tolerance against failures or peers departure is equal to r. In other words, r is the number of peers storing copies of data object that may leave the network or fail without the data object being lost. 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 [27].

Erasure coding

This 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. [78, 17]), r redundant fragments are added as depicted in Figure 1.1. To download a block of data, any s fragments are needed out of the $s+r$ (downloading the size of the original block S_B). 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. 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 $s/(s+r)$. Intermemory [45, 21], OceanStore [59], Total Recall [10] and UbiStorage [92] are some examples of existing P2P systems that use erasure coding mechanisms to provide some level of system reliability and data availability.

A new class of codes, so-called regenerating codes (RC) has been proposed recently in [35]. 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 [35], 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 $F_{RC} = \theta * S_B$, where S_B stands for the size of the given block of data to be stored. A newcommer (a new peer in our notation) 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; the new peer will store all the downloaded data whose size is equal to the size of the stored fragments instead to download the equivalent of original block size, in the case of EC, to regenerate one fragment and deleting later the downloaded fragments. In the same way, downloading the block, by a data collector, in EC requires the download of its size ($s * F_{EC} = S_B$), where in RC, it requires the download of $s * F_{RC} = s * \theta * S_B = \sigma S_B$, where $\beta > 1$. The authors show that $\beta \rightarrow 1$ as $s \rightarrow \infty$. Until the time of writing this thesis, the regenerating codes is not yet used in any P2P system.

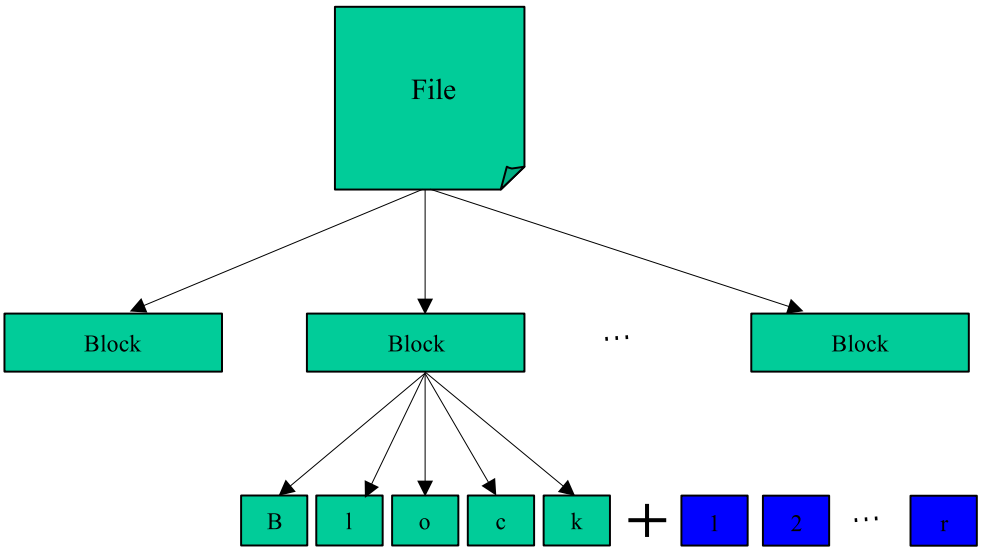


Figure 1.1: General data organization scheme in erasure coded systems.

Let us use hereafter the notation of the erasure coding mechanism introduced in Section 1.2.1 to capture the case of the three redundancy mechanisms; the *whole-file-level replication* scheme, the *fragment-level replication* and the *erasure coding* scheme. When the size of the block D is equal to the whole size of the file f ($b = 1$), and after setting $s = 1$, the r redundant fragments will be simple replicas of the unique file f as the case of the *whole-file-level replication* scheme (e.g. Napster [66], PAST [84] and Gnutella [41]). For each block D of data of the file f, and after setting $s = 1$, we obtain r simple replicas of D of the file f as the case of the *fragment-level replication* scheme (e.g. CFS [27] and eDonkey [64]) where the size of a fragment is equal to the whole size of a block of data. Even for erasure coding mechanism,

some systems use it for $b = 1$ like Carbonite [23] and OceanStore [59], and then the whole file of data is fragmented into s fragments to which, and using the erasure coding algorithm, r redundant fragments are added. UbiStorage is an example of those systems splitting each file into equally sized blocks ($b \neq 1$), and then splitting each block into s equally sized fragments and adding to them r redundant fragments using erasure code. Note that when fixing the sizes of blocks and fragments in the system, the values of s and r will be the same for all the blocks of data stored in the system. This notation—and hence the modeling presented in this thesis—is general enough to study any system that uses one of these schemes by playing with the values of b and s.

Replication vs. erasure coding

The comparison between the redundancy mechanisms was the subject of several papers.

In [97], Weatherspoon and Kubiatowicz characterize the availability and durability gains provided by an erasure-resilient system. They quantitatively compare replication-based and erasure-coded systems. They show that erasure codes use an order of magnitude less bandwidth and storage than replication for systems with similar durability.

In [8], the authors show that an erasure codes scheme makes backup systems more scalable than replication and block-level replication schemes as the required availability gets higher. They show as well that the scalability of the block-level scheme with respect to the total storage required is even lower than that of the replication scheme. This is due to the fact that when enough replicas of a given block of data fail such that any single block cannot be found, then the entire file object that contains the considered block becomes unavailable. The erasure coding approach reduces the traffic of the replication by using the computing power. In fact, the time to encode and to decode the data in erasure coding are considered negligible with respect to the download time of the data fragments; cf. [17].

Utard and Vernois perform in [93] another comparison between replication and erasure coding mechanisms through a simple stochastic model for node behavior. They observe that simple replication schemes may be more efficient than erasure codes only in the presence of low peers availability. However, the authors of the P2P storage system TotalRecall [10] (presented in Sect. 1.2.3) say that replication can be highly inefficient in low-availability environments since many storage replicas are required to tolerate potential transient failures.

1.2.2 Recovery policies and mechanisms

P2P backup and storage systems need to compensate the loss of data due to peers departure from the system by continuously storing additional redundant data onto other hosts to be able to achieve high data durability or/and availability.

In fact, similarly to file sharing systems, backup and storage systems may rely on a central authority that reconstructs files or fragments when necessary; these systems will be referred to as *centralized-recovery systems*. Alternatively, secure agents running on some active nodes can reconstruct by themselves the data to be stored on the nodes disks. Such systems will be referred to as *distributed-recovery systems*.

Recovery policies

Regardless of the recovery mechanism used, two repair policies can be enforced. In the **eager** policy, when the system detects that one peer has left the network, it immediately initiates the reconstruction of all data hosted by that failed peer, and stores them on new peers upon recovery. So, in erasure-coded system, a fragment of a given block D of data is reconstructed as soon as it has become unavailable due to a peer disconnection. Using this policy, data only becomes unavailable when peers fail more quickly than failures can be detected and repaired. This policy is simple but makes no distinction between permanent departures that need to be recovered, and transient disconnections that may do not. Moreover, operating in this manner generates a great number of reconstruction processes and therefore leads to a sudden increase in bandwidth use upon each failure. Glacier [48] and CFS [27] are examples of systems that use this policy. However, such a policy can be used in stable networks that have good connectivity where peers tend likely to stay on-line as long as possible and when it is unacceptable to loose data (e.g. the environment of INRIA where computers are online all the time except if an unexpected error occurs such as a disk failure or a problem of energy).

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. So in this policy, the repair is delayed until the number of unavailable fragments of a block D of data reaches a given threshold, denoted k. In this case, we must have $k \leq r$ since D is lost if more than r fragments are missing from the storage system. This alternative policy 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 peers departures for extended periods of time as in TotalRecall [10]. This policy is called **lazy repair** because the explicit goal is to delay repair work for as long as possible. This policy allows the reintegration of redundant fragments back into the system instead of creating additional fragments ahead of time and hence it reduces the global usage of the bandwidth in the 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.

Centralized-recovery scheme

Let us consider a block D of data and assume that the system misses k (threshold of recovery) fragments so that lost fragments have to be restored.

In the centralized implementation, a central authority will: (1) download *in parallel* s fragments of D from peers currently available, (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. In fact, 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. Step 1 (resp. Step 3) ends executing when the download (resp. upload) of the last fragment is completed.

Although this scheme is implemented only in centralized P2P systems when both b,s $\neq 1$ like UbiStorage [92], it is implemented in the distributed P2P systems when either $b = 1$ as in Carbonite [23] or $s = 1$ as in CFS [27].

Hence, this recovery scheme can be seen as a repair policy in which all missing fragments have to be reconstructed at the end of the recovery process. In fact, this recovery mechanism can be done in a DHT-like systems for both b,s $\neq 1$ if the node responsible for a file stores its b blocks on b nodes and each of these nodes becomes responsible for the received block. Then, each node responsible for a block executes the three steps mentioned above so that it plays the role of the centralized server for this given block. The b nodes and those that will store fragments of data must be chosen among the neighbors of the node responsible for the file (resp. the block). The identifiers of these neighbors nodes are equal to or larger than the identifier assigned to the node responsible for the file (resp. the block) as the *leaf set* in Pastry and the *successors set* in Chord. The size of these leaf set and successors set have to be always larger than b and $s + r$ respectively.

Distributed-recovery scheme

In the distributed implementation, a secure agent on one new peer is notified of the identity of *one* out of the k unavailable fragments to reconstruct it. Upon notification, the secure agent (1) downloads s fragments 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 so 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.

Again, this scheme can be done in a centralized system. The server can do the same steps (or

ask a peer to do them) in the objective of recovering only one missing fragment and allowing more time to reintegrate fragments that eventually "reappear" in the system due to a peer reconnection. So, this scheme is actually a policy that recovers only one missing fragment at the end of a single recovery process.

1.2.3 Some existing P2P backup and storage systems

Intermemory [45, 21] is one of the earliest distributed storage systems (proposed in 1998). It was proposed in a period of growing interest in digital libraries and it was motivated by the problem of preserving digital documents [82]. The system can be envisioned as either a public peer-to-peer application where the peers are servers at libraries or a server-to-server application where libraries and institutions cooperate to create a robust storage substrate for their archives. It uses an erasure coding to provide a durable archival storage. It implements a distributed block-store substrate on which arbitrary data structures, including conventional file systems, can be built. The addressing approach combines hashing, pseudo-random generators, and a distributed name server. The system uses database synchronization between random pairs of Intermemory nodes to propagate metadata and data fragments efficiently and to provide an automated self-repair mechanism. Many ideas from the original Intermemory project are now contributing to the Intermemory.net commercial project.

Freenet [24] is one of the pioneers among anonymous publication systems that ensures true freedom of communication over the Internet and prevents censorship of distributed data. It is build on a routing overlay whose interface is similar to Tapestry's one [101]. Freenet is a loosely structured system that uses file and peer identifier similarity to produce an estimate of where a file may be located, and a chain mode propagation approach to forward queries from peer to peer where no peer is privileged over any other peer. It uses lazy replication to increase accessibility of popular data thanks to its request mechanism in which popular data are transparently replicated by the system and mirrored closer to requesters to improve the response time in the requester region for the requested file. Requested files are copied as well onto each peer along the way providing fault-tolerance against the failure of the source node.

Files in Freenet are identified by unique binary keys. Three types of keys are supported, the simplest of which is based on applying a hash function (e.g. SHA-1 [87]) on a short descriptive text string that accompanies each file as it is stored in the network by its original owner. Each Freenet node maintains its own local data store, which it makes available to the network for reading and writing, as well as a dynamic routing table containing the addresses of other nodes, based on local knowledge, and the files they are thought to hold. To search for a file, the user sends a request message specifying the key and a timeout (hops-to-live) value which is decremented at each peer to prevent infinite loops.

In order to join the network, a peer has to know the address of one or more of the existing

peers, and hence, the problem of establishing initial network connection holds. It has also obvious disadvantages in terms of discovering documents, name collisions, etc. An unpopular file might disappear from the network if the source peer fails.

A class of systems similar to Freenet but dedicated to long term archival is **OceanStore** [59]. It is designed using a cooperative utility model in which consumers pay the service providers certain fees to ensure access to persistent storage. It provides a universal availability to its users through a two-tiered storage system. The upper tier consists of powerful servers that are well connected and have good bandwidth capacities. These servers work together to serialize changes, archive results and provide a storage service with support of nomadic data; data that is allowed to flow and be cached freely and data are separated from the phisical location. To that end, an erasure coding and self-monitoring mechanisms are used. The second tier (called lower tier) is for storage replication and consists of less powerful hosts, including users' peers, which mainly provide storage resources to the system. It has as a goal that data can be cached anywhere, and anytime on lower tier peers. This policy is called promiscuous caching and does not aim to ensure data durability, but rather to place data closer to the users in order to guarantee the best download time of stored data. OceanStore uses a hierarchical hashing method to verify the integrity of each fragment. It generates a hash of each fragment, and recursively hashes over the concatenation of pairs of hashes to form a binary tree. OceanStore servers use Tapestry to disseminate encoded file blocks efficiently, and clients can quickly locate and retrieve nearby file blocks by their ID, despite server and network failures. One important aspect of OceanStore that differs from existing systems is the fact that the archival mechanisms are tightly coupled with the update activity based on the Byzantine agreement protocol [20].

Cooperative File System (**CFS**) [27] is a P2P read-only storage system that provides provable guarantees for the efficiency, robustness, and load-balance of file storage and retrieval. CFS achieves this with a completely decentralized architecture. In CFS, multiple providers of content cooperate to store and serve each other data. Spreading the total load evenly over all participant hosts lowers the total cost of the system, since each participant needs to provide capacity only for the average load, not for the peak load, and hence, this solution can scale to large systems. A file is divided into constituent fragments that are stored among different peers. CFS has three layers: (i) File-System (FS) which interprets fragments as files and presents a file system interface to applications, (ii) the DHash (Distributed Hash) layer performs fragment fetches for the peer, distributes the fragments among the servers, and maintains cached and replicated copies, DHash finds fragments using (iii) the Chord [89] location protocol, which operates in time logarithmic in the number of servers. As mentioned in Section 1.2.1, CFS uses the fragment-level replication scheme to increase availability, so DHash replicates each fragment on r CFS servers to increase availability, maintains the r replicas automatically as servers come and go, and places the replicas in a way that clients can easily find them. However, CFS (as

Glacier [48]) *eagerly* maintains redundancy, which however does not explore the trade-offs of cost and resilience. DHash places a fragment's replicas at the r servers immediately after the fragment's successor on the Chord ring. Servers close to each other on the ID ring are not likely to be physically close to each other, since a server's ID is based on a hash of its IP address. This provides the desired independence of failure. DHash can easily find the identities of these servers from Chord's l entry successor list (note that $l \geq r$ must hold). CFS's caching scheme is similar to Freenet's in the sense that both allow cached copies of data along the query path from client to where the data was found. A problem that can arise in CFS (or in PAST [84]) is that if a node joins and takes responsibility for a portion of the ID space, a considerable number of files (or fragments) may need to be transferred to it. Thus, when nodes join or leave these systems, a high cost will often result, as a burst of transfers is triggered. To avoid this problem, it may be useful to store pointers to the right locations instead of transferring completely data, as done in OceanStore. In addition, it is shown in [8] that the fragment-level replication scheme, which is used in CFS, is the least efficient redundancy scheme.

TotalRecall [10] is a P2P storage system that guarantees a predefined high availability level by automatically adapting the degree of redundancy and frequency of repair to the distribution of peers failures. It uses a modified version of the DHash peer-to-peer object location service described in the CFS paper [27]. After estimating the availability of peers, and predicting their future availability based on past behavior, the system applies a replication mechanism in high-availability environments and an erasure coding mechanism in low-availability environments. To the contrary of finding in [93] as we have shown in Section 1.2.1, the authors of TotalRecall show that replication can be highly inefficient in low-availability environments since many storage replicas are required to tolerate potential transient failures. TotalRecall was one of the first systems to exploit the fact that most peers have short session time but long life-time in the system, and thus use *lazy* maintenance of redundantly stored content unlike CFS [27] and Glacier [48] for example.

In [42], authors introduce **Carbonite** as a new replication algorithm for DHT-backup systems like CFS [27] and PAST [84]. Carbonite aims to provide high durability for backup systems to the contrary of P2P Storage systems like TotalRecall [10] whose design was driven by the goal of achieving a very high availability. Carbonite separates durability from availability so that the recovery process must create new copies of data objects faster than permanent disk failures but not faster than any temporary departure. Hence, Carbonite-based systems are not suitable for P2P storage systems but they can provide a good solution for data backup with less bandwidth cost than in storage systems like TotalRecall that repair unavailable data with a high frequency to maintain high availability. This causes to waste bandwidth by creating new replicas in response roughly to any failure (temporary or permanent). Authors conclude that a careful choice of data placement policies can decrease repair time. To that end, Carbonite

remembers which replicas were stored on nodes that have failed so that it can reintegrate data stored on them when they return back to the system. Once the recovery algorithm is triggered at a given threshold of redundancy, Carbonite regenerates all missing replicas.

The **UbiStorage** [92] French company was created in the early 2006 and is currently addressing the on-line backup market for small and medium companies. It uses the Ubiquitous Storage (**US**) [76, 88] prototype that aims to provide a virtual storage device to each user which insures data durability. The main mechanism used to insure data durability is redundancy based on erasure code. US uses a centralized authority to control the system and to locate data. However, current efforts try to control and administrate the system in a distributed way to increase the system scalability and to make the system self-organized. Contrarily to TotalRecall, US does not care about temporary disconnections of peers and pays attention only to disk failures or permanent disconnections of peers. Although data are distributed on end user peers, US provides for its clients a special storage device, *US box*, that may still be accessible even if the peers are turned off.

The **Tahoe** [95] project is a recent distributed filesystem, which addresses files backup on multiple machines to protect against hardware failures throughout a decentralized architecture. Blocks of data to be stored are encrypted, then split up into several redundant fragments using "erasure coding", with $s = 3$ and $r = 7$ by default. Tahoe is composed of three layers. The lowest layer is effectively a Distributed Hash Table (DHT). In this DHT, there are a large number of "slots" which are filled with arbitrary data. The middle layer is the Virtual Drive, which organizes these slots into directories. The top-most layer is an application or presentation service (interface); something that uses the virtual drive for some purpose. The most mature interface is a RESTful HTTP interfaces, in which PUT, GET, and POST are used to add and retrieve files and directories. Tahoe is a free software sponsored by allmydata.com. The *Tahoe-LAFS* client is included in the new version of Ubuntu 9.10. Using this open-source software to do some real experiments can be one of our objective in the next steps.

1.3 Related work and motivations

Although the literature on the architecture and file system of distributed backup and storage systems is abundant as we saw in the previous section, 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.

In this section, we first discuss some existing analytical, simulation and measurement works that aim to understand and evaluate the peers availability and the download process in a distributed environment. These are two important factors in any analytical work that aims to evaluate and optimize the P2P backup and storage systems in terms of data lifetime and availability as we will see throughout this thesis. Last we briefly review related performance evaluation works of data availability and durability in the P2P backup and/or storage systems; the main objective of this thesis.

1.3.1 Works related to peers availability

A major problem in any P2P application is that peers are free to join and leave temporarily (for long or short time) the system at any time. Some peers can fail due to software or hardware problems and so they leave permanently the system. This leave/join phenomenon is named *churn*. In general, joining the system has no remarkable impact on the system. It can add some delay in routing results until the system detects the new joint nodes and updates the pointers of data location toward the right nodes. However, the departure and failure events have an important negative impact because they may cause data loss.

In [80], Rhea et al. addressed the global churn rate in DHT-like systems. They performed an empirical evaluation of the routing layers of existing DHT implementations, and they found that these implementations are unable to withstand the short session times observed in the wild. Moreover, beyond a certain level of churn, lookups in existing systems either take excessively long to complete, or fail to complete altogether, or return inconsistent results. In addition, the ability of new nodes to join the DHT is often impaired. So, they presented Bamboo, a DHT that handles high levels of churn.

Binzenhöfer and Leibnitz [11] proposed a distributed algorithm to estimate the churn rate in DHT systems (structured overlay) by exchanging measurement observations among neighbors list (e.g. successors in Chord or leafs in Pastry). They based on a peer behavior model in which a peer can stay on-line or off-line for some time. The duration of an on-line and off-line session are random variables that follow a generic distribution. They consider that a failed peer will rejoin the system with its data at a later point in time. For the case of an exponential distribution of the on-line/off-line durations, they derive a close form for the probability distribution of the time between two observed leave or two observed join events.

Ramabhadran and Pasquale analyzed, in [75], the *All-pairs-ping* data set [90] that reports measures of both uptime and downtime for PlanetLab [74] nodes. By plotting the cumulative distribution function of each duration (uptime/downtime), they conjecture from the figures that an exponential distribution is a reasonable fit for both uptime and downtime durations of the PlanetLab nodes.

Characterizing machine availability both in local and wide area environments has been the

focus of [69]. 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. Although, an exponential distribution seems good enough to fit the machine availability distribution (uptimes durations) in PlanetLab-like environments as concluded the authors of [75], considering a hyper-exponential distribution will give more accurate and general work that is applicable to many distributed environments as the exponential distribution is a special case of the hyper-exponential distribution. This work comes to support key assumptions of our models.

1.3.2 Works related to download and recovery processes

One measure of the quality of the service given by the distributed storage/parallel download infrastructure is the time it takes to retrieve the complete document. This in turn depends on the throughput of the different flows created to obtain the fragments of this document. Their values are, *a priori*, a function of the demand and capacities of the complete network entities: clients, servers and links.

The basic problem of predicting the instantaneous shares of the bandwidth received by each flow of a TCP-based network has received quite some attention in the last 15 years, in connection with the notion of *fairness*; yet, there is no clear consensus in the literature on a simple formula or algorithm to give a reasonable solution of this problem.

On the one hand, some authors have shown that the dynamics of TCP have been shown to be quite chaotic is some situations. Other authors on the other hand, have argued that TCP tends to share the bandwidth between flows reasonably. For instance, Heyman *et al.* [52], followed by Fredj *et al.* [43], have studied a single bottleneck link shared by a given number of identical sources that alternately send documents through the shared link and stop sending for a randomly thinking time. They showed through simulations that TCP shares *fairly* the bottleneck (that is, in equal shares) and they introduced analytical tools that can predict the expectation of the transmitting rate. Varki proposed in [94] a simple approximation for the expected response time based on the fork-join model. Massoulié and Roberts proposed in [63] a model similar to that of [52] where the inter-flows arrival times are independent and identically-distributed random variables (iid rv) and follow an exponential distribution. They studied the network as M/G/1 PS queue. In [22], Chiu and Eun, the authors have focused on the average download time of each user in a P2P network while considering the heterogeneity of service capacities of peers. They point out that the common approach of analyzing the average download time based on average service capacity is fundamentally flawed.

Other studies have put forward the concepts of max-min fairness, proportional fairness,

balanced fairness and utility-based resource-sharing models (see *e.g.* [16] and the reference therein). One conclusion of these studies is that throughput allocations resulting from the use of the TCP protocol for infinitely long flows are usually *not* max-min fair. However, the results of Bonald and Proutière [15] suggested that when the flows are dynamic (flows are continuously created and have a finite duration), the average throughput obtained by flows under various sharing mechanism tend to be similar. It is quite possible that, from a practical perspective, the predictions obtained with a max-min fair sharing mechanism may be "good enough".

One purpose of this thesis is to assess whether max-min fairness for the allocation throughput is a proper model when evaluating response times of parallel downloads through the development and the analysis of an algorithm that we called the "progressive-filling flow-level algorithm" or PFFLA. We will see in Chapter 5 that PFFLA can be used as the core of a flow-level simulator of P2P storage systems.

1.3.3 Works related to ata lifetime and availability

Actually, few studies have developed analytical models, for both P2P backup and storage systems, with the goal of understanding the trade-offs between the availability and lifetime of data on the one hand and the redundancy involved in storing the data and the repair frequency on the other hand. In addition, capturing the behavior of both eager and lazy repair policies and both replication-based and erasure code-based systems in modeling, and accommodating both temporary and permanent disconnections of peers are not well done.

In [8], Bhagwan, Savage and Voelker have provided a probabilistic analysis for the efficiency of whole-file replication, fragment-level replication, and erasure coding redundancy schemes that can be used to overcome the temporary disconnections of peers and to increase reliability of the P2P storage systems. They studied as well the storage costs of maintaining a given level of availability in the long term by regularly recovering missing data after every time t (e.g. ten months). They showed that using erasure codes makes the system more scalable than whole-file replication and fragment-level replication schemes as the required availability gets higher.

However, this study gives only the expected availability of any file stored in the system based only on the replication factor and the expectation of peers availability. Moreover, the authors neglect the bandwidth factor and they consider only the storage costs. They concluded that even in environments with pervasive failure it is possible to offer a storage service with a high degree of availability at a moderate cost in storage overhead. In this thesis, we will come to an opposite conclusion in some scenarios. We will see that when the churn rate is high and the required levels of *both the availability and the durability metrics are high* as well, the system is *infeasible if* we want to consider a reasonable storage and bandwidth-use overhead when a *distributed repair scheme* is considered. Our outcome is in agreement with the analysis done in

[13, 81].

In [13], Blake and Rodrigues have argued that the cost of dynamic membership makes the cooperative storage infeasible in transiently available peer-to-peer environments. In other words, when redundancy, data scale, and dynamics are all high, the needed inter-connections capacities in the system are unreasonable when clients desire to download files during a reasonable time. Rodrigues and Liskov [81] arrived at the same conclusion. In high churn environments, erasure codes provide a large benefit for building P2P storage systems but the bandwidth cost is too high.

Utard and Vernois [93] have performed a comparison between the full replication mechanism and erasure codes through a simple stochastic model for the node behavior. They observed that simple replication schemes may be more efficient than erasure codes in presence of very low peers availability.

Ramabhadran and Pasquale have analyzed in [75] a storage system that uses *full replication* for data reliability. So, in this aspect, [75] is the closest work to ours even though the analysis does not apply for erasure-coded systems (we will see later that our models apply to either replicated or erasure-coded systems). The authors developed a Markov chain analysis, then derived an expression for the lifetime of the replicated state and studied the impact of bandwidth and storage limits on the system. However—and these are major differences with the work presented here—transient disconnections are ignored in their model and only the distributed-repair scheme is considered. The recovery process is considered to be exponentially distributed for the sake of mathematical tractability and is made in the absence of studies characterizing the "real" distribution of the recovery process. Last, the authors assumed in their model an exponential distribution for the uptime durations of peers.

Duminuco, Biersack and En-Najjary [36] have proposed a proactive technique to reduce the maintenance cost, namely the bandwidth use, based on an on-going estimation of the departure rate of nodes that store blocks of data.

In [6], Bernard and Le Fessant have proposed a technique to estimate P2P backup systems reliability and optimize their performance while introducing a new criterion "peers' age or lifetime". The longer a peer has been in the system, the longer it is expected to stay in it. So, by carefully selecting the peers on which backup data is stored, repairing cost can be highly reduced while providing high durability level. The authors described a method to estimate the age of peers and they validate their method by simulation.

In [28], Dalle et al. have developed a stochastic model based on a fluid approximation to characterize the expectation and the standard deviation of the data lifetime in a P2P backup system while taking into account the fact that many data blocks are lost at the same time when a peer leaves permanently the system due to a disk failure. They do not consider churn and do not study data availability. They studied a system that never produces replicas as a result

of a transient failure, and hence when a disk crashes, it gets replaced by a new disk with no data. A recovery mechanism is then triggered for each block of data that has, after a failure, less than a predefined threshold. The recovery process tends to repair as fast as possible all the missing fragments whenever enough fragments of the considered block of data are available in the system. In addition, they considered that the system is much more affected by the fault of an old disk than a young one because it hosts probably more fragments as fragments of each block of data are uniformly distributed among peers.

1.4 Contribution and organization of this thesis

We address in this thesis the data lifetime and availability in distributed P2P backup and storage systems. In such systems, data are no longer stored on expensive magnetic tapes but on much cheaper hard disks. Although inexpensive, these storage systems pose many problems of reliability, confidentiality, availability, routing, performance, etc.

The goal of my thesis is to develop and evaluate mathematical models to characterize fundamental performance metrics of P2P backup and storage systems; data lifetime and availability. These systems use erasure codes redundancy mechanism to increase their reliability. Recall that replication is a special case of erasure code. A distributed repair mechanism is used in the first place and a centralized one in the second place to face the problems of nodes' permanent departure or failure or even the long transient disconnections when a high availability level is needed.

Our first contribution to the analysis of data durability and availability in P2P storage systems is [3]. In this work, simplifying assumptions have been considered while modeling the system. We have mainly assumed in [3] that both peer on-times durations and the recovery process are exponentially distributed, following the assumptions and results of [75, 11, 31]. Based on these assumptions, we evaluated data lifetime and data availability through a markovian analysis. Although the models are simple, they capture the behavior of both eager and lazy repair policies and both replication-based and erasure code-based systems, and accommodate both temporary and permanent disconnections of peers. For illustrative purpose and for having simple and explicit formulas, we introduced fluid approximations of the systems at hand to estimate the mean number of fragments available in each system. It is also possible to evaluate the performance of the P2P storage systems, that use regenerating codes [35] as a redundancy mechanism, using same models.

However, the recovery process in erasure-coded systems can differ from that in replicated systems. The implications of the exponential assumption on the recovery process are not the same in both systems. To understand how the recovery process could be better modeled, we implemented these process in the network simulator NS-2 [39]. The implementation details have

appeared in [31]. Then, we performed a simulation analysis of the download and the recovery processes in erasure-coded systems; cf. [32, 31]. We ran several experiments and collected a large number of samples of these processes in a large variety of scenarios. We used expectation maximization and least square estimation algorithms to fit the empirical distributions and tested the goodness of our fits using statistical (Kolmogorov-Smirnov test) and graphical methods. We found that the download time of a fragment of data located on a single peer follows approximately an exponential distribution in most (not in all) considered scenarios. We also found that the recovery time essentially follows a hypo-exponential distribution with many distinct phases for same scenarios. In some other scenarios, we found however that an exponential distribution can model the download and the recovery processes in a distributed P2P storage systems.

Building on the findings of [32], we developed in [30] markovian models to study data lifetime and availability in P2P storage systems assuming that the *fragment* download/upload time is exponentially distributed. The models in [30] are therefore more realistic than those in [3].

In light of the conclusions of [69], i.e., that machine availability is better modeled with a hyper-exponential distribution than with an exponential, Pareto, or Weibull distribution, we later extended in [29] and [33] the models of [3] and [30] respectively by assuming that peer on-times durations are hyper-exponentially distributed. Doing so, our modeling is valid under different distributed environments, cf. [26, 74, 69]. The work in [33] is under submission to IFIP WG 7.3 International Symposium on Computer Performance, Modeling, Measurements and Evaluation (Performance 2010).

Last, to overcome the limitation of scalability met in the packet-level simulator (NS-2), we propose and analyze an algorithm, that we called the "progressive-filling flow-level algorithm" or PFFLA. The algorithm is efficient in time and quite simple and uses the concept of "Progressive-Filling" (or max-min fairness), hence the name. The validation of this algorithm consists in characterizing the *distribution* of the response time of parallel downloads in a distributed storage system, through simulations. This is a joint work with Alain Jean-Marie (Research director at INRIA and LIRMM, CNRS/Université Montpellier 2). This last piece of work has been submited to the Third International Conference on Communication Theory, Reliability, and Quality of Service (CTRQ 2010).

The outline of the remainder of this thesis is as follows.

Chapter 2 presents a markovian analysis and a simple fluid approximation for P2P storage systems with a distributed repair mechanism under several assumptions on the peer availability and the recovery process. Also, it provides some numerical results to support the mathematical models. Similar to Chapter 2, Chapter 3 is devoted to the modeling of a P2P storage system with centralized repair mechanism through a markovian analysis and a fluid approximation.

In addition, we compare the performance of the two recovery mechanisms (centralized and distributed repair). In Chapter 4, we describe a packet-level simulation model and provide several experiments covering a large variety of scenarios while taking into consideration the impact of the heterogeneity of peers, the underlying network topologies, the propagation delays and the transport protocol. Chapter 5 introduces a simple, reliable and scalable "progressive-filling flow-level algorithm" or PFFLA, that characterizes the *distribution* of the response time of parallel downloads. PFFLA can be used as the core of any P2P simulator. Last, Chapter 6 concludes this thesis and discusses some open issues.

2

PERFORMANCE EVALUATION OF DATA LIFETIME AND AVAILABILITY IN DISTRIBUTED-REPAIR SYSTEMS

2.1 Introduction

In this chapter, we study the performance of distributed-repair P2P storage systems, in terms of data lifetime and availability through markovian models. These systems rely on data fragmentation and distributed storage. Files are partitioned into fixed-size blocks that are themselves partitioned into fragments. To ensure data reliability, redundant data is inserted in the system. Redundancy is achieved, in practice, either by replication or by using erasure codes. 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 systems need to compensate the loss of data by continuously storing additional redundant data onto new hosts. The distributed-repair scheme recovers one loss at a time.

The *lifetime* of data in the P2P system is a random variable (rv); we will investigate its distribution function. *Data availability* metrics refer to the amount of redundant fragments. We will consider two such metrics: the expected number of available redundant fragments, and the fraction of time during which the number of available redundant fragment exceeds a given threshold. The impact of each system parameter on the performance is evaluated. Guidelines are derived on how to engineer the system and tune its key parameters in order to provide desired lifetime and/or availability of data.

As we will show in Chapter 4 through simulations, the recovery process in distributed-repair systems can be modeled, in some known scenarios, by either an exponential or hypo-exponential distribution. This last assumption is nothing but a consequence of the finding that successive download durations of a fragment can be seen as iid rvs with a common exponential distribution function with parameter α, in addition to the assumption that concurrent fragments downloads are not correlated. Indeed, each of the recovery durations is the summation of s *independently* distributed exponential rvs having each its own rate [50]. We will see through simulations, in Chapter 4, that these concurrent downloads are weakly correlated in some interesting scenarios. The recovery process distribution essentially depends on the demand and capacities of the complete network entities: peers upload/download, routers and links, in addition to the volume of the background traffic.

Concerning peers availability, we will make two different assumptions as follow.

First, we simplify the study and assume that both peer on-times and off-times durations are exponentially distributed, i.e., they follow the assumptions and results of [75] on the peers availability as discussed in Section 1.3.

Second, in light of the conclusions of [69], i.e., that peers on-times duration are better modeled with a hyper-exponential distribution than with an exponential, Pareto, or Weibull distribution, the exponential assumption on the peers availability will therefore be relaxed later in this chapter, and it will be shown later on that a more precise, more realistic modeling is possible.

Therefore, we propose two simple models in which the peer availability is considered to follow an exponential distribution. The recovery process is considered to follow an exponential distribution in the first simple model, and a hypo-exponential distribution in the second simple model. We then extend the two simple models to more general ones by assuming that peers on-times durations, in both extended models, are hyper-exponentially distributed. Doing so, our modeling is general, realistic and valid under different distributed environments. A simple fluid model has been introduced under simple assumptions in order to have an explicit formula of the availability metric.

The rest of this chapter is organized as follows. Section 2.2 introduces system description and assumptions . In Section 2.3 we define some functions and introduce the notation. Sections 2.4 to 2.7 are devoted to the analysis of the distributed-repair P2P backup and storage systems, in terms of data lifetime and availability, through simple and extended markovian models as mentioned above. A simple fluid approximation is as well proposed in Section 2.4. In Section 2.8, we validate the approximation made to compute the availability metrics in the models thoughout simulation. Section 2.9 validates the simple fluid model made in Sect. 2.4.2. In Section 2.10, guidelines are derived on how to engineer the system and tune its key parameters, namely r and k, in order to provide desired lifetime and/or availability of data. Numerical

results that support the analysis and illustrate how to engineer the system are introduced in Section 2.11.

2.2 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 in which peers fully cooperate but randomly join and leave. The following assumptions on the P2P backup and storage system design will be enforced throughout the chapter:

- A single block of data D is partitioned into s equally sized fragments to which, using erasure codes (e.g. [78]), 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. Using our models, it is also possible to evaluate the performance of the systems that use regenerating codes [35] as a redundancy mechanism. As we have seen in Section 1.2.1, the difference between erasure codes and regererating codes is in the sizes of fragments and the required amount of data to be downloaded in order to recovery lost fragments. However, evaluating the complication cost of the implementation of the regenerating codes, with respect to their advantages, is left for future work.

- 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).

- 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 two different repair policies: the *eager* and the *lazy* repair policies. In the eager policy, a fragment of D is reconstructed as soon as it has become unavailable due to a peer disconnection. In the lazy policy, the repair is delayed until the number of unavailable fragments reaches a given threshold, denoted k. In the latter case, 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 distributed or in a centralized (studied in the next chapter) way. In the following description, we assume that the system misses k fragments so that lost fragments have to be restored.

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 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 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.

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 P2P backup and storage systems. 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. In fact, the two states of data already described appear when the recovery process is modeled by a hypo-exponential distribution as we will see later. 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 [75].

Assumption 2: (on-times) For the sake of simplification, we first assume that successive durations of on-times of a peer are iid rvs with a common exponential distribution function with parameter μ. To have more elaborate models, we second assume that these successive durations 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 [69]; when $n = 1$, it is in agreement with the analysis in [75]. According to Assumption 2 for $n > 1$, 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.

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: (recovery durations) We assume in the first place that successive recovery durations (download durations of the whole block or s fragments in parallel) are iid rvs with a common exponential distribution function with parameter γ.

This last assumption is supported by our findings in [31] as explains chapter 4. We will see in chapter 4 through experiments results how the recovery times distribution is impacted by the characteristics of the the peers and the considered overlay network.

Assumption 5: (download durations) We assume in the second place that successive download durations of a fragment, rather than the whole block of data as in Assumption 4, are iid rvs with a common exponential distribution function with parameter α. We further assume that concurrent fragments downloads are not correlated.

Assumption 5 is supported by our findings in [32, 31] as explains Chapter 4. The fragment download/upload time was found to follow approximately an exponential distribution and we

Table 2.1: 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)
γ	Recovery rate of a missing fragment when the recovery process is exponentially distributed

have found in simulations that these successive download durations are weakly correlated as long as the total workload is equally distributed over the active peers and the core network has a good connectivity and the peers upload/download capacities are asymmetric [85, 47].

Assumption 6: (recovery durations) A consequence of Assumption 5 is that each of the block download durations and then the recovery durations of a missing fragment in this distributed-repair scheme are iid rvs with a common hypo-exponential distribution function [50] with s phases. Indeed, each of these durations is the summation of s *independently* distributed exponential rvs.

It is worth mentioning that the simulation analysis of [32] 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. It was also found in [32] that a hypo-exponential model gives a more reasonable approximation of the recovery process than an exponential model even in cases when the null hypothesis is rejected for a good significant level in a scenario when the core networks has a good connectivity and the peers upstream and downstream bandwidths are asymmetric.

Given Assumptions 1–6, the models developed in this thesis are general, realistic and can capture the behaviors of the P2P storage systems in large variety of scenarios and environments. Table 2.1 recapitulates the parameters introduced in this chapter. 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 as the *network* parameter.

2.3 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 [5, 58]:

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

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

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

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

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

.

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:

$$R(i) = \frac{E[n_i]}{\sum_{l=1}^{n} E[n_l]} = \frac{\rho_i}{\sum_{l=1}^{n} \rho_l} = \frac{p_i/\mu_i}{\sum_{l=1}^{n} p_l/\mu_l} \tag{2.1}$$

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 "e" (resp. "h") will indicate that we are considering the recovery process to be modeled by an exponential (resp. a hyper-exponential) distribution; in the basic (resp. extended) models, we will add to the rvs a superscript "e" (resp. "h") referring to the assumption on the distribution of peers on-times:

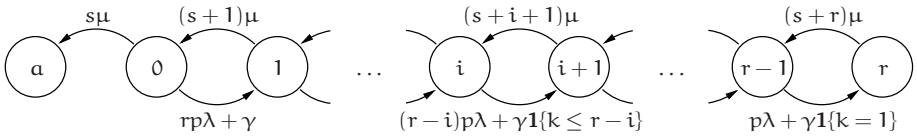


Figure 2.1: Transition rates of the basic absorbing Markov chain $\{X_e^e(t), t \geq 0\}$ in the distributed-recovery implementation.

exponential in the basic models and hyper-exponential in the extended models. 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 $\{\vec{B}(t),\ t \geq 0\}$.

2.4 Simple model, recovery process is exponentially distributed

In this section, we address the performance analysis of the P2P storage systems with the distributed implementation of the recovery process as described in Section 2.2. For the sake of simplification, we consider in this section that successive peers off-times (resp. on-times) durations and the recovery durations are exponentially distributed with parameters λ (resp. μ) and γ following Assumptions 1,3 and 4 in Section 2.2. We will focus on a single block of data and we will only pay attention to peers storing fragments of this block.

Let $X_e^e(t)$ be a $\{a, 0, 1, \ldots, r\}$-valued rv, where $X_e^e(t) = i \in \mathcal{T}_e^e := \{0, 1, \ldots, r\}$ indicates that $s + i$ fragments are available at time t, and $X_e^e(t) = a$ indicates that less than s fragments are available at time t. We assume that $X_e^e(0) \in \mathcal{T}_e^e$ so as to reflect the assumption that at least s fragments are available at $t = 0$. Thanks to the assumptions made in Section 2.2, it is easily seen that $\mathbf{X}_e^e := \{X_e^e(t),\ t \geq 0\}$ is an absorbing homogeneous Continuous-Time Markov Chain (CTMC) with transient states $0, 1, \ldots, r$ and with a single absorbing state a representing the situation when the block of data is lost. Non-zero transition rates of $\{X_e^e(t),\ t \geq 0\}$ are shown in Fig. 2.1.

2.4.1 Data lifetime

This section is devoted to the analysis of the data lifetime. Let $T_e^e(i) := \inf\{t \geq 0 : X_e^e(t) = a\}$ be the time until absorption in state a starting from $X_e^e(0) = i$, or equivalently the time at which the block of data is lost. In the following, $T_e^e(i)$ will be referred to as the *conditional block*

lifetime. We are interested in $P(T_e^e(i) < x)$, the probability distribution of the block lifetime given that $X_e^e(0) = i$ for $i \in \mathcal{T}_e^e$, and the expected time spent by the absorbing Markov chain in transient state j, given that $X_e^e(0) = i$. The infinitesimal generator has the following canonical form

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

where $\vec{R}_e^e$ is a non-zero column vector of size $|\mathcal{T}_e^e| = r + 1$, and $\vec{Q}_e^e$ is $|\mathcal{T}_e^e|$-by-$|\mathcal{T}_e^e|$ matrix. The elements of $\vec{R}_e^e$ are the transition rates between the transient states $x \in \mathcal{T}_e^e$ and the absorbing state a. The diagonal elements of $\vec{Q}_e^e$ are each the total transition rate out of the corresponding transient state. The other elements of $\vec{Q}_e^e$ are the transition rates between each pair of transient states. The only non-zero element of $\vec{R}_e^e$ in this simple model is $s\mu$ for $x = 0$. Let us proceed to the definition of the non-zero entries of $\vec{Q}_e^e$.

$$\begin{aligned} q_e^e(i, i-1) &= c_i \ , & i &= 1, 2, \ldots, r \ , \\ q_e^e(i, i+1) &= d_i + w_i \ , & i &= 0, 1, \ldots, r-1 \ , \\ q_e^e(i, i) &= -(c_i + d_i + w_i) \ , & i &= 0, 1, \ldots, r \ , \end{aligned}$$

with $w_i := \gamma \mathbb{1}\{i \leq r - k\}$, $c_i := (s + i)\mu$ and $d_i := (r - i)p\lambda$ for $i \in \mathcal{T}_e^e$.

From the theory of absorbing Markov chains we know that (e.g. [68, Lemma 2.2])

$$P(T_e^e(i) < x) = 1 - \vec{e}_{r+1}^{\,i+1} \cdot \exp(x\vec{Q}_e^e) \cdot \vec{1}_{r+1} \ , \quad x > 0 \ , i \in \mathcal{T}_e^e \ , \tag{2.2}$$

where $\vec{e}_{r+1}^{\,i+1}$ and $\vec{1}_{r+1}$ are vectors of dimension $r + 1$; all entries of $\vec{e}_{r+1}^{\,i+1}$ are null except the $(i + 1)$-th entry that is equal to 1, and all entries of $\vec{1}_{r+1}$ are equal to 1 (see Section 2.3 for all the definitions). In particular [68, p. 46]

$$E[T_e^e(i)] = -\vec{e}_{r+1}^{\,i+1} \cdot \left(\vec{Q}_e^e\right)^{-1} \cdot \vec{1}_{r+1} \ , \quad i \in \mathcal{T}_e^e \ , \tag{2.3}$$

where the existence of $\left(\vec{Q}_e^e\right)^{-1}$ is a consequence of the fact that all states in $\mathcal{T}_e^e$ are transient [68, p. 45]. Let $T_e^e(i, j) = \int_0^{T_e^e(i)} \mathbb{1}\{X_e^e(t) = j\}\, dt$ be the total time spent by the CTMC in transient state j given that $X_e^e(0) = i$. It can also be shown that [46]

$$E[T_e^e(i, j)] = -\vec{e}_{r+1}^{\,i+1} \cdot \left(\vec{Q}_e^e\right)^{-1} \cdot {}^t\vec{e}_{r+1}^{\,j+1} \ , \quad i, j \in \mathcal{T}_e^e \ . \tag{2.4}$$

where ${}^t\vec{y}$ denotes the transpose of any row vector $\vec{y}$. In other words, $E[T_e^e(i, j)]$ is the (i, j)-th entry of matrix $-\left(\vec{Q}_e^e\right)^{-1}$.

2.4.2 Data availability

In this section we introduce different metrics to quantify the availability of the block of data. The fraction of time spent by the absorbing Markov chain $\{X_e^e(t),\, t \geq 0\}$ in state j with $X_e^e(0) = i$ is $E[(1/T_e^e(i)) \int_0^{T_e^e(i)} \mathbb{1}\{X_e^e(t) = j\}\, dt]$. However, since it is difficult to find a closed-form expression for this quantity, we will instead use the following approximation

$$E\left[\frac{1}{T_e^e(i)} \int_0^{T_e^e(i)} \mathbb{1}\{X_e^e(t) = j\} dt\right] \approx \frac{E[T_e^e(i,j)]}{E[T_e^e(i)]}, \quad i,j \in \{0,\ldots,r\}, \tag{2.5}$$

We will validate this approximation through simulation in Section 2.8. With this in mind, we introduce

$$M_{e,1}^e(i) := \sum_{j=0}^{r} j \frac{E[T_e^e(i,j)]}{E[T_e^e(i)]} \quad , \quad M_{e,2}^e(i) := \sum_{j=m}^{r} \frac{E[T_e^e(i,j)]}{E[T_e^e(i)]} \quad , \quad i \in \mathcal{T} \ . \tag{2.6}$$

The first availability metric can be interpreted as the expected number of available redundant fragments during the block lifetime, given that $X_e^e(0) = i \in \mathcal{T}_e^e$. The second metric can be interpreted as the fraction of time when there are at least m redundant fragments during the block lifetime, given that $X_e^e(0) = i \in \mathcal{T}_e^e$. Both quantities $M_{e,1}^e(i)$ and $M_{e,2}^e(i)$ can be (numerically) computed from (2.3) and (2.4). Numerical results are reported in Section 2.11 for $i = r$ and $m = r - k$ in (2.6).

Continuous time Markov chain CTMC

Since it is difficult to come up with an explicit expression for either metric $M_{e,1}^e(i)$ or $M_{e,2}^e(i)$, we make the assumption that parameters k and r have been selected so that the time before absorption is "large". This can be formalized, for instance, by requesting that $P(T_e^e(r) > q) > 1 - \epsilon$, where parameters q and ϵ are set according to the particular storage application(s).

In this setting, one may represent the state of the storage system by a new irreducible and aperiodic—and therefore ergodic—Markov chain $\tilde{\mathbf{X}}_e^e := \{\tilde{X}_e^e(t),\, t \geq 0\}$ on the state-space $\mathcal{T}_e^e$ which is the same of the absorbing CTMC without the absorption sate considering that it can no longer be reached. Let $\tilde{\mathbf{Q}}_e^e = [\tilde{q}_e^e(i,j)]_{0 \leq i,j \leq r}$ be its infinitesimal generator. Matrices $\tilde{\vec{Q}}_e^e$ and $\vec{Q}_e^e$, whose non-zero entries are given in Section 2.4.1, are identical except for $\tilde{q}_e^e(0,0) = -(d_0 + w_0)$. More precisely, $\tilde{\mathbf{X}}_e^e$ becomes a birth and death process (see Fig. 2.1). Let $\pi(i)$ be the stationary probability that $\tilde{\mathbf{X}}_e^e$ is in state i, then (e.g. [56])

$$\pi(i) \;=\; \left[1 + \sum_{i=1}^{r} \prod_{j=0}^{i-1} \frac{d_j + w_j}{c_{j+1}}\right]^{-1} \cdot \prod_{j=0}^{i-1} \frac{d_j + w_j}{c_{j+1}}, \quad i \in \mathcal{T}^e. \tag{2.7}$$

From (2.7) we can derive the expected number of available redundant fragments through the formula:

$$E[\tilde{X}_e^e] = \sum_{i=0}^{r} i\pi(i). \tag{2.8}$$

Simple fluid model

In order to have an expression for $E[\tilde{X}_e^e]$ more explicit than (2.8) we will use the idea of the fluid approximation using [60, Thm. 3.1] for the case of the eager policy $k = 1$.

As $\tilde{X}_e^e(t)$ represents the number of available redundant fragments in the system. Thus, $r^{-1}.\tilde{X}_e^e(t)$ would be the proportion of available redundant fragments. We must find a continuous function $f(x, l)$, where $x \in [0, 1]$ and l is an integer component such that the infinitesimal parameters corresponding to $\tilde{X}_e^e(t)$ are given by

$$q_{i,i+l} = rf(i/r, l), \quad l \neq 0$$

The infinitesimal parameters for the CTMC, as we saw in the previous section, are given by:

$$\begin{aligned} q_{i,i+1} &= (r-i)p\lambda + \gamma, \quad 0 \leq i \leq r-1 \\ q_{i,i-1} &= (s+i)\mu, \quad 1 \leq i \leq r \end{aligned}$$

These transitions can be rewritten as

$$\begin{aligned} q_{i,i+1} &= r.f(i/r, 1) \\ q_{i,i-1} &= r.f(i/r, -1) \end{aligned}$$

where

$$\begin{aligned} f(x, 1) &= (1-x)p\lambda + \frac{\gamma}{r} \\ f(x, -1) &= (\frac{s}{r} + x)\mu \end{aligned}$$

Introduce the function $F(x) := \sum_l l \times f(x, l)$. Then

$$F(x) = (1-x)p\lambda + \frac{\gamma}{r} - (\frac{s}{r} + x)\mu$$

This function is continuous and verifies the conditions of Theorem 3.1 in [60]. The Process $\frac{\tilde{X}_e^e(t)}{r}$ converges then in distribution to the Process $\xi(t)$ solution of this ordinary differential equation (ODE)

$$\frac{d\xi(t)}{dt} = F(\xi(t)) = -\xi(t)(p\lambda + \mu) + p\lambda + \frac{\gamma - s\mu}{r}$$

At stationarity, the solution of this ODE is the expected proportion of available redundant fragments in the system. In other words,

$$E[\xi] = \frac{p\lambda + \frac{\gamma - s\mu}{r}}{p\lambda + \mu} \tag{2.9}$$

The expected number of available redundant fragments is thus approximately

$$E[\tilde{X}_e^e(t)] \approx \frac{rp\lambda + \gamma - s\mu}{p\lambda + \mu} \tag{2.10}$$

The expected number of the available fragments is thus approximately

$$E[\tilde{X}_e^e(t)] + s \approx \frac{(s+r)p\lambda + \gamma}{p\lambda + \mu} \tag{2.11}$$

Numerical results for $E[\tilde{X}_e^e]$, or more precisely, for its deviation from $M_{e,1}^e(r)$ are reported in Section 2.9. We will show that the fluid model converges very well when value of r increases relatively to s or to the ratio r/s.

2.5 Simple model, recovery process is hypo-exponentially distributed

Similarly to Section 2.4, we address in this section the performance analysis of the distributed implementation of the P2P storage systems while considering Assumptions 1,3,5 and 6 introduced in Section 2.2. In other words, we consider that successive peers off-times (resp. on-times) durations are exponentially distributed with parameters λ (resp. μ), and we assume that successive download durations of a fragment are iid rvs with a common exponential distribution function with parameter α. We further assume that concurrent fragments downloads are not correlated. Therefore, the recovery processes is a rv following a hypo-exponential distribution of s phases [50] having each its own rate. We will focus on a single block of data and we will only pay attention to peers storing fragments of this block.

Let $X_h^e(t)$ and $Y_h^e(t)$ be two rvs denoting respectively the number of fragments in the system that are available for download and the state of the recovery process. Recall that the recovery process consists of a series of s exponential distributions that can be seen as s stages and that the distributed scheme repairs only one fragment at a time. We denote $Y_h^e(t) = j$ ($j = 0, 1, \ldots, s-1$) to express that j exponential rvs have been realized at time t, so that $s - j$ are still to go. When the last stage is completed, the recovery process is completed and $Y_h^e(t) = 0$. The process $X_h^e(t)$ takes value in the set $\{s-1, s, \ldots, s+r\}$.

Consider now the joint process $(X_h^e(t), Y_h^e(t))$. When $X_h^e(t) \geq s$, data D is *available*, regardless of $Y_h^e(t)$. When $X_h^e(t) < s$ (in particular $s-1$) but s different fragments can be located between the secure agent involved in the recovery process and the available fragments in the

system, namely $X_h^e(t) + Y_h^e(t) \geq s$, D is *unavailable* but still alive. When $X_h^e(t) + Y_h^e(t) < s$ or $X_h^e(t) + Y_h^e(t) \geq s$ but less than s distinct fragments are still accessible by the system, D is *lost*. The latter situation will be modeled by a single state a. Introduce the set

$$\begin{aligned}
\mathcal{T}_h^e := \{ \quad & \left.\begin{array}{l}(s-1,1),(s-1,2),\ldots,(s-1,s-1),\end{array}\right\} \text{ D is unavailable} \\
& \left.\begin{array}{l}(s,0),(s,1),\ldots,(s,s-1),\\ (s+1,0),(s+1,1),\ldots,(s+1,s-1),\ldots,\\ (s+r-1,0),(s+r-1,1),\ldots,(s+r-1,s-1),\\ (s+r,0)\ \}\end{array}\right\} \text{ D is available} \\
|\mathcal{T}_h^e| = s(r+1).&
\end{aligned}$$

Thanks to the considered assumptions, it is easily seen that the two-dimensional process $\{(X_h^e(t), Y_h^e(t)),\ t \geq 0\}$ is an absorbing homogeneous Continuous-Time Markov Chain (CTMC) with transient states the elements of $\mathcal{T}_h^e$ and with a single absorbing state a representing the situation when D is lost. Without loss of generality, we assume that $X_h^e(0) \geq s$. The infinitesimal generator has the following canonical form

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

The analysis of the absorbing Markov chain $\{(X_h^e(t), Y_h^e(t)) : t \geq 0\}$ that takes value in $\mathcal{T}_h^e \cup \{a\}$ is similar to the analysis in Section 2.4, we will then only sketch it. In particular, $\vec{R}_h^e$ and $\vec{Q}_h^e$ have similar definitions as $\vec{R}_e^e$ and $\vec{Q}_e^e$ after replacing the subscript "e" with the subscript "h" whenever needed. The elements of $\vec{R}_h^e$ are lexicographically ordered alike the order in $\mathcal{T}_h^e$. The non-zero elements of $\vec{R}_h^e$ and $\vec{Q}_h^e$ are as follows

$$\begin{aligned}
& r_h^e(s-1,j) = (s-1)\mu, \text{ for } j = 1,\ldots,s-1; \quad r_h^e(s,j) = (s-j)\mu, \text{ for } j = 0,\ldots,s-1. \\
& q_h^e((i,j),(i-1,j)) = \begin{cases} j\mu, & \text{for} \quad i = s, j = 1,\ldots,s-1; \\ i\mu, & \text{for} \quad i = s+1,\ldots,s+r-1, j = 0,\ldots,s-1; \\ & \text{or} \quad i = s+r, j = 0. \end{cases} \\
& q_h^e((i,j),(i,j+1)) = (s-j)\alpha, \quad \text{for} \quad i = s,\ldots,s+r-k, j = 0; \\
& \qquad\qquad \text{or} \quad i = s-1,\ldots,s+r-1, j = 1,\ldots,s-2. \\
& q_h^e((i,s-1),(i+1,0)) = \alpha, \quad \text{for} \quad i = s-1,\ldots,s+r-2. \\
& q_h^e((i,j),(i+1,j)) = (s+r-i)\lambda p, \quad \text{for} \quad i = s-1, j = 1,\ldots,s-1; \\
& \qquad\qquad \text{or} \quad i = s,\ldots,s+r-2, j = 0,\ldots,s-1. \\
& q_h^e((s+r-1,j),(s+r,0)) = \lambda p + \mathbb{1}\{j = s-1\}\alpha, \quad \text{for} \quad j = 0,\ldots,s-1. \\
& q_h^e((i,j),(i,j)) = -r_h^e(i,j) - \textstyle\sum_{(i',j') \in \mathcal{T}_h^e - \{(i,j)\}} q_h^e((i,j),(i',j')), \quad \text{for } (i,j) \in \mathcal{T}_h^e.
\end{aligned}$$

For illustration purposes, we depict in Fig. 2.2 an example of the absorbing CTMC with its non-zero transition rates when $s = 3$, $r = 2$, and $k = 2$.

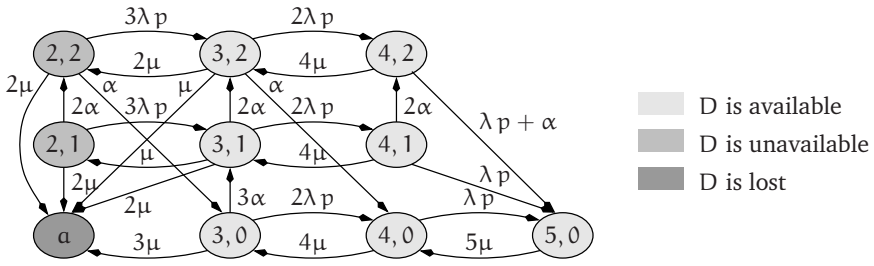


Figure 2.2: The Markov chain $\{(X_h^e(t), Y_h^e(t)),\ t \geq 0\}$ with the distributed-repair scheme when $s = 3$, $r = 2$, and $k = 2$.

2.5.1 Data lifetime

Similar to what was done in Section 2.4.1, this section is devoted to the analysis of the lifetime of D. Let $T_h^e(i,j) := \inf\{t > 0 : (X_h^e(t), Y_h^e(t)) = a | (X_h^e(0), Y_h^e(0)) = (i,j)\}$ be the time until absorption in state a, or equivalently the time until D is lost, given that the initial state of D is (i,j). In the following, $T_h^e(i,j)$ will be referred to as the *conditional block lifetime*. We are interested in $P(T_h^e(i,j) \leq x)$ and $E[T_h^e(i,j)]$, respectively the probability distribution of the conditional block lifetime and its expectation, given that $(X_h^e(0), Y_h^e(0)) = (i,j) \in \mathcal{T}_h^e$. From the theory of absorbing Markov chains, we know that (e.g. [68, Lemma 2.2])

$$P(T_h^e(i,j) \leq x) = 1 - \vec{e}_{|\mathcal{T}_h^e|}^{\,\mathrm{ind}(i,j)} \cdot \exp\left(x\vec{Q}_h^e\right) \cdot \vec{1}_{|\mathcal{T}_h^e|}, \quad x > 0,\ (i,j) \in \mathcal{T}_h^e \tag{2.12}$$

where $\mathrm{ind}(i,j)$ refers to the index of the state $(i,j) \in \mathcal{T}_h^e$ in the matrix $\vec{Q}_h^e$. Recall that the elements of $\vec{Q}_h^e$ are numbered according to the lexicographic order. Definitions of vectors $\vec{e}_i^{\,j}$ and $\vec{1}_i$ are given at the end of Section 2.3. Observe that the term $\vec{e}_{|\mathcal{T}_h^e|}^{\,\mathrm{ind}(i,j)} \cdot \exp\left(x\vec{Q}_h^e\right) \cdot \vec{1}_{|\mathcal{T}_h^e|}$ in the r.h.s. of (2.12) is nothing but the summation of all $|\mathcal{T}_h^e|$ elements in row $\mathrm{ind}(i,j)$ of matrix $\exp\left(x\vec{Q}_h^e\right)$.

We know from [68, p. 46] that the expected time until absorption can be written as

$$E\left[T_h^e(i,j)\right] = -\vec{e}_{|\mathcal{T}_h^e|}^{\,\mathrm{ind}(i,j)} \cdot \left(\vec{Q}_h^e\right)^{-1} \cdot \vec{1}_{|\mathcal{T}_h^e|}, \quad (i,j) \in \mathcal{T}_h^e, \tag{2.13}$$

where the existence of $\left(\vec{Q}_h^e\right)^{-1}$ is a consequence of the fact that all states in $\mathcal{T}_h^e$ are transient [68, p. 45]. Inverting $\vec{Q}_h^e$ analytically can rapidly become cumbersome as s or r increases. We will instead perform numerical computations as reported in Section 2.11.

Consider now

$$T_h^e((i,j),(i',j')) := \int_0^{T_h^e(i,j)} \mathbb{1}\left\{(X_h^e(t), Y_h^e(t)) = (i',j')\right\} dt$$

that is the total time spent by the CTMC in transient state (i', j') given that $\{X_h^e(0), Y_h^e(0)\} = (i, j)$. It can also be shown that [46, p. 419]

$$\mathrm{E}\left[T_h^e((i,j),(i',j'))\right] = -\vec{e}_{|\mathcal{T}_h^e|}^{\mathrm{ind}(i,j)} \cdot \left(\vec{Q}_h^e\right)^{-1} \cdot {}^t\vec{e}_{|\mathcal{T}_h^e|}^{\mathrm{ind}(i',j')}, \quad (i,j),(i',j') \in \mathcal{T}_h^e, \tag{2.14}$$

where ${}^t\vec{y}$ denotes the transpose of a given vector $\vec{y}$. In other words, the expectation $\mathrm{E}\left[T_h^e((i,j),(i',j'))\right]$ is the entry of matrix $\left(-\vec{Q}_h^e\right)^{-1}$ at row $\mathrm{ind}(i,j)$ and column $\mathrm{ind}(i',j')$.

2.5.2 Data availability

In this section we introduce different metrics to quantify the availability of D. We are interested in the fraction of time spent by the CTMC in any given state (i', j') before absorption. However, this quantity is difficult to find in closed-form. Therefore, we resort to using the following approximation

$$\mathrm{E}\left[\frac{T_h^e((i,j),(i',j'))}{T_h^e(i,j)}\right] \approx \frac{\mathrm{E}[T_h^e((i,j),(i',j'))]}{\mathrm{E}[T_h^e(i,j)]}. \tag{2.15}$$

Here, (i, j) is the state of D at $t = 0$. This approximation have been validated through simulations, as shown later in Section 2.8. With this approximation in mind, we introduce two availability metrics: the first can be interpreted as the expected number of fragments of D that are in the system during the lifetime of D; the second can be interpreted as the fraction of time when at least m fragments are in the system during the lifetime of D. More formally, given that $(X_h^e(0), Y_h^e(0)) = (i, j) \in \mathcal{T}_h^e$, we define

$$M_{h,1}^e(i,j) := \sum_{(i',j') \in \mathcal{T}_h^e} i' \frac{\mathrm{E}[T_h^e((i,j),(i',j'))]}{\mathrm{E}[T_h^e(i,j)]}, \tag{2.16}$$

$$M_{h,2}^e((i,j),m) := \sum_{(i',j') \in \mathcal{T}_h^e, i' \geq m} \frac{\mathrm{E}[T_h^e((i,j),(i',j'))]}{\mathrm{E}[T_h^e(i,j)]}. \tag{2.17}$$

For instance, $M_{h,2}^e((s+r,0),s)$ is the proportion of time when data D is *available* for users, given that $s + r$ fragments of D are initially available for download.

Similarly to was done in Section 2.4.2, we can assume that the parameters r and k are tuned such that the time before absorption in state a is arbitrarily long, and then compute the expected number of available fragments in the system through an ergodic Markov chain. However, since the formula is not explicit, such an evaluation would be useless because we have already closed-forms of the availability metrics. Introducing a fluid approximation as what was done in Section 2.4.2 under the assumptions made in this Section is a complex task.

Let us proceed now to the extended models where the peers on-times are hyper-exponentially distributed, with n phases.

2.6 Extended model, recovery process is exponentially distributed

We consider in this section that peers on-times are hyper-exponentially distributed, with n phases (Assumption 2 for $n > 1$ in Section 2.2); 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 for $i = 1, \ldots, n$. We naturally have $\sum_{i=1}^{n} p_i = 1$. Recall that the hyper-exponential distribution is a mixture or weighted sum of exponentials, with density function equal to $\sum_{i=0}^{n} p_i \mu_i \exp(-\mu_i x)$. According to this assumption, 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 $i \in [1..n]$. We also consider that the recovery process is modeled by an exponential distribution (Assumption 4 in Section 2.2). This model is a generalization of the basic model introduced in Section 2.4, since setting $n = 1$ (then $p_1 = 1$) and $\mu_1 = \mu$ returns the basic model.

The system state-space, unlike the basic model, will have to include knowledge of the peers on-times phases to be able to incorporate hyper-exponential distribution into a Markov chain model, where the Markov property must hold [98, p. 266]. It is no longer sufficient to consider solely the number of available *redundant* fragments of D as was done in Section 2.4.

Let us consider a n-tuple $\vec{i} = (i_1, \ldots, i_n)$ with $i_l \in \{0, \ldots, s+r\}$ and a function $S(\vec{i}) := \sum_{l=1}^{n} i_l$ as already defined in the end of Section 2.3. It will be convenient for later use to introduce sets $\mathcal{L}_I := \{\vec{i} \in \{0, \ldots, s+r\}^n, S(\vec{i}) = I\}$ for $I = s, \ldots, s+r$. The set $\mathcal{L}_I$ consists of all system states in which the number of fragments of D currently available is equal to I. For any I, the cardinal of $\mathcal{L}_I$ 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).
Let $X_e^h(t)$ represent the system state at time t. The rv $X_e^h(t)$ takes value in $\{a\} \cup \mathcal{T}_e^h$ where $\mathcal{T}_e^h := \bigcup_{I=s}^{s+r} \mathcal{L}_I$. The identity $X_e^h(t) = a$ indicates that less than s fragments of D are available at time t, and $X_e^h(t) = \vec{i} = (i_1, \ldots, i_n)$ indicates that $i_l \in \{0, \ldots, s+r\}$ fragments of D are stored on a peer in phase l for $l \in \{1, \ldots, n\}$, such that the total number of available fragments $S(\vec{i})$ lies between s and $s+r$.

The process $\mathbf{X}_e^h := \{X_e^h(t), t \geq 0\}$ is an absorbing Markov chain, with a single absorbing state a and $|\mathcal{T}_e^h| = \sum_{I=s}^{s+r} \binom{I+n-1}{n-1}$ transient states.

2.6.1 Data lifetime

Introduce $T_e^h(\mathcal{L}_I) := \inf\{t \geq 0 : X_e^h(t) = a | X_e^h(0) \in \mathcal{L}_I\}$, the time until absorption in state a given that the initial number of fragments of D available in the system is equal to I. In this section, we will derive the probability distribution and the expectation of $T_e^h(\mathcal{L}_I)$.

Let $\vec{Q}_e^h$ and $\vec{R}_e^h$ have similar definitions to those $\vec{Q}_e^e$ and $\vec{R}_e^e$ which are defined in Section 2.4.1 after replacing the superscript "e" with "h" whenever needed. Non-zero elements of $\vec{R}_e^h$ are

given by $r_e^h(\vec{i}) = \sum_{l=1}^n i_l\mu_l$ for $\vec{i} \in \mathcal{L}_s$. Introduce for $\vec{i}, \vec{j} \in \mathcal{T}_e^h$ and $l = 1, \ldots, n$.

$$\begin{aligned}
A_l &:= i_l\mu_l \mathbb{1}\{1 \le i_l \le s + r\},\\
B_{\vec{i},l} &:= p_l(s + r - S(\vec{i}))p\lambda \mathbb{1}\{0 \le i_l \le s + r - 1\},\\
D_{\vec{i},l} &:= R(l)\gamma \mathbb{1}\{S(\vec{i}) \le s + r - k\},
\end{aligned}$$

Where the definition of $R(l)$, the probability of selecting a *new* peer in phase l or equivalently the percentage of the connected peers in phase l, was intoduced in Section 2.3 ($R(l) = \frac{p_l/\mu_l}{\sum_{i=1}^n p_i/\mu_i}$). Non-zero elements of $\vec{Q}_e^h = [q_e^h(\vec{i}, \vec{j})]_{\vec{i},\vec{j}\in\mathcal{T}_e^h}$ are

$$\left.\begin{aligned}
q_e^h\left(\vec{i}, \vec{i} - \vec{e}_n^l\right) &= A_l, && s + 1 \le S(\vec{i}) \le s + r,\\
& && 1 \le i_l \le s + r,\\
q_e^h\left(\vec{i}, \vec{i} + \vec{e}_n^l\right) &= B_{\vec{i},l} + D_{\vec{i},l}, && s \le S(\vec{i}) \le s + r - 1,
\end{aligned}\right\} \text{ for } l = 1, \ldots, n,$$

$$q_e^h(\vec{i}, \vec{i}) = -\sum_{l=1}^n (A_l + B_{\vec{i},l} + D_{\vec{i},l}), \qquad s \le S(\vec{i}) \le s + r, \tag{2.18}$$

Similarly to (2.2) and (2.12), we can write

$$P\left(T_e^h(\{\vec{i}\}) \le x\right) = 1 - \vec{e}_{|\mathcal{T}_e^h|}^{\,\mathrm{ind}(\vec{i})} \cdot \exp\left(x\vec{Q}_e^h\right) \cdot \vec{1}_{|\mathcal{T}_e^h|},\ x > 0,\ \vec{i} \in \mathcal{T}_e^h, \tag{2.19}$$

where $\mathrm{ind}(\vec{i})$ refers to the index of state $\vec{i}$ in matrix $\vec{Q}_e^h$ and $T_e^h(\{\vec{i}\})$ is the time until absorption in state a given that the system is in state $\vec{i}$ at time 0. Let $\pi_{\vec{i}}$ denote the probability that the system starts in state $\vec{i} \in \mathcal{L}_I$ at time 0 given that $X_e^h(0) \in \mathcal{L}_I$. We can write

$$\pi_{\vec{i}} := P\left(X_e^h(0) = \vec{i} \in \mathcal{L}_I | X_e^h(0) \in \mathcal{L}_I\right) = \binom{I}{i_1, i_2, \ldots, i_n} \prod_{l=1}^n R(l)^{i_l}. \tag{2.20}$$

Clearly $\sum_{\vec{i}\in\mathcal{L}_I} \pi_{\vec{i}} = 1$ for $I = s, \ldots, s + r$. Using (2.19) and (2.20) and the total probability theorem yields

$$\begin{aligned}
P\left(T_e^h(\mathcal{L}_I) \le x\right) &= \sum_{\vec{i}\in\mathcal{L}_I} P\left(T_e^h(\{\vec{i}\}) \le x\right) \pi_{\vec{i}} && (2.21)\\
&= 1 - \sum_{\vec{i}\in\mathcal{L}_I} \pi_{\vec{i}} \vec{e}_{|\mathcal{T}_e^h|}^{\,\mathrm{ind}(\vec{i})} \cdot \exp\left(x\vec{Q}_e^h\right) \cdot \vec{1}_{|\mathcal{T}_e^h|},\ x > 0,\ \vec{i} \in \mathcal{T}_e^h. && (2.22)
\end{aligned}$$

The expectation of the block lifetime when there are initially I fragments available in the system is given by

$$E\left[T_e^h(\mathcal{L}_I)\right] = -\sum_{\vec{i}\in\mathcal{L}_I} \pi_{\vec{i}} \vec{e}_{|\mathcal{T}_e^h|}^{\,\mathrm{ind}(\vec{i})} \cdot \left(\vec{Q}_e^h\right)^{-1} \cdot \vec{1}_{|\mathcal{T}_e^h|},\ I = s, \ldots, s + r. \tag{2.23}$$

Similarly to (2.4) and (2.14), we can compute

$$\begin{aligned} \mathrm{E}\left[\mathrm{T}_e^h(\mathcal{L}_I,\mathcal{L}_J)\right] &= \sum_{\vec{j}\in\mathcal{E}_J} \mathrm{E}\left[\mathrm{T}_e^h(\mathcal{L}_I,\{\vec{j}\})\right] \\ &= -\sum_{\vec{i}\in\mathcal{L}_I}\sum_{\vec{j}\in\mathcal{E}_J} \pi_{\vec{i}}\vec{e}^{\,\mathrm{ind}(\vec{i})}_{|\mathcal{T}_e^h|}\cdot\left(\vec{Q}_e^h\right)^{-1}\cdot\vec{e}^{\,\mathrm{ind}(\vec{j})}_{|\mathcal{T}_e^h|},\ s\le I,J\le s+r, \end{aligned} \tag{2.24}$$

where $\mathrm{T}_e^h(\mathcal{L}_I,\mathcal{L}_J)$ is the total time spent in transient states $\vec{j}\in\mathcal{L}_J$ given that $X_e^h(0)\in\mathcal{L}_I$. In other words, $\mathrm{T}_e^h(\mathcal{L}_I,\mathcal{L}_J)$ represents the time during which J fragments of D are available given that there were I fragments of D in the system at time 0.

2.6.2 Data availability

The data availability are quantified, as motivated in Section 2.4.2 and Section 2.5.2, by the following two metrics (for $s\le I\le s+r$).

$$M_{e,1}^h(\mathcal{L}_I) := \sum_{J=s}^{s+r} J\frac{\mathrm{E}\left[\mathrm{T}_e^h(\mathcal{L}_I,\mathcal{L}_J)\right]}{\mathrm{E}\left[\mathrm{T}_e^h(\mathcal{L}_I)\right]},\quad M_{e,2}^h(\mathcal{L}_I) := \sum_{J=m}^{s+r}\frac{\mathrm{E}\left[\mathrm{T}_e^h(\mathcal{L}_I,\mathcal{L}_J)\right]}{\mathrm{E}\left[\mathrm{T}_e^h(\mathcal{L}_I)\right]}. \tag{2.25}$$

after using the following approximation.

$$\mathrm{E}\left[\frac{\mathrm{T}_e^h(\mathcal{L}_I,\mathcal{L}_J)}{\mathrm{T}_e^h(\mathcal{L}_I)}\right]\approx\frac{\mathrm{E}[\mathrm{T}_e^h(\mathcal{L}_I,\mathcal{L}_J)]}{\mathrm{E}[\mathrm{T}_e^h(\mathcal{L}_I)]}, \tag{2.26}$$

The first availability metric can be interpreted as the expected number of available fragments during the block lifetime, given that the initial number of fragments at time $t=0$ is I. The second metric can be interpreted as the fraction of time when there are at least m fragments during the block lifetime, given that the initial number of fragments at time $t=0$ is I. Both quantities can be numerically computed.

2.7 Extended model, recovery process is hypo-exponentially distributed

In this section, we consider that the peers on-times are hyper-exponentially distributed, with n phases (Assumption 2 for $n>1$ in Section 2.2); 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 for $i=1,\ldots,n$. We naturally have $\sum_{i=1}^n p_i=1$. As mentioned previously, according to this assumption, 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 $i\in[1..n]$. The probability of selecting an *active* peer in phase i, denoted by $R(i)=\dfrac{p_i/\mu_i}{\sum_{l=1}^n p_l/\mu_l}$, is equal to the percentage of the connected

peers in phase i. We also consider that the recovery process is modeled by a hypo-exponential distribution (Assumption 5 and 6 in Section 2.2). In fact, this section introduces a general model that is applicable to many realistic distributed environments and scenarios as justified in Chapter 4.

The system at time t can be described under the considered assumptions by both the number of fragments that are available for download and the state of the recovery process. Unlike the centralized-repair scheme, 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 $\vec{X}_h^h(t)$, $\vec{Y}_h^h(t)$, $\vec{Z}_h^h(t)$, where n is the number of phases of the hyper-exponential distribution of peers on-times durations, and a $3n$-dimensional vector $\vec{W}(t) = (\vec{X}_h^h(t), \vec{Y}_h^h(t), \vec{Z}_h^h(t))$. Vectors $\vec{Y}_h^h(t)$ and $\vec{Z}_h^h(t)$ describe the recovery process. The formal definition of these vectors is as follows:

- $\vec{X}_h^h(t) := (X_{h,1}^h(t), \ldots, X_{h,n}^h(t))$ where $X_{h,l}^h(t)$ 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. $\vec{X}_h^h(t)$ must verify $S(\vec{X}_h^h(t)) \in [s-1..s+r]$.
- $\vec{Y}_h^h(t) := (Y_{h,1}^h(t), \ldots, Y_{h,n}^h(t))$ where $Y_{h,l}^h(t)$ 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).
- $\vec{Z}_h^h(t) := (Z_{h,1}^h(t), \ldots, Z_{h,n}^h(t))$ where $Z_{h,l}^h(t)$ 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.

Given the above definitions, we necessarily have $Y_{h,l}^h(t) \leq X_{h,l}^h(t)$ 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 $S(\vec{X}_h^h(t))$ (recall the definition of the function S in Section 2.3). During the recovery process, $S(\vec{Y}_h^h(t)) + S(\vec{Z}_h^h(t)) = s$, such that $S(\vec{Y}_h^h(t)), S(\vec{Z}_h^h(t)) \in [1..s-1]$. Because the distributed-recovery scheme repairs fragments only one at a time, we have $S(\vec{X}_h^h(t)) \in [s-1..s+r]$. The end of the download phase is also the end of the recovery process. We will then have $\vec{Y}_h^h(t) = \vec{Z}_h^h(t) = \vec{0}$ until the recovery process is again triggered.

According to the terminology introduced in Section 2.2, at time t, data D is *available* if $S(\vec{X}_h^h(t)) \geq s$, regardless of the state of the recovery process. It is *unavailable* if $S(\vec{X}_h^h(t)) < s$ but $S(\vec{Z}_h^h(t))$—the number of fragments hold by the secure agent—is larger than $s - S(\vec{X}_h^h(t))$ *and* at least $s - S(\vec{X}_h^h(t))$ fragments out of $S(\vec{Z}_h^h(t))$ are different from those $S(\vec{X}_h^h(t))$ 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 secure agent—may be unknown due to peers churn. However, we are able to find a lower bound on it, namely,

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

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 $b(\vec{X}_h^h(t), \vec{0}, \vec{0}) = S(\vec{X}_h^h(t))$ gives the exact number of distinct fragments.

According to the description and assumptions listed in Section 2.2, the state of data D at time t can be represented by $\vec{W}(t)$ and the multi-dimensional process $\vec{\mathbf{W}} := \{\vec{W}(t),\ t \geq 0\}$ is an absorbing homogeneous CTMC with a single absorbing state a. The set of transient states $\mathcal{T}_h^h$ 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 matrix $\vec{R}_h^h$ and $\vec{Q}_h^h$ have similar definitions as $\vec{R}_e^e$ and $\vec{Q}_e^e$ after replacing the subscript and the superscript "e" with "h" whenever needed. The non-zero elements of $\vec{R}_h^h$ are, for $S(\vec{y}_h^h) \in [1..s-1]$ and $S(\vec{z}_h^h) = s - S(\vec{y}_h^h)$,

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

$$r_h^h(\vec{x}_h^h, \vec{y}_h^h, \vec{z}_h^h) = \sum_{l=1}^{n} x_{h,l}^h \mu_l, \qquad \text{for} \quad S(\vec{x}_h^h) = s - 1.$$

$$r_h^h(\vec{x}_h^h, \vec{y}_h^h, \vec{z}_h^h) = \sum_{l=1}^{n} y_{h,l}^h \mu_l \cdot \mathbb{1}\Big\{b(\vec{x}_h^h, \vec{y}_h^h, \vec{z}_h^h) = s\Big\}, \qquad \text{for} \quad S(\vec{x}_h^h) = s.$$

We next write the non-zero elements of $\vec{Q}_h^h$. Let us drop **hereafter and until the end of this section** the subscript h and the superscript h from the random variables and metrics to simplify the readability of the equations.

The case when a peer leaves the system

There are three 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 the other two situations, the recovery process is ongoing and at least one download is completed. 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), which is not the case of the third situation, where the secure agent must start downloading a fragment from another available peer that is uniformly

selected among all available peers not currently involved in the recovery process. The elements of $\vec{Q}$ corresponding to these three situations are, for $l \in [1..n]$, $m \in [1..n]$, $S(\vec{y}) \in [1..s-1]$ and $S(\vec{z}) = s - S(\vec{y})$,

$$\begin{aligned}
&q((\vec{x},\vec{0},\vec{0}),(\vec{x}-\vec{e}_n^{\,l},\vec{0},\vec{0})) = x_l\mu_l,\\
&\qquad \text{for} \quad S(\vec{x}) \in [s+1..s+r].\\
&q((\vec{x},\vec{y},\vec{z}),(\vec{x}-\vec{e}_n^{\,l},\vec{y},\vec{z})) = [x_l - y_l]^+\mu_l,\\
&\qquad \text{for} \quad S(\vec{x}) \in [s..s+r-1].\\
&q((\vec{x},\vec{y},\vec{z}),(\vec{x}-\vec{e}_n^{\,l},\vec{y}-\vec{e}_n^{\,l}+\vec{e}_n^{\,m},\vec{z})) = \frac{y_l\mu_l[x_m - y_m - z_m]^+}{\sum_{i=1}^n [x_i - y_i - z_i]^+},\\
&\qquad \text{for} \quad S(\vec{x}) \in [s..s+r-1].
\end{aligned}$$

The case when a peer rejoins the system

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

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

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. In fact, when a recovery process is initiated, the system state verifies $S(\vec{x}) \in [s..s+r-k]$ and $\vec{y} = \vec{z} = \vec{0}$. The secure agent on the new peer selects s peers out of the $S(\vec{x})$ 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 $0 \le i_l \le x_l$, for $l \in [1..n]$, and $S(\vec{i}) = s$. This selection occurs with probability

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

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 2.3). Similarly, when the number of

ongoing downloads is $\vec{y}$, the probability that the first download to be completed out of $S(\vec{y})$ was from a peer in phase l is equal to $f_l(\vec{y}) = y_{c,l}/S(\vec{y})$.

The third situation occurs when the last download is completed, which is essentially the end of the recovery phase. The elements of $\vec{Q}$ corresponding to these three situations are, for $l \in [1..n]$ and $m \in [1..n]$,

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

And last :

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

For illustration purposes, we depict in Fig. 2.3 some of the transitions of the absorbing CTMC when $n = 2$, $s = 4$, $r = 2$, and $k = 1$.

2.7.1 Data lifetime

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

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

The set $\mathcal{E}_I$ consists of all states of the process $\vec{\mathbf{W}}$ 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 $T(\mathcal{E}_I) := \inf\{t > 0 : \vec{W}(t) = a | \vec{W}(0) \in \mathcal{E}_I\}$, 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, $T(\mathcal{E}_I)$ will be referred to as the *conditional block lifetime*. We are interested in the conditional probability distribution function, $P(T(\mathcal{E}_I) \le t)$, and the conditional expectation, $E[T(\mathcal{E}_I)]$, given that $\vec{W}(0) \in \mathcal{E}_I$ for $I \in [s..s+r]$.

From the theory of absorbing Markov chains, we can compute $P(T(\{\vec{w}\}) \le t)$ where $T(\{\vec{w}\})$ is the time until absorption in state a given that the system initiates in state $\vec{w} \in \mathcal{T}$. We know that (e.g. [68, Lemma 2.2])

$$P(T(\{\vec{w}\}) \le t) = 1 - \vec{e}_{|\mathcal{T}|}^{\,\mathrm{ind}(\vec{w})} \cdot \exp\left(t\vec{Q}\right) \cdot \vec{1}_{|\mathcal{T}|}, \quad t > 0,\ \vec{w} \in \mathcal{T} \tag{2.27}$$

where $\text{ind}(\vec{w})$ refers to the index of state $\vec{w}$ in the matrix $\vec{Q}$. Definitions of vectors $\vec{e}_j^{\,i}$ and $\vec{1}_j$ were given at the end of Section 2.3. Observe that the term $\vec{e}_{|\mathcal{T}|}^{\,\text{ind}(\vec{w})} \cdot \exp\left(t\vec{Q}\right) \cdot \vec{1}_{|\mathcal{T}|}$ in the right-hand side of (2.27) is nothing but the summation of all $|\mathcal{T}|$ elements in row $\text{ind}(\vec{w})$ of matrix $\exp\left(t\vec{Q}\right)$.

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

$$\pi_{\vec{x}} := P\left(\vec{W}(0) = \vec{w} \in \mathcal{E}_I | \vec{W}(0) \in \mathcal{E}_I\right) = \binom{I}{x_1, \ldots, x_n} \prod_{l=1}^{n} R(l)^{x_l}. \tag{2.28}$$

Clearly $\sum_{\vec{w} \in \mathcal{E}_I} \pi_{\vec{x}} = 1$ for $I \in [s..s+r]$. Using (2.27) and (2.28) and the total probability theorem yields, for $I \in [s..s+r]$,

$$\begin{aligned} P(T(\mathcal{E}_I) \leq t) &= \sum_{\vec{w} \in \mathcal{E}_I} \pi_{\vec{x}} \, P\left(T(\{\vec{w}\}) \leq t\right) \\ &= 1 - \sum_{\vec{w} \in \mathcal{E}_I} \pi_{\vec{x}} \, \vec{e}_{|\mathcal{T}|}^{\,\text{ind}(\vec{w})} \cdot \exp\left(t\vec{Q}\right) \cdot \vec{1}_{|\mathcal{T}|}, \quad t > 0. \end{aligned} \tag{2.29}$$

We know from [68, p. 46] that the expected time until absorption given that the $\vec{W}(0) = \vec{w} \in \mathcal{T}$ can be written as

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

where the existence of $\left(\vec{Q}\right)^{-1}$ is a consequence of the fact that all states in $\mathcal{T}$ are transient [68, p. 45]. The conditional expectation of $T(\mathcal{E}_I)$ is then (recall that the elements of $\mathcal{E}_I$ are of the form $(\vec{x}, \vec{0}, \vec{0})$

$$\begin{aligned} E\left[T(\mathcal{E}_I)\right] &= \sum_{\vec{w} \in \mathcal{E}_I} \pi_{\vec{x}} \, E\left[T(\{\vec{w}\})\right] \\ &= -\sum_{\vec{w} \in \mathcal{E}_I} \pi_{\vec{x}} \, \vec{e}_{|\mathcal{T}|}^{\,\text{ind}(\vec{w})} \cdot \left(\vec{Q}\right)^{-1} \cdot \vec{1}_{|\mathcal{T}|}, \quad \text{for } I \in [s..s+r]. \end{aligned} \tag{2.30}$$

2.7.2 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 $\mathcal{T}$, for $J \in [0..s+r]$,

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

The set $\mathcal{F}_J$ consists of all states of process $\vec{\mathbf{W}}$ 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 $\mathcal{T}$. We may define now

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

$T(\mathcal{E}_I, \mathcal{F}_J)$ is the total time spent by the CTMC in the set $\mathcal{F}_J$ before being absorbed in state a, given that $\vec{W}(0) \in \mathcal{E}_I$. Similarly, $T(\{\vec{w}\}, \{\vec{w}'\})$ is the total time spent by the CTMC in state $\vec{w}'$ before being absorbed in state a, given that $\vec{W}(0) = \vec{w}$. We know from [46, p. 419] that

$$E\left[T(\{\vec{w}\}, \{\vec{w}'\})\right] = -\vec{e}_{|\mathcal{T}|}^{\,\mathrm{ind}(\vec{w})} \cdot \left(\vec{Q}\right)^{-1} \cdot {}^t\vec{e}_{|\mathcal{T}|}^{\,\mathrm{ind}(\vec{w}')}, \qquad \vec{w}, \vec{w}' \in \mathcal{T} \tag{2.31}$$

where ${}^t\vec{y}$ denotes the transpose of a given vector $\vec{y}$. In other words, the expectation $E[T(\{\vec{w}\}, \{\vec{w}'\})]$ is the entry of matrix $\left(-\vec{Q}\right)^{-1}$ at row $\mathrm{ind}(\vec{w})$ and column $\mathrm{ind}(\vec{w}')$. Using (2.28) and (2.31), we derive for $I \in [s..s+r]$ and $J \in [0..s+r]$

$$\begin{aligned} E[T(\mathcal{E}_I, \mathcal{F}_J)] &= \sum_{\vec{w}' \in \mathcal{F}_J} E\left[T(\mathcal{E}_I, \{\vec{w}'\})\right] \\ &= \sum_{\vec{w} \in \mathcal{E}_I} \sum_{\vec{w}' \in \mathcal{F}_J} \pi_{\vec{x}} E\left[T(\{\vec{w}\}, \{\vec{w}'\})\right] \\ &= -\sum_{\vec{w} \in \mathcal{E}_I} \sum_{\vec{w}' \in \mathcal{F}_J} \pi_{\vec{x}} \vec{e}_{|\mathcal{T}|}^{\,\mathrm{ind}(\vec{w})} \cdot \left(\vec{Q}\right)^{-1} \cdot {}^t\vec{e}_{|\mathcal{T}|}^{\,\mathrm{ind}(\vec{w}')}. \end{aligned} \tag{2.32}$$

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

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

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

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

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, $M_{h,2}^h(\mathcal{E}_{s+r}, s)$ 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

$$E\left[\frac{T(\mathcal{E}_I, \mathcal{F}_J)}{T(\mathcal{E}_I)}\right] \approx \frac{E[T(\mathcal{E}_I, \mathcal{F}_J)]}{E[T(\mathcal{E}_I)]}, \tag{2.33}$$

where the terms in the right-hand side have been derived in (2.32) and (2.30). We will come back to this approximation in Section 2.8. With this approximation in mind, the two availability metrics become

$$M^h_{h,1}(\mathcal{E}_I) = \sum_{J=0}^{s+r} J \, \frac{E[T(\mathcal{E}_I, \mathcal{F}_J)]}{E[T(\mathcal{E}_I)]}, \quad \text{where } I \in [s..s+r], \tag{2.34}$$

$$M^h_{h,2}(\mathcal{E}_I, m) = \sum_{J=m}^{s+r} \frac{E[T(\mathcal{E}_I, \mathcal{F}_J)]}{E[T(\mathcal{E}_I)]}, \quad \text{where } I \in [s..s+r]. \tag{2.35}$$

2.8 Validation of the approximation made to compute the availability metrics

In this section, we validate the approximations (2.5), (2.15), (2.26), and (2.33) which have been made to compute the two availability metrics in our models presented in this chapter. In order to do that, we need to simulate the Markov chain until absorption, and measure $T(\mathcal{E}_I, \mathcal{F}_J)$ and $T(\mathcal{E}_I)$ (we dropped the subscript and the superscript as this applies to any model). The sample mean of their ratio should then be compared to the ratio of the analytical expression. Each simulation scenario should be repeated many times in order to have a good estimation of the mean with a good confidence interval.

We decided to simulate the simple Markov chains $\{X^e_e(t),\ t \geq 0\}$ (Sect. 2.4) and $\{(X^e_h(t), Y^e_h(t)),\ t \geq 0\}$ (Sect. 2.5) as there state-spaces are smaller than that of the extended chains.

The environments that we simulated have the following characteristics: the expected off-time is $1/\lambda = 3$ hour (resp. 1 hour); the expected on-time is $1/\mu = 5$ hour (resp. 3 hour); the persistence probability is $p = 0.8$ (resp. 0.7); the original number of fragments of D is $s = 8$ (resp. 8); the block/fragment sizes are 16MB/2MB (resp. 8MB/1MB); the expected block download time (resp. fragment download time) is $1/\gamma = 30$ minutes (resp. $1/\alpha = 2$ minutes). We simulated a total of 276 (resp. 10) different scenarios, each having different values of r and k. We have varied r from 1 to 23 (resp. from 1 to 4) and k from 1 to r in both models. For $\{X^e_e(t),\ t \geq 0\}$ (resp. for $\{(X^e_h(t), Y^e_h(t)),\ t \geq 0\}$), in each scenario, we have a total of $|\mathcal{T}^e_e| = (r+1)$ (resp. $|\mathcal{T}^e_h| = s(r+1)$) instances of (2.5) (resp. (2.15)), yielding a total of $\sum_{r=1}^{23}(r+1)r = 4600$ (resp. $\sum_{r=1}^{4} 8(r+1)r = 320$) different instances of the approximation over all scenarios.

In order to obtain a maximum estimation error of about 1% with 95% (resp. 97%) confidence interval, we need to have over 100 (resp. 150) sampled values. Hence, each scenario is simulated 100 (resp. 150) times.

For each instance, we collect the 100 (resp. 150) simulated values of the ratio $T(i,j)/T(i)$ (resp. $T^e_h((i,j),(i',j'))/T^e_h((i,j))$) and compute their average. This is the estimation of the

left-hand side of (2.5) (resp. (2.15)) and will be considered as the "correct" value. The right-hand side of (2.5) (resp. (2.15)) is the "approximate" value. We compute the relative error between the correct value and the approximate value. Having collected all values of the relative error from all 4600 (resp. 320) instances, we derive the empirical complementary cumulative distribution function of the relative error, as depicted in Figures 2.4 and 2.5.

We have found that only 10% of the values are larger than 0.75×10^{-3} (resp. 0.9×10^{-3}) and, most importantly, the maximum value of the relative error is 0.0028 (resp. 0.004). *We conclude that the approximation that we have made for computing the availability metrics is very good and will definitely not imperil the correctness of any result based on it*. Even though we have not simulated the extended Markov chains presented in this chapter, we are convinced that the approximation will be equally good.

2.9 Validation of the simple fluid model made in Sect. 2.4.2

In this section, we validate the fluid approximation made in Section 2.4.2. The environments that we considered for the numerical results have the following characteristics: the expected off-time is $1/\lambda = 3$ hour; the expected on-time is $1/\mu = 5$ hour; the persistence probability is $p = 0.8$; the original number of fragments of D is $s = 8$ (resp. 8); the block/fragment sizes are 16MB/2MB; the expected block download time (resp. fragment download time) is $1/\gamma = 30$ minutes.

We have computed the expected number of available redundant fragments $E[\tilde{X}_e^e]$ (resp. $M_{e,1}^e(r)$) from (2.10) (resp. from (2.6)). The results obtained from these two metrics are almost identical. To illustrate the good convergence of the fluid approximation towards the Markov chain, the deviation between $E[\tilde{X}_e^e]$ and $M_{e,1}^e(r)$ are computed. Figure 2.9 delimits the regions where the deviation is within certain value ranges. For instance, in region V the deviation is smaller than 1‰. If the storage system is operating with values of r and k from this region, then it will be attractive to evaluate the data availability using $E[\tilde{X}_e^e]$ instead of $M_{e,1}^e(r)$.

2.10 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.

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 (γ or α) 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 [69]. The distribution of the *recovery process* and the values of its parameters (γ or α) 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 Chapter 4 or build on the flow-level simulation model introduced in Chapter 5 to simulate a large network. Another solution is to estimate the fragment/block download times using log files of some P2P applications or FTP clients run on some peers involved in the P2P storage solution. If the goal is to estimate the gross behavior of the system, we can consider the simple models, and then we need to estimate the mean block download time or the recovery time.

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 thesis 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 P2P backup or storage system that ensures the desired data lifetime, and availability for a reasonable storage overhead r/s and acceptable recovery threshold k. One may be interested in only guaranteeing large data lifetime. Values of r and k are then set according to the desired contour line of the CCDF of data lifetime. 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).

2.11 Numerical results

The models presented in Sections 2.6 and 2.7 are a generalization of those presented in Sections 2.4 and 2.5. As a matter of curiosity, we will compare in this section the results obtained with the simple and the general models presented in Sections 2.5 and 2.7 when considering an environment that is known to violate the exponential assumption on peers on-times made in the simple models. This allows us to see whether the simple models (e.g. this introduced in 2.5) are robust against a violation of this assumption. Once this question addressed, we

solve numerically our models to evaluate the lifetime and availability of data stored on P2P storage systems running in different contexts. Last, we illustrate how our models can be used to engineer storage systems as explained in Section 2.10.

2.11.1 Parameter values

Our mathematical models have been solved numerically using a set of parameters values. However, as we do not have values of all the needed parameters in a distributed environment, we will try to set reasonable and logical values when required for illustrative purpose.

Network parameters λ, $\{p_i, \mu_i\}_{i=1,...,n}$ 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 *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 [26] 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 [90] after the processing of ping requests between each pair of PlanetLab [74] 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 [69], even though they report different flavors of peer "availability". An exponential distribution is found to "reasonably" fit the *All-pairs-ping* data set in [75]. The basic characteristics of the three data sets considered here and the corresponding values of the peers parameters are reported in Table 2.2. 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 [9, 10] 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 section, 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 simple models while the *CSIL* and *Condor* contexts need the more general models. 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 2.2, is the probability of finding a peer connected or equivalently the percentage of on-times in a peer life cycle. We have set $p = 0.7$ in the *Condor* scenario as peers churn rate is very high and $p = 0.3$ or 0.4 oth-

Table 2.2: 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 [69] (best fit)	H_2 [69] (best fit)	Exp. [75] (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 [75]
Mean off-time (hours)	48	1.567 or 0.522	61 [75]
Percentage of on-times	0.727	0.5 or 0.75	0.750
Persistence probability p	0.3 or 0.4	0.7	0.3

erwise. This is to reflect that disconnections in stable environments are likely due to software or hardware problems.

Protocol parameters s**,** r **and** k**.** Nowadays, block sizes in P2P *storage* systems are usually set to either 4MB, 8MB and 9MB (or 16MB for huge files as in Grid Delivery Network (GDN) or backup systems) and fragment sizes are set somewhere between 256KB and 2MB. 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 *CSIL*- and *Condor*-like systems, we will consider block sizes of 4MB and fragment sizes of 1MB and then $s = 4$. Regarding PlanetLab context, we considered block sizes of 8MB and fragment sizes of 1MB and then $s = 8$. In the *CSIL* scenario (resp. PlanetLab-like scenario) where peers churn is low, we vary the redundancy r from 1 to $1.5s = 6$ (resp. from 1 to $s = 8$). In the high dynamic scenario (*Condor*), we vary the redundancy r from 1 to $3s = 12$. In all the considered scenarios, we vary the threshold k from 1 to r.

Observe that the optimal amount of redundancy r comes as a trade-off between high data availability and high storage efficiency and depends on the recovery threshold k. Smaller threshold values allow for smaller amounts of redundant data at the expense of higher band-

width utilization. The trade-off here is between efficient storage use (small r) and efficient bandwidth use (large k).

Recovery process parameters α or γ. Fragments download/upload times depend on the upload/download capacities of the peers, the interconnection link capacities of the underlying network and the load in the system. The measurement study [85] 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 [47] 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). For illustrative purpose and based on the two mentioned studies and our experience from the simulation resultas presented in the two next chapters, we assume that $1/\alpha = 56$, 88 or 104 seconds.

2.11.2 Comparison between simple and extended models

As mentioned previously, the simple models presented in Sections 2.4 and 2.5 are a special case of the general models developed in Sections 2.6 and 2.7, namely when the number of phases of the hyper-exponential distribution of on-times is $n = 1$. Because of the reduced state-space, solving the simple models is much less time consuming than solving the general or the extended models. The simple models can describe 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 developed in Sections 2.5 and 2.7 and compare the results. In [69], 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 2.3 reports the expected data lifetime obtained for $s = 4$, $1/\lambda = 0.522$ hour, $1/\alpha = 22 * s = 88$ seconds, 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 simple model with $1/\mu = 1.543$ hours (best exponential fit found in [69]) 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 $E[T(\mathcal{E}_{s+r})]$ (extended model; column 3) and $E[T^e_h(s+r)]$ (simple model; columns 4 and 6) are reported in columns 5 and 7.

Table 2.3 reveals that the simple 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 simple models do not capture the essence of the system performance when peers on-times are not exponentially distributed. *Henceforth, we will use the simple models in scenarios with the* All-pairs-ping *characteristics, and the general models in scenarios with the characteristics of either* CSIL*or* Condor.

2.11.3 Performance analysis

We have solved numerically the data lifetime and availability metrics (e.g. (2.12), (2.13), (2.16) and (2.17) given that all $s + r$ fragments of D are initially available, considering either Planet-lab, *Condor* or *CSIL* context. Results are reported partially in Table 2.4.

It appears that, whichever the scenario considered, the expected data lifetime increases roughly exponentially with r and decreases with an increasing k. Regardless of the context considered, the distributed scheme yields a significantly small expected data lifetime when peers churn rate is high; cf. columns 3-4 in Table 2.4 where the performance in *Condor*-like systems with distributed-repair scheme (or "repair one missing fragment at a time" policy) is very poor. Observe also how the performance deteriorates as peer churn becomes more important: compare for instance in Table 2.4 row 4 vs. 16 and 23, and row 10 vs. 18 and 25; these correspond to the same storage overhead $r/s = 1$ and the same value of recovery threshold $k = 1$ and 2 respectively, but the context is different.

Regarding the expected number of available fragments, we again observe that the distributed scheme is not efficient when peers churn rate is high; cf. column 4 in Table 2.4 for *Condor*-like systems. We conclude that *when peers churn rate is high, the distributed-repair scheme can not be efficient for storage objective, while the storage overhead should be kept within a reasonable value (that is* $r/s \leq 2$*)*. The conclusion can be different for backup systems as system designers are interested in the permanent departures of peers rather than the intermediate disconnections. However, as the distributed-repair scheme involves less traffic than the centralized one, it will be a good implementation choice in large networks where hosts have a good availability. In addition, the use of a regenerating code will improve the system performance as less traffic needs to be transferred for recovering some unavailable fragments. Our models are able in fact to evaluate systems that use this new class of codes in terms of durability and availability. However, more efforts have to be done to understand the feasibility of these codes with respect to the complication they add to the system.

2.11.4 Engineering the system

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 that 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. 2.7. Consider point A (resp. B) 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 P2P storage system 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.

2.12 Conclusion

We have proposed simple and general analytical models for evaluating the performance of two approaches for recovering lost data in distributed storage systems. Simple fluid model has been introduced under simple assumptions in order to have an explicit formula of the availability metric. We have analyzed the lifetime and the availability of data achieved by distributed-repair systems through markovian analysis considering realistic assumptions. 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 (or "repair one missing fragment at a time" policy) offers a reliable, scalable and cheap storage/backup solution. This is in contrast with the case of highly dynamic environments, where the distributed-repair scheme is inefficient as long as the storage overhead is kept reasonable. P2P storage systems may be applicable in highly dynamic environments with centralized-repair scheme (or "repair all missing fragment" policy) which is the subject of the next chapter.

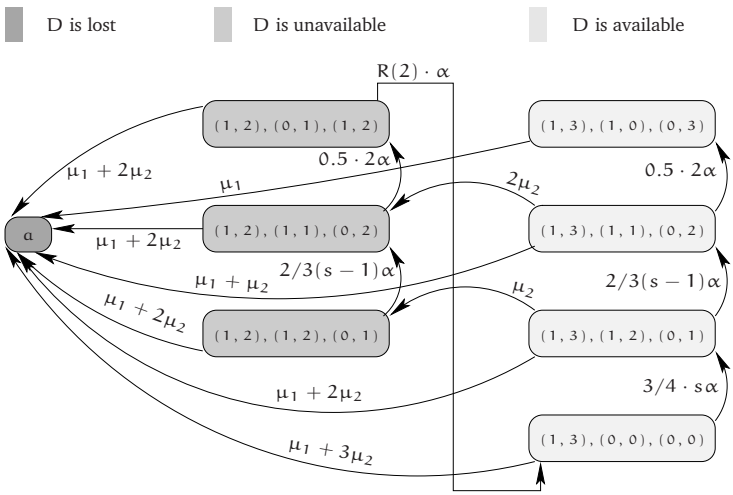


Figure 2.3: Some transition rates of the Markov chain $\vec{\mathbf{W}}$ when $n = 2$, $s = 4$, $r = 2$, and $k = 1$.

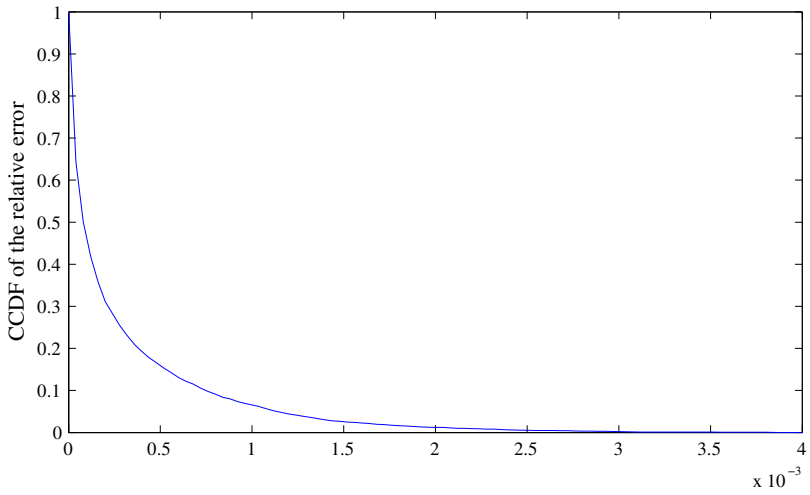


Figure 2.4: The CCDF of the relative error induced by the approximation (2.5).

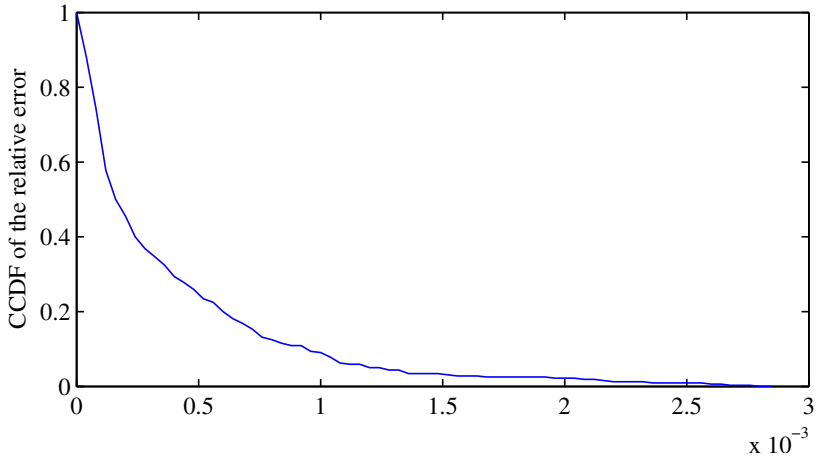


Figure 2.5: The CCDF of the relative error induced by the approximation (2.15).

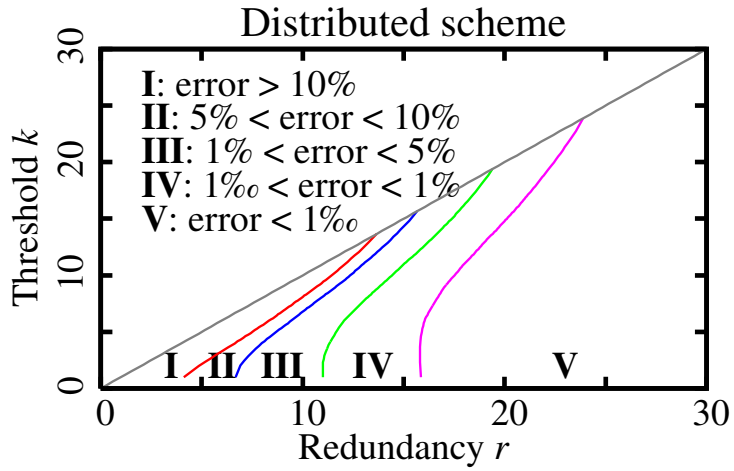


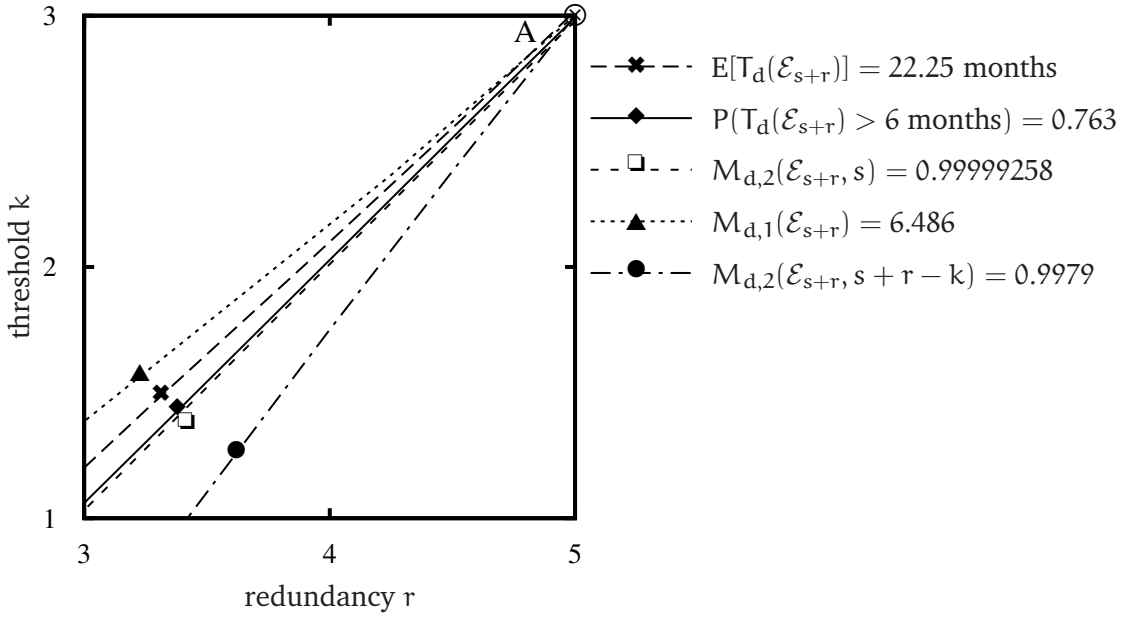
Figure 2.6: Validation of the fluid approximation: Relative error $|M^{e}_{e,1} - E[\tilde{X}^{e}_{e}|/M^{e}_{e,1}$.

Table 2.3: Expected data lifetime (expressed in hours) in a *Condor* scenario using a distributed-recovery scheme. Comparison between $E[T(\mathcal{E}_{s+r})]$ (extended model) and $E[T_h^e(s+r)]$ (simple model).

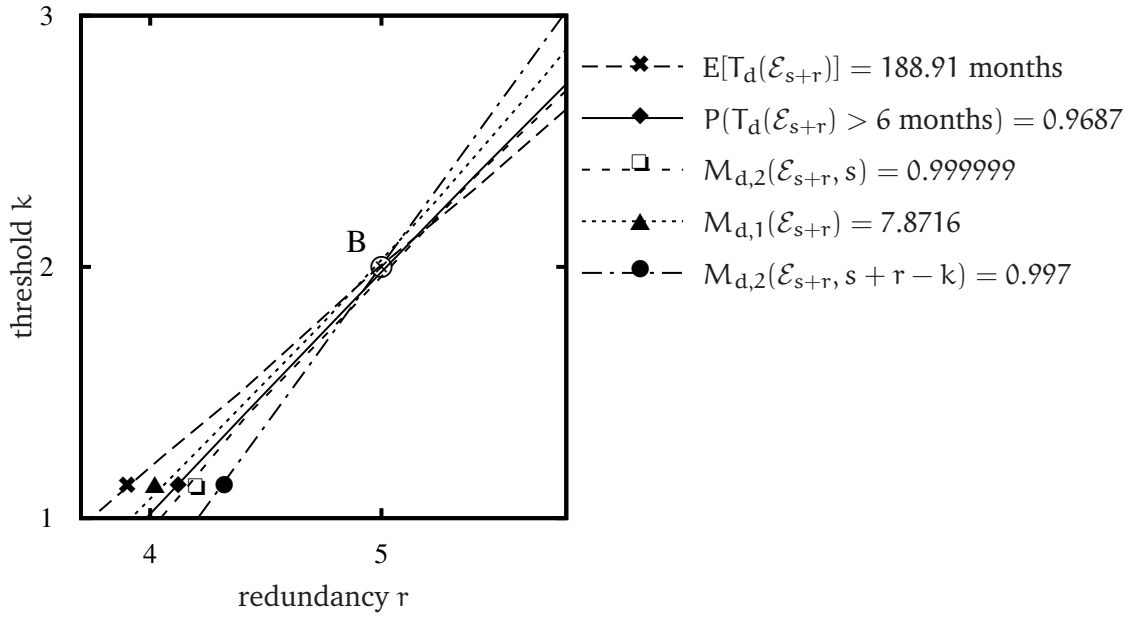
$s=4$		H_2 fit [69]	Exponential fit [69]		equating 1st moments	
		$E[T(\mathcal{E}_{s+r})]$	$E[T_h^e(s+r)]$	error	$E[T_h^e(s+r)]$	error
$k=1$	$r=2$	1.437	0.78	-45.7%	1.017	-29.2%
	$r=4$	5.866	3.453	-41.1%	4.09	-30.2%
	$r=6$	15.751	14.04	-10.8%	14.44	-8.32%
$k=2$	$r=2$	0.729	0.492	-3.5%	0.633	-13.1%
	$r=4$	3.689	2.34	-36.5%	2.74	-25.7%
	$r=6$	12.263	10.464	-14.67%	10.732	-12.48%

Table 2.4: Expected lifetime and first availability metric

Condor context		$E[T_h^h(\mathcal{E}_{s+r})]$ (in days)	$M_{h,1}^h(\mathcal{E}_{s+r})$
	$s = 4$	$1/\alpha = 22 * s$ sec	$1/\alpha = 22 * s$ sec
$k = 1$	$r = 2$	5.99e-02	5.44
	$r = 4$	0.244	6.761
	$r = 6$	0.656	8.01
	$r = 8$	1.841	9.27
	$r = 10$	3.14	10.44
	$r = 12$	8.123	11.41
$k = 2$	$r = 2$	3.04e-02	5
	$r = 4$	0.154	6.31
	$r = 6$	0.511	7.66
$k = 4$	$r = 4$	3.57e-02	5.43
CSIL context		$E[T_h^h(\mathcal{E}_{s+r})]$ (in months)	$M_{h,1}^h(\mathcal{E}_{s+r})$
	$s = 4$	$1/\alpha = 22$ sec	$1/\alpha = 22$ sec
$k = 1$	$r = 2$	3.002	5.980
	$r = 4$	209.36	7.962
$k = 2$	$r = 2$	0.202	5.011
	$r = 4$	24.42	6.985
$k = 4$	$r = 4$	0.162	5.053
CSIL context		$E[T_h^h(\mathcal{E}_{s+r})]$ (in months)	$M_{h,1}^h(\mathcal{E}_{s+r})$
	$s = 4$	$1/\alpha = 56$ sec	$1/\alpha = 56$ sec
$k = 1$	$r = 2$	0.852	5.959
	$r = 4$	21.75	7.919
$k = 2$	$r = 2$	9.94e-02	5.000
	$r = 4$	4.097	6.952
$k = 4$	$r = 4$	7.878e-02	5.046
PlanetLab context		$E[T_h^e(s + r, 0)]$ (in months)	$M_{h,1}^e(s + r, 0)$
	$s = 8$	$1/\alpha = 104$ sec	$1/\alpha = 104$ sec
$k = 1$	$r = 2$	0.11	8.04
	$r = 4$	1.05	8.68
	$r = 6$	7.61	9.80
	$r = 8$	46.24	12.12
$k = 2$	$r = 4$	0.37	8.19
	$r = 6$	3.20	9.25
	$r = 8$	23.34	11.37
$k = 4$	$r = 8$	4.34	9.81



(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 2.7: Contour lines of performance metrics (CSIL context, distributed-repair scheme).

3

PERFORMANCE EVALUATION OF DATA LIFETIME AND AVAILABILITY IN CENTRALIZED-REPAIR SYSTEMS

3.1 Introduction

In this chapter, we focus on the performance evaluation of centralized-repair P2P backup and storage systems, in terms of data lifetime and availability through markovian models under similar assumptions of those made in Chapter 2. The impact of each system parameter on the performance is evaluated, and guidelines are derived on how to engineer the system and tune its key parameters in order to provide desired lifetime and/or availability of data. As was discussed in Section 1.2.2, in the centralized-repair scheme, a *recovery process* can compensate at once multiple losses of a given block of data, requiring multiple fragments, namely s, of that "block" to be downloaded in parallel for an enhanced service.

Concerning the assumptions we make on the recovery process, we first consider it to follow an exponential distribution for the sake of simplification. Second, motivated by the simulation results presented in Chapter 4, we consider that each of the durations of the centralized recovery process is a rv following a hypo-exponential distribution with many distinct phases (generalized Erlang distribution). This is nothing but a consequence of the finding that successive download (resp. upload) durations of a fragment can be seen as iid rvs with a common exponential distribution function with parameter α (resp. β), in addition to the assumption that concurrent fragments downloads are not correlated. Indeed, each of the recovery durations is

the summation of $s + k$ *independently* distributed exponential rvs (if k fragments are to be reconstructed) having each its own rate [50]. Note however that we have found in simulations that these are weakly correlated in some well-known scenarios as we will see in Chapter 4.

Concerning peers availability, we will make two different assumptions as follows. First, we simplify the study and assume that the peers availability is modeled by an exponential distribution, i.e., they follow the assumptions and results of [75] on the peers availability as discussed in Section 1.3.

Second, in light of the conclusions of [69], we assume that peers availability is modeled with a hyper-exponential distribution.

In summary, we propose two simple models in which the peer availability is considered to follow an exponential distribution, and the recovery process is considered to follow an exponential distribution in the first simple model, and a hypo-exponential distribution in the second simple model. We will then extend only the *second* simple model by assuming that peers ontimes durations are hyper-exponentially distributed while keeping the hypo-exponential distribution assumption on the recovery process. Doing so, our modeling is general, realistic and valid under different distributed environments. The evaluation of an extension of the first simple model is essentially the same as that of the general model presented in Section 3.5.

When the "block" download time can be modeled by an exponential distribution, then, the recovery process would follow a hypo-exponential distribution of two phases, having each its own rate. The rate of the first phase is independent of the system state or the number of missing fragments to recover because it always consists of downloading s equally sized fragments (constituting a "block"). However, the rate of the second phase depends on the current state of the system, i.e., the number of fragments to be reconstructed and uploaded. Therefore, modeling the system under such assumption will follow the same methodology of modeling the more general model which we present in Section 3.5.

The rest of this chapter is organized as follows. Section 3.2 introduces system description, assumptions and notation. Sections 3.3 to 3.5 are devoted to the analysis of the distributed-repair P2P backup and storage systems, in terms of data lifetime and availability, through simple and extended markovian models as mentioned above. A simple fluid approximation is as well proposed in Section 3.3. Numerical results that support the analysis, illustrate how to engineer the system in order to provide desired lifetime and/or availability of data, and discuss the impact of parameter values are introduced in Section 3.6.

3.2 System description, assumptions and notation

We consider that same redundancy mechanisms and repair policies introduced in Section 2.2 are enforced throughout this chapter. We make similar assumptions on peers off-

times/on-times durations and on peers independency as what was introduced in the same section (Sect. 2.2), in particular Assumptions 1–3.

We will investigate the performance of the centralized-recovery scheme. Assume that $k \leq r$ fragments are no longer available due to peer disconnections, and have to be restored. In the centralized implementation, a central authority (or the block responsible node in DHT like systems) 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 (e.g. k). 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. Step 1 (resp. Step 3) ends executing when the download (resp. upload) of the last fragment is completed.

Assumption 4: (recovery durations) For the sake of simplification, we assume in the first place that successive recovery durations (total times required to perform the recovery task) are iid rvs exponentially distributed with rate $\gamma(k)$, where k is the number of reconstructed fragments.

Assumption 5: (download/upload durations) We assume in the second place 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 are not correlated.

Assumption 5 is supported by our findings in [32, 31] as explained in Chapter 4. The fragment download/upload time was found to follow approximately an exponential distribution in some interesting contexts (the core network has a good connectivity and the peers upload-/download capacities are asymmetric). 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.

Assumption 6: (recovery durations) A consequence of Assumption 5 is that the recovery processes is a rv following a hypo-exponential distribution [50]. Indeed, each of these durations is the summation of $s + k$ *independently* distributed exponential rvs (if k fragments are to be reconstructed) having each its own rate.

It is worth mentioning that the simulation analysis of [32] has concluded that in some cases the recovery time follows roughly a hypo-exponential distribution. It was also found in [32] that a hypo-exponential model gives a more reasonable approximation of the recovery process than an exponential model even in cases when the null hypothesis is rejected for a good significant

level in such a scenario when the core networks has a good connectivity and the peers upstream and downstream bandwidths are asymmetric.

Recall that we consider two different assumptions on the distribution of peers on-times (Assumption 2 in Section 2.2). The exponential distribution with parameter μ is first used in the simple model whose analysis takes place in Section 3.3 where the recovery process is exponentially distributed and is second used in Section 3.4 where the recovery process is hypo-exponentially distributed. The hyper-exponential distribution with n phases is considered in the more general model whose analysis is presented in Section 3.5 where the recovery process is hypo-exponentially distributed.

Last, we use same notation introduced at the end of Section 2.3 in Chapter 2.

3.3 Simple model, recovery process is exponentially distributed

In order to introduce a simple model, we consider in this section that successive peers off-times (resp. on-times) durations and the recovery durations are exponentially distributed with parameters λ (resp. μ) and $\gamma(k)$ (if k fragments are to be reconstructed).

Let $X_e^e(t)$ be a $\{a, 0, 1, \ldots, r\}$-valued rv, where $X_e^e(t) = i \in \mathcal{T}^e := \{0, 1, \ldots, r\}$ indicates that $s + i$ fragments of D are available at time t, and $X_e^e(t) = a$ indicates that less than s fragments of D are available at time t. We assume that $X_e^e(0) \in \mathcal{T}_e^e$ so as to reflect the assumption that at least s fragments are available at $t = 0$.

If at a given time t a peer disconnects from the storage system while $X_e^e(t) = 0$, then there will be strictly less than s fragments of D in the system. Recovering then lost fragments is impossible unless one of the peers having a fragment of D reconnects to the system *and* still stored its data. Recall that the latter event occurs with probability p; in other words, recovering D becomes a probabilistic event. The block of data D is available with probability 1 as long as there are at least s fragments of D (implying $X_e^e(t) \in \mathcal{T}_e^e$). Otherwise, we consider the block D to be lost.

Thanks to the considered assumptions, it is easily seen that $\mathbf{X}_e^e := \{X_e^e(t),\ t \geq 0\}$ is an absorbing homogeneous continuous-time Markov chain (CTMC) with transient states $0, 1, \ldots, r$ and with a single absorbing state a representing the situation when D is lost. Non-zero transition rates of $\{X_e^e(t),\ t \geq 0\}$ are displayed in Fig. 3.1.

3.3.1 Data lifetime

This section is devoted to the analysis of the data lifetime. Let $T_e^e(i) := \inf\{t \geq 0 : X_e^e(t) = a\}$ be the time until absorption in state a starting from $X_e^e(0) = i$. In the following, $T_e^e(i)$ will be referred to as the *conditional block lifetime*.

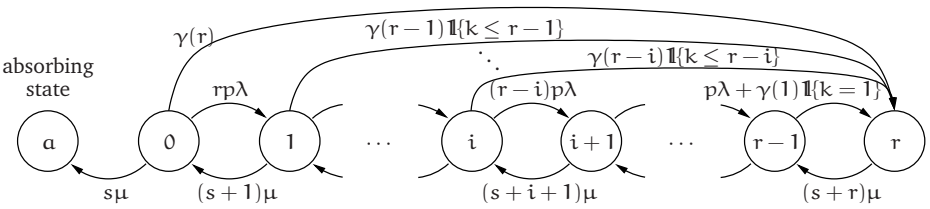


Figure 3.1: Transition rates of the absorbing Markov chain $\{X_e^e(t), t \ge 0\}$.

We are interested in $P(T_e^e(i) \le x)$ and $E[T_e^e(i)]$, the probability distribution block lifetime and the expectation of the block lifetime, respectively, given that $X_e^e(0) = i$ for $i \in \mathcal{T}_e^e$.

The infinitesimal generator has the following canonical form

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

where $\vec{R}_e^e$ is a non-zero column vector of size $|\mathcal{T}_e^e| = r+1$, and $\vec{Q}_e^e$ is $|\mathcal{T}_e^e|$-by-$|\mathcal{T}_e^e|$ matrix. The elements of $\vec{R}_e^e$ are the transition rates between the transient states $x \in \mathcal{T}_e^e$ and the absorbing state a. The diagonal elements of $\vec{Q}_e^e$ are each the total transition rate out of the corresponding transient state. The other elements of $\vec{Q}_e^e$ are the transition rates between each pair of transient states. The only non-zero element of $\vec{R}_e^e$ in this simple model is $s\mu$ for $x = 0$. Let us proceed to the definition of the non-zero entries of $\vec{Q}_e^e$.

$$\begin{array}{lcll} q_e^e(i,i-1) & = & a_i, & i = 1,2,\ldots,r, \\ q_e^e(i,i+1) & = & b_i + \mathbb{1}\{i = r-1\}c_{r-1}, & i = 0,1,\ldots,r-1, \\ q_e^e(i,r) & = & c_i, & i = 0,1,\ldots,\min\{r-k, r-2\}, \\ q_e^e(i,i) & = & -(a_i+b_i+c_i), & i = 0,1,\ldots,r, \end{array} \tag{3.1}$$

where $a_i := (s+i)\mu$, $b_i := (r-i)p\lambda$ and $c_i := \gamma(r-i)\mathbb{1}\{i \le r-k\}$ for $i \in \mathcal{T}_e^e$. Note that $\vec{Q}_e^e$ is not an infinitesimal generator since entries in its first row ($i = 0$) do not sum up to 0.

From the theory of absorbing Markov chains, we know that (e.g. [68, Lemma 2.2])

$$P(T_e^e(i) \le x) = 1 - \vec{e}_{r+1}^{\,i+1} \cdot \exp\left(x\vec{Q}_e^e\right) \cdot \vec{1}_{r+1}, \quad x > 0, \quad i \in \mathcal{T}_e^e. \tag{3.2}$$

Definitions of vectors $\vec{e}_r^{\,i}$ and $\vec{1}_r$ are given at the end of Section 2.3.

We also know that the expectation of the time until absorption can be written as [68, p. 46]

$$E\left[T_e^e(i)\right] = -\mathbf{e}_{r+1}^{(i+1)} \cdot \left(\vec{Q}_e^e\right)^{-1} \cdot \vec{1}_{r+1}, \quad i \in \mathcal{T}_e^e, \tag{3.3}$$

where the existence of $\left(\vec{Q}_e^e\right)^{-1}$ is a consequence of the fact that all states in $\mathcal{T}_e^e$ are transient [68, p. 45].

Consider now

$$T_e^e(i,j) \;:=\; \int_0^{T_e^e(i)} \mathbb{1}\{X_e^e(t)=j\}dt$$

that is the total time spent by the CTMC $\mathbf{X}_c^e$ in transient state j given that $X_e^e(0)=i$. It can also be shown that [46]

$$E\left[T_e^e(i,j)\right] \;=\; -\vec{e}_{r+1}^{\,i+1}\cdot\left(\vec{Q}_e^e\right)^{-1}\cdot {}^t\vec{e}_{r+1}^{\,j+1}, \quad i,j\in\mathcal{T}_e^e, \tag{3.4}$$

where ${}^t\vec{y}$ denotes the transpose of any row vector $\vec{y}$. In other words, $E[T_e^e(i,j)]$ is the (i,j)-th entry of matrix $-\left(\vec{Q}_e^e\right)^{-1}$.

Even when $\gamma(0)=\cdots=\gamma(r)$, an explicit calculation of either $P(T_e^e(i)<x)$, $E[T_e^e(i)]$ or $E[T_e^e(i,j)]$ is intractable, for any value of the threshold k in $\{1,2,\ldots,r\}$. Numerical results for $E[T_e^e(r)]$ and $P(T_e^e(r)>x)$ are instead reported in Section 3.6 when $\gamma(0)=\cdots=\gamma(r)$.

3.3.2 Data availability

In this section we introduce different metrics to quantify the availability of the block of data.

Similar to what was done in the previous chapter, the fraction of time spent by the absorbing Markov chain $\{X_e^e(t),\, t\geq 0\}$ in state j starting at time $t=0$ from state i is approximated by the ratio

$$\frac{E[T_e^e(i,j)]}{E[T_e^e(i)]}.$$

Note that we have validated this approximation by simulation in Section 2.8 as shown in Figures 2.4 and 2.5.

With this approximation in mind, we introduce the first availability metric

$$M_{e,1}^e(i) \;:=\; \sum_{j=0}^{r} j\frac{E[T_e^e(i,j)]}{E[T_e^e(i)]}, \quad i\in\mathcal{T}_e^e, \tag{3.5}$$

that we can interpret as the expected number of available redundant fragments during the block lifetime, given that $X_e^e(0)=i\in\mathcal{T}_e^e$.

A second metric is

$$M_{e,2}^e(i) \;:=\; \sum_{j=m}^{r} \frac{E[T_e^e(i,j)]}{E[T_e^e(i)]}, \quad i\in\mathcal{T}_e^e, \tag{3.6}$$

that we can interpret as the fraction of time when there are at least m redundant fragments during the block lifetime, given that $X_e^e(0)=i\in\mathcal{T}_e^e$.

Continuous time Markov chain CTMC

Since it is difficult to come up with an explicit expression for either metric $M^e_{e,1}(i)$ or $M^e_{e,2}(i)$, we make the assumption that parameters k and r have been selected so that the time before absorption is arbitrarily "large". This can be formalized, for instance, by requesting that $P(T^e_e(r) > q) > 1 - \epsilon$, where parameters q and ϵ are set according to the particular storage application(s).

In this setting, one may represent the state of the storage system by a new Markov chain $\tilde{\mathbf{X}}^e_e := \{\tilde{X}^e_e(t),\, t \geq 0\}$, which is irreducible and aperiodic – and therefore ergodic – on the state-space $\mathcal{T}^e_e$. Let $\tilde{\vec{Q}}^e_e = [\tilde{q}^e_e(i,j)]_{i,j\in\mathcal{T}^e_e}$ be its infinitesimal generator. Matrices $\tilde{\vec{Q}}^e_e$ and $\vec{Q}^e_e$—whose non-zero entries are given in (3.1)—are identical except for $\tilde{q}^e_e(0,0) = -(b_0 + c_0)$. Until the end of this section we assume that $\gamma(i) = \gamma$ for $i \in \mathcal{T}^e_e$.

Let $\pi(i)$ be the stationary probability that $\tilde{\mathbf{X}}^e_e$ is in state i. Our objective is to compute $E[\tilde{X}^e_e] = \sum_{i=0}^{r} i\pi(i)$, the (stationary) expected number of available redundant fragments. To this end, let us introduce $f(z) = \sum_{i=0}^{r} z^i\pi(i)$, the generating function of the stationary probabilities $\pi = (\pi(0), \pi(1), \ldots, \pi(r))$.

Starting from the Kolmogorov balance equations $\pi\tilde{\vec{Q}}^e_e = 0$, and using the normalizing equation $\pi \cdot \vec{1}_{r+1} = 1$, standard algebra yields

$$\begin{aligned}(\mu + p\lambda z)\frac{df(z)}{dz} &= rp\lambda f(z) - s\mu\frac{f(z) - \pi(0)}{z} + \gamma\,\frac{f(z) - z^r}{1 - z} \\ &\quad -\gamma \sum_{i=r-k+1}^{r} \frac{z^i - z^r}{1 - z}\pi(i).\end{aligned}$$

Letting $z = 1$ and using the identities $f(1) = 1$ and $df(z)/dz|_{z=1} = E[\tilde{X}^e_e]$, we find

$$E[\tilde{X}^e_e] = \frac{r(p\lambda + \gamma) - s\mu(1 - \pi(0)) - \gamma\sum_{i=0}^{k-1} i\pi(r-i)}{\mu + p\lambda + \gamma}. \tag{3.7}$$

Unfortunately, it is not possible to find an explicit expression for $E[\tilde{X}^e_e]$ since this quantity depends on the probabilities $\pi(0), \pi(r-(k-1)), \pi(r-(k-2)), \ldots, \pi(r)$, which cannot be computed in explicit form. If $k = 1$ then

$$E[\tilde{X}^e_e] = \frac{r(p\lambda + \gamma) - s\mu(1 - \pi(0))}{\mu + p\lambda + \gamma}, \tag{3.8}$$

which still depends on the unknown probability $\pi(0)$.

Below, we use a mean field approximation to develop an approximation formula for $E[\tilde{X}^e_e]$ for $k = 1$, in the case where the maximum number of redundant fragments r is large.

Simple fluid model

Using [60, Thm. 3.1] and similar to what was done in Section 2.4.2, we know that, when r is large, the expected number of available redundant fragments at time t, $E[\tilde{X}^e_e(t)]$, is solution

of the following first-order differential (ODE) equation

$$\dot{y}(t) = -(\mu + p\lambda + \gamma)y(t) - s\mu + r(p\lambda + \gamma).$$

The equilibrium point of the above ODE is reached when time goes to infinity, which suggests to approximate $E[\tilde{X}_e^e]$, when r is large, by

$$E[\tilde{X}_e^e] \approx y(\infty) = \frac{r(p\lambda + \gamma) - s\mu}{\mu + p\lambda + \gamma}. \tag{3.9}$$

Observe that this simply amounts to neglect the probability $\pi(0)$ in (3.8) for large r.

3.4 Simple Model, recovery process is hypo-exponentially distributed

We consider in this section that successive peers off-times (resp. on-times) durations are exponentially distributed with parameters λ (resp. μ), and 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 are not correlated. A consequence, the recovery processes is a rv following a hypo-exponential distribution of $s + k$ phases [50] (if k fragments are to be reconstructed) having each its own rate.

Let $X_h^e(t)$ and $Y_h^e(t)$ be two rvs denoting respectively the number of fragments in the system that are available for download and the state of the recovery process. Recall that, when k fragments are to be reconstructed, the recovery process consists of a series of $s + k$ exponential distributions that can be seen as $s + k$ stages. We denote $Y_h^e(t) = j$ ($j = 0, 1, \ldots, s + k - 1$) to express that j exponential rvs have been realized at time t, so that $s + k - j$ are still to go. When the last stage is completed, the recovery process is completed and $Y_h^e(t) = 0$. Unlike the distributed-recovery scheme, given that there could be as much as $s + r$ fragments to be reconstructed, the process $Y_h^e(t)$ takes value in the set $\{0, 1, \ldots, 2s + r - 1\}$. As for $X_h^e(t)$, it takes value in the set $\{0, 1, \ldots, s + r\}$.

Consider now the joint process $(X_h^e(t), Y_h^e(t))$. When $X_h^e(t) \geq s$, data D is *available*, regardless of $Y_h^e(t)$. When $X_h^e(t) < s$ but $X_h^e(t) + Y_h^e(t) \geq s$, D is *unavailable*. When $X_h^e(t) + Y_h^e(t) < s$,

D is *lost*. The latter situation will be modeled by a single state a. Introduce the set

$$\begin{aligned}
\mathcal{T}_h^e := \{\ & (0,s),(0,s+1),\ldots,(0,2s+r-1),\\
& (1,s-1),(1,s),\ldots,(1,2s+r-2),\\
& \ldots, \qquad \Big\}\ \text{D is unavailable}\\
& (s,0),(s,1),\ldots,(s,s+r-1),\\
& (s+1,0),(s+1,1),\ldots,(s+1,s+r-2),\\
& \ldots,(s+r-1,0),(s+r-1,1),\ldots,(s+r-1,s),\\
& (s+r,0)\ \Big\}\ \text{D is available}\\
|\mathcal{T}_h^e| &= (s+r)^2 - r(r-1)/2 + 1.
\end{aligned}$$

Thanks to the considered assumptions, it is easily seen that the two-dimensional process $\{(X_h^e(t), Y_h^e(t)),\ t \geq 0\}$ is an absorbing homogeneous Continuous-Time Markov Chain (CTMC) with transient states the elements of $\mathcal{T}_h^e$ and with a single absorbing state a representing the situation when D is lost. Without loss of generality, we assume that $X_h^e(0) \geq s$. The infinitesimal generator has the following canonical form

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

where $\vec{R}_h^e$ is a non-zero column vector of size $|\mathcal{T}_h^e|$, and $\vec{Q}_h^e$ is $|\mathcal{T}_h^e|$-by-$|\mathcal{T}_h^e|$ matrix. The elements of $\vec{R}_h^e$ are the transition rates between the transient states $(i,j) \in \mathcal{T}_h^e$ and the absorbing state a, namely, $r_h^e(i,j) = (s-j)\mu$, for $i = 1,\ldots,s$, and $j = s-i,\ldots,s-1$. The elements of $\vec{R}_h^e$ are lexicographically ordered alike the order in $\mathcal{T}_h^e$. The diagonal elements of $\vec{Q}_h^e$ are each the total transition rate out of the corresponding transient state. The other elements of $\vec{Q}_h^e$ are the transition rates between each pair of transient states. The non-zero elements of $\vec{Q}_h^e$ are:

$$q_h^e((i,j),(i-1,j)) = \begin{cases}
i\mu, & \text{for} \quad i = 1,\ldots,s, j = s,\ldots,2s+r-1-i;\\
& \text{or} \quad i = s+1,\ldots,s+r-1,\\
& \qquad\ j = 0,\ldots,2s+r-1-i;\\
& \text{or} \quad i = s+r, j = 0;\\
(i+j-s)\mu, & \text{for} \quad i = 2,\ldots,s, j = s+1-i,\ldots,s-1.
\end{cases}$$

$$q_h^e((i,j),(i,j+1)) = \begin{cases}
(s-j)\alpha, & \text{for} \quad i = s,\ldots,s+r-k, j = 0;\\
& \text{or} \quad i = 1,\ldots,s-1, j = s-i,\ldots,s-1;\\
& \text{or} \quad i = s,\ldots,s+r-1, j = 1,\ldots,s-1;\\
(2s+r-i-j)\beta, & \text{for} \quad i = 0,\ldots,s+r-2,\\
& \qquad\ j = s,\ldots,2s+r-2-i.
\end{cases}$$

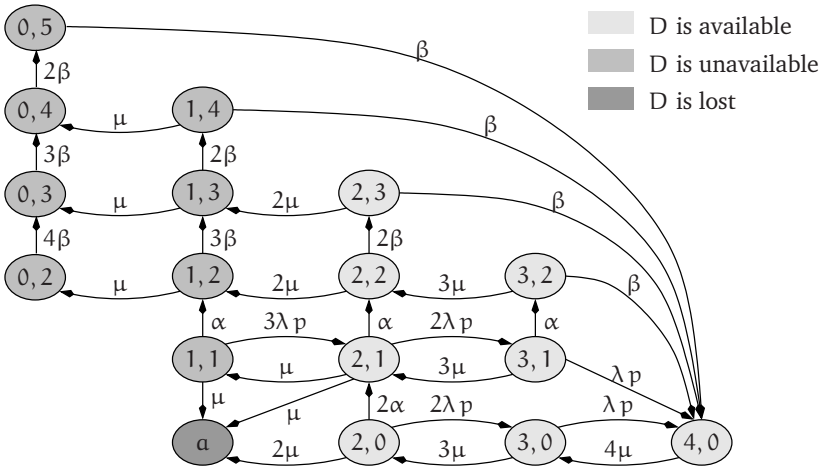


Figure 3.2: The Markov chain $\{(X_h^e(t), Y_h^e(t)),\ t \geq 0\}$ when $s = 2$, $r = 2$, $k = 2$.

$$
\begin{aligned}
&q_h^e((i, 2s+r-1-i), (s+r, 0)) = \beta, && \text{for } i = 0, \ldots, s+r-1.\\
&q_h^e((i,j), (i+1,j)) = (s+r-i)\lambda p, && \text{for } i = 1, \ldots, s, j = s-i, \ldots, s-1;\\
& && \text{or } i = s+1, \ldots, s+r-2, j = 0, \ldots, s-1.\\
&q_h^e((s+r-1,j), (s+r, 0)) = \lambda p, && \text{for } j = 0, \ldots, s-1.\\
&q_h^e((i,j),(i,j)) = -r_c(i,j) - \textstyle\sum_{(i',j') \in \mathcal{T}_h^e - \{(i,j)\}} q_h^e((i,j),(i',j')), && \text{for } (i,j) \in \mathcal{T}_h^e.
\end{aligned}
$$

Note that $\vec{Q}_h^e$ is not an infinitesimal generator since entries in some rows do not sum up to 0. For illustration purposes, we depict in Fig. 3.2 an example of the absorbing CTMC with its non-zero transition rates when $s = 2$, $r = 2$, and $k = 2$.

3.4.1 Data lifetime

This section is devoted to the analysis of the lifetime of D. Let $T_h^e(i,j) := \inf\{t > 0 : (X_h^e(t), Y_h^e(t)) = a | (X_h^e(0), Y_h^e(0)) = (i,j)\}$ be the *conditional block lifetime*. We are interested in $P(T_h^e(i,j) \leq x)$ and $E[T_h^e(i,j)]$ given that $(X_h^e(0), Y_h^e(0)) = (i,j) \in \mathcal{T}_h^e$. From the theory of absorbing Markov chains, we know that (e.g. [68, Lemma 2.2])

$$P(T_h^e(i,j) \leq x) = 1 - \vec{e}_{|\mathcal{T}_h^e|}^{\,\mathrm{ind}(i,j)} \cdot \exp\left(x\vec{Q}_h^e\right) \cdot \vec{1}_{|\mathcal{T}_h^e|}, \quad x > 0,\ (i,j) \in \mathcal{T}_h^e \tag{3.10}$$

where $\mathrm{ind}(i,j)$ refers to the index of the state $(i,j) \in \mathcal{T}_h^e$ in the matrix $\vec{Q}_h^e$. Recall that the elements of $\vec{Q}_h^e$ are numbered according to the lexicographic order. Definitions of vectors $\vec{e}_i^{\,j}$ and $\vec{1}_i$ are given at the end of Section 2.3. Observe that the term $\vec{e}_{|\mathcal{T}_h^e|}^{\,\mathrm{ind}(i,j)} \cdot \exp\left(x\vec{Q}_h^e\right) \cdot \vec{1}_{|\mathcal{T}_h^e|}$ in the r.h.s. of (3.10) is nothing but the summation of all $|\mathcal{T}_h^e|$ elements in row $\mathrm{ind}(i,j)$ of matrix $\exp\left(x\vec{Q}_h^e\right)$.

We know from [68, p. 46] that the expected time until absorption can be written as

$$\mathrm{E}\left[T_h^e(i,j)\right] = -\vec{e}_{|\mathcal{T}_h^e|}^{\,\mathrm{ind}(i,j)} \cdot \left(\vec{Q}_h^e\right)^{-1} \cdot \vec{1}_{|\mathcal{T}_h^e|}, \quad (i,j) \in \mathcal{T}_h^e, \tag{3.11}$$

where the existence of $\left(\vec{Q}_h^e\right)^{-1}$ is a consequence of the fact that all states in $\mathcal{T}_h^e$ are transient [68, p. 45]. Inverting $\vec{Q}_h^e$ analytically can rapidly become cumbersome as s or r increases. We will instead perform numerical computations as reported in Section 5.4. Consider now

$$T_h^e((i,j),(i',j')) := \int_0^{T_h^e(i,j)} \mathbb{1}\left\{(X_h^e(t), Y_h^e(t)) = (i',j')\right\} \mathrm{d}t$$

that is the total time spent by the CTMC in transient state (i',j') given that $\{X_h^e(0), Y_h^e(0)\} = (i,j)$. It can also be shown that [46, p. 419]

$$\mathrm{E}\left[T_h^e((i,j),(i',j'))\right] = -\vec{e}_{|\mathcal{T}_h^e|}^{\,\mathrm{ind}(i,j)} \cdot \left(\vec{Q}_h^e\right)^{-1} \cdot {}^{\mathrm{t}}\vec{e}_{|\mathcal{T}_h^e|}^{\,\mathrm{ind}(i',j')}, \quad (i,j),(i',j') \in \mathcal{T}_h^e, \tag{3.12}$$

where ${}^{\mathrm{t}}\vec{y}$ denotes the transpose of a given vector $\vec{y}$. In other words, the expectation $\mathrm{E}\left[T_h^e((i,j),(i',j'))\right]$ is the entry of matrix $\left(-\vec{Q}_h^e\right)^{-1}$ at row $\mathrm{ind}(i,j)$ and column $\mathrm{ind}(i',j')$.

3.4.2 Data availability

In this section we introduce different metrics to quantify the availability of D. We are interested in the fraction of time spent by the CTMC in any given state (i',j') before absorption. However, this quantity is difficult to find in closed-form. Therefore, we resort to using the following approximation

$$\mathrm{E}\left[\frac{T_h^e((i,j),(i',j'))}{T_h^e(i,j)}\right] \approx \frac{\mathrm{E}[T_h^e((i,j),(i',j'))]}{\mathrm{E}[T_h^e(i,j)]}. \tag{3.13}$$

Here, (i,j) is the state of D at $t = 0$. This approximation have been validated through simulations in the distributed implementation of recovery process in Section 2.8. With this approximation in mind, we introduce two availability metrics: the first can be interpreted as the expected number of fragments of D that are in the system during the lifetime of D; the second can be interpreted as the fraction of time when at least m fragments are in the system during the lifetime of D. More formally, given that $(X_h^e(0), Y_h^e(0)) = (i,j) \in \mathcal{T}_h^e$, we define

$$M_{h,1}^e(i,j) := \sum_{(i',j') \in \mathcal{T}_h^e} i' \frac{\mathrm{E}[T_h^e((i,j),(i',j'))]}{\mathrm{E}[T_h^e(i,j)]}, \tag{3.14}$$

$$M_{h,2}^e((i,j),m) := \sum_{(i',j') \in \mathcal{T}_h^e, i' \geq m} \frac{\mathrm{E}[T_h^e((i,j),(i',j'))]}{\mathrm{E}[T_h^e(i,j)]}. \tag{3.15}$$

3.5 General model, recovery process is hypo-exponentially distributed and peers availability is hyper-exponentially distributed

We aim in the section to introduce a general model that is able to evaluate the centralized-repair P2P backup and storage systems. We assume that the recovery process is hypo-exponentially distributed and peers availability is hyper-exponentially distributed following the Assumptions 1, 2 for $n > 1$, 3 and 5–6 introduced in Sections 2.2 and 3.2.

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

- $\vec{X}^h_h(t) := (X^h_{h,1}(t), \ldots, X^h_{h,n}(t))$ where $X^h_{h,l}(t)$ 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.
- $\vec{Y}^h_h(t) := (Y^h_{h,1}(t), \ldots, Y^h_{h,n}(t))$ where $Y^h_{h,l}(t)$ 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).
- $\vec{Z}^h_h(t) := (Z^h_{h,1}(t), \ldots, Z^h_{h,n}(t))$ where $Z^h_{h,l}(t)$ 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.
- $\vec{U}^h_h(t) := (U^h_{h,1}(t), \ldots, U^h_{h,n}(t))$ where $U^h_{h,l}(t)$ 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).
- $\vec{V}^h_h(t) := (V^h_{h,1}(t), \ldots, V^h_{h,n}(t))$ where $V^h_{h,l}(t)$ 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 $Y^h_{h,l}(t) \leq X^h_{h,l}(t)$ 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 $S(\vec{X}^h_h(t))$

(recall the definition of the function S in Section 2.3). 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) $S(\vec{Y}_h^h(t)) + S(\vec{Z}_h^h(t)) = s$, such that $S(\vec{Y}_h^h(t)), S(\vec{Z}_h^h(t)) \in [1..s-1]$. Once the download phase is completed, the central authority will reconstruct at once all missing fragments, that is $s + r - S(\vec{X}_h^h(t))$. Therefore, during the upload phase, we have $S(\vec{U}_h^h(t)) + S(\vec{V}_h^h(t)) = s + r - S(\vec{X}_h^h(t))$. Observe that, once the download phase is completed, the number of available fragments, $S(\vec{X}_h^h(t))$, 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 $\vec{Y}_h^h(t) = \vec{0}$ and $S(\vec{Z}_h^h(t)) = s$. The end of the upload phase is also the end of the recovery process. We will then have $\vec{Y}_h^h(t) = \vec{Z}_h^h(t) = \vec{U}_h^h(t) = \vec{V}_h^h(t) = \vec{0}$ until the recovery process is again triggered.

According to the terminology introduced in Section 2.2, at time t, data D is *available* if $S(\vec{X}_h^h(t)) \geq s$, regardless of the state of the recovery process. It is *unavailable* if $S(\vec{X}_h^h(t)) < s$ but $S(\vec{Z}_h^h(t))$—the number of fragments hold by the central authority—is larger than $s - S(\vec{X}_h^h(t))$ *and* at least $s - S(\vec{X}_h^h(t))$ fragments out of $S(\vec{Z}_h^h(t))$ are different from those $S(\vec{X}_h^h(t))$ 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,

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

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 $b(\vec{X}_h^h(t), \vec{0}, \vec{0}) = S(\vec{X}_h^h(t))$ gives the exact number of distinct fragments.

The system state at time t can be represented by the $5n$-dimensional vector $\vec{W}(t)$. The multi-dimensional process $\vec{\mathbf{W}} := \{\vec{W}(t), t \geq 0\}$ is an absorbing homogeneous continuous-time Markov chain (CTMC) with a set of transient states $\mathcal{T}_h^h$ 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 $\mathcal{T}_h^h$ is tedious, we will simply say that $\mathcal{T}_h^h$ 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 $\mathcal{T}_h^h$ must verify the constraints mentioned above.

Without loss of generality, we assume that $S(\vec{X}_h^h(0)) \geq s$. The infinitesimal generator has

the following canonical form

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

where definitions of $\vec{R}_h^h$ and $\vec{Q}_h^h$ are similar of $\vec{R}_e^e$ and $\vec{Q}_e^e$ introduced in Section 3.3. The non-zero elements of $\vec{R}_c$ are, for $S(\vec{y}_h^h) \in [1..S(\vec{x}_h^h)]$ and $S(\vec{z}_h^h) = s - S(\vec{y}_h^h)$,

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

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

Let us proceed to the definition of the non-zero elements of $\vec{Q}_h^h$. First, let us drop **hereafter and until the end of this section** the subscript h and the superscript h from the random variables and metrics to simplify the readability of the equations.

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 $m \in [1..n]$,

$$
\begin{aligned}
&q((\vec{x},\vec{0},\vec{0},\vec{0},\vec{0}),(\vec{x}-\vec{e}_n^{\,l},\vec{0},\vec{0},\vec{0},\vec{0})) = x_l\mu_l,\\
&\qquad \text{for} \quad S(\vec{x}) \in [s+1..s+r].\\
&q((\vec{x},\vec{y},\vec{z},\vec{0},\vec{0}),(\vec{x}-\vec{e}_n^{\,l},\vec{y},\vec{z},\vec{0},\vec{0})) = [x_l - y_l]^+\mu_l,\\
&\qquad \text{for} \quad S(\vec{x}) \in [s..s+r-1], \quad S(\vec{y}) \in [1..s-1], \quad S(\vec{z}) = s - S(\vec{y});\\
&\qquad \text{or} \quad S(\vec{x}) \in [2..s-1], \quad S(\vec{y}) \in [1..S(\vec{x})-1], \quad S(\vec{z}) = s - S(\vec{y}).\\
&q((\vec{x},\vec{y},\vec{z},\vec{0},\vec{0}),(\vec{x}-\vec{e}_n^{\,l},\vec{y}-\vec{e}_n^{\,l}+\vec{e}_n^{\,m},\vec{z},\vec{0},\vec{0})) = \frac{y_l\mu_l[x_m - y_m - z_m]^+}{\sum_{i=1}^{n}[x_i - y_i - z_i]^+},\\
&\qquad \text{for} \quad S(\vec{x}) \in [s..s+r-1], \quad S(\vec{y}) \in [1..s-1], \quad S(\vec{z}) = s - S(\vec{y});\\
&\qquad \text{or} \quad S(\vec{x}) \in [2..s-1], \quad S(\vec{y}) \in [1..S(\vec{x})-1], \quad S(\vec{z}) = s - S(\vec{y}).
\end{aligned}
$$

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

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

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

The case when one download is completed during the recovery process

When a recovery process is initiated, the system state verifies $S(\vec{x}) \in [s..s+r-k]$ and $\vec{y} = \vec{z} = \vec{u} = \vec{v} = \vec{0}$. The central authority selects s peers out of the $S(\vec{x})$ 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 $0 \le i_l \le x_l$, for $l \in [1..n]$, and $S(\vec{i}) = s$. This selection occurs with probability

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

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 2.3). Similarly, when the number of ongoing downloads is $\vec{y}$, the probability that the first download to be completed out of $S(\vec{y})$ was from a peer in phase l is equal to $f_l(\vec{y}) = y_l/S(\vec{y})$.

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

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

The case when one upload is completed during the recovery process

When the download phase is completed, the system state verifies $S(\vec{z}) = s$ and $\vec{y} = \vec{u} = \vec{v} = \vec{0}$. The central authority selects $s + r - S(\vec{x})$ *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 $s + r - S(\vec{x})$ would be in phase l, for $l \in [1..n]$. Let $\vec{i} = (i_1,\ldots,i_n)$. We naturally have $0 \le i_l \le s+r-S(\vec{x})$, for $l \in [1..n]$, and $S(\vec{i}) = s+r-S(\vec{x})$. This selection occurs with probability

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

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

$$q((\vec{x},\vec{0},\vec{z},\vec{0},\vec{0}),(\vec{x},\vec{0},\vec{z},\vec{i}-\vec{e}_n^{\,l},\vec{e}_n^{\,l})) = S(\vec{i})\beta\ h(\vec{i},\vec{x})\ f_l(\vec{i}),$$
$$\text{for} \quad S(\vec{x}) \in [0..s+r-2], \quad \vec{i} \in [0..s+r-S(\vec{x})]^n, \quad S(\vec{i}) = s+r-S(\vec{x}).$$

$$q((\vec{x},\vec{0},\vec{z},\vec{u},\vec{v}),(\vec{x},\vec{0},\vec{z},\vec{u}-\vec{e}_n^{\,l},\vec{v}+\vec{e}_n^{\,l})) = S(\vec{u})\beta\ f_l(\vec{u}),$$
$$\text{for} \quad S(\vec{x}) \in [0..s+r-2], \quad S(\vec{u}) \in [2..s+r-S(\vec{x})-1],$$
$$S(\vec{v}) = s+r-S(\vec{x})-S(\vec{u}).$$
$$q((\vec{x},\vec{0},\vec{z},\vec{e}_n^{\,l},\vec{v}),(\vec{x}+\vec{v}+\vec{e}_n^{\,l},\vec{0},\vec{0},\vec{0},\vec{0})) = \beta,$$
$$\text{for} \quad S(\vec{x}) \in [0..s+r-2], \quad S(\vec{v}) = s+r-S(\vec{x})-1.$$
$$q((\vec{x},\vec{0},\vec{z},\vec{0},\vec{0}),(\vec{x}+\vec{e}_n^{\,l},\vec{0},\vec{0},\vec{0},\vec{0})) = R(l)\beta,$$
$$\text{for} \quad S(\vec{x}) = s+r-1.$$

Note that $\vec{Q}$ 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 $\vec{Q}$ are

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

For illustration purposes, we depict in Fig. 3.3 some of the transitions of the absorbing CTMC when $n = 2$, $s = 3$, $r = 1$, and $k = 1$.

3.5.1 Data lifetime

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

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

The set $\mathcal{E}_I$ consists of all states of the process $\vec{\mathbf{W}}$ 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 $T(\mathcal{E}_I) := \inf\{t > 0 : \vec{W}(t) = a | \vec{W}(0) \in \mathcal{E}_I\}$, 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, $T(\mathcal{E}_I)$ will be referred to as the *conditional block lifetime*. We are interested in the conditional probability distribution function, $P(T(\mathcal{E}_I) \leq t)$, and the conditional expectation, $E[T(\mathcal{E}_I)]$, given that $\vec{W}(0) \in \mathcal{E}_I$ for $I \in [s..s+r]$.

From the theory of absorbing Markov chains, we can compute $P(T(\{\vec{w}\}) \leq t)$ where $T(\{\vec{w}\})$ is the time until absorption in state a given that the system initiates in state $\vec{w} \in \mathcal{T}$. We know that (e.g. [68, Lemma 2.2])

$$P(T(\{\vec{w}\}) \leq t) = 1 - \vec{e}_{|\mathcal{T}|}^{\,\mathrm{ind}(\vec{w})} \cdot \exp\left(t\vec{Q}\right) \cdot \vec{1}_{|\mathcal{T}|}, \quad t > 0,\ \vec{w} \in \mathcal{T} \tag{3.16}$$

where $\mathrm{ind}(\vec{w})$ refers to the index of state $\vec{w}$ in the matrix $\vec{Q}$. Definitions of vectors $\vec{e}_j^{\,i}$ and $\vec{1}_j$ were given at the end of Section 2.3.

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

$$\pi_{\vec{x}} := P\left(\vec{W}(0) = \vec{w} \in \mathcal{E}_I | \vec{W}(0) \in \mathcal{E}_I\right) = \binom{I}{x_1, \ldots, x_n} \prod_{l=1}^{n} R(l)^{x_l}. \tag{3.17}$$

Clearly $\sum_{\vec{w} \in \mathcal{E}_I} \pi_{\vec{x}} = 1$ for $I \in [s..s+r]$. Using (3.16) and (3.17) and the total probability theorem yields, for $I \in [s..s+r]$,

$$\begin{aligned} P(T(\mathcal{E}_I) \leq t) &= \sum_{\vec{w} \in \mathcal{E}_I} \pi_{\vec{x}}\, P\left(T(\{\vec{w}\}) \leq t\right) \\ &= 1 - \sum_{\vec{w} \in \mathcal{E}_I} \pi_{\vec{x}}\, \vec{e}_{|\mathcal{T}|}^{\,\mathrm{ind}(\vec{w})} \cdot \exp\left(t\vec{Q}\right) \cdot \vec{1}_{|\mathcal{T}|}, \quad t > 0. \end{aligned} \tag{3.18}$$

We know from [68, p. 46] that the expected time until absorption given that the $\vec{W}(0) = \vec{w} \in \mathcal{T}$ can be written as

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

The conditional expectation of $T(\mathcal{E}_I)$ is then (recall that the elements of $\mathcal{E}_I$ are of the form $(\vec{x}, \vec{0}, \vec{0}, \vec{0}, \vec{0})$)

$$\begin{aligned} E\left[T(\mathcal{E}_I)\right] &= \sum_{\vec{w} \in \mathcal{E}_I} \pi_{\vec{x}}\, E\left[T(\{\vec{w}\})\right] \\ &= -\sum_{\vec{w} \in \mathcal{E}_I} \pi_{\vec{x}}\, \vec{e}_{|\mathcal{T}|}^{\,\mathrm{ind}(\vec{w})} \cdot \left(\vec{Q}\right)^{-1} \cdot \vec{1}_{|\mathcal{T}|}, \quad \text{for } I \in [s..s+r]. \end{aligned} \tag{3.19}$$

3.5.2 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 $\mathcal{T}$, for $J \in [0..s+r]$,

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

The set $\mathcal{F}_J$ consists of all states of process $\vec{\mathbf{W}}$ 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 $\mathcal{T}$. We may define now

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

$T(\mathcal{E}_I, \mathcal{F}_J)$ is the total time spent by the CTMC in the set $\mathcal{F}_J$ before being absorbed in state a, given that $\vec{W}(0) \in \mathcal{E}_I$. Similarly, $T(\{\vec{w}\}, \{\vec{w}'\})$ is the total time spent by the CTMC in state $\vec{w}'$ before being absorbed in state a, given that $\vec{W}(0) = \vec{w}$. We know from [46, p. 419] that

$$E\left[T(\{\vec{w}\}, \{\vec{w}'\})\right] = -\vec{e}^{\,ind(\vec{w})}_{|\mathcal{T}|} \cdot \left(\vec{Q}\right)^{-1} \cdot {}^t\vec{e}^{\,ind(\vec{w}')}_{|\mathcal{T}|}, \qquad \vec{w}, \vec{w}' \in \mathcal{T} \tag{3.20}$$

Using (3.17) and (3.20), we derive for $I \in [s..s+r]$ and $J \in [0..s+r]$

$$\begin{aligned} E\left[T(\mathcal{E}_I, \mathcal{F}_J)\right] &= \sum_{\vec{w}' \in \mathcal{F}_J} E\left[T(\mathcal{E}_I, \{\vec{w}'\})\right] \\ &= \sum_{\vec{w} \in \mathcal{E}_I} \sum_{\vec{w}' \in \mathcal{F}_J} \pi_{\vec{w}} E\left[T(\{\vec{w}\}, \{\vec{w}'\})\right] \\ &= -\sum_{\vec{w} \in \mathcal{E}_I} \sum_{\vec{w}' \in \mathcal{F}_J} \pi_{\vec{w}} \vec{e}^{\,ind(\vec{w})}_{|\mathcal{T}|} \cdot \left(\vec{Q}\right)^{-1} \cdot {}^t\vec{e}^{\,ind(\vec{w}')}_{|\mathcal{T}|}. \end{aligned} \tag{3.21}$$

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

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

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

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

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

$$E\left[\frac{T(\mathcal{E}_I, \mathcal{F}_J)}{T(\mathcal{E}_I)}\right] \approx \frac{E[T(\mathcal{E}_I, \mathcal{F}_J)]}{E[T(\mathcal{E}_I)]}, \tag{3.22}$$

where the terms in the right-hand side have been derived in (3.21) and (3.19). We have already seen through simulation in Section 2.8 that such an approximation converges very well to the exact expectation with a very small relative error. With this approximation in mind, the two availability metrics become

$$M^{h}_{h,1}(\mathcal{E}_I) = \sum_{J=0}^{s+r} J \, \frac{E[T(\mathcal{E}_I, \mathcal{F}_J)]}{E[T(\mathcal{E}_I)]}, \quad \text{where } I \in [s..s+r], \tag{3.23}$$

$$M_{c,2}(\mathcal{E}_I, m) = \sum_{J=m}^{s+r} \frac{E[T(\mathcal{E}_I, \mathcal{F}_J)]}{E[T(\mathcal{E}_I)]}, \quad \text{where } I \in [s..s+r]. \tag{3.24}$$

3.6 Numerical results

In this section, we characterize the performance metrics defined in this chapter against the system parameters. In particular, we consider the simple and the extended models introduced in Sections 3.4 and 3.5 and we consider the *Condor* and the PlanetLab-like contexts whose peers parameters and characteristics are reported in Table 2.2 in the previous chapter. 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.

3.6.1 Parameter values

As we have seen in Section 2.11.1 ,The set *Condor* is best fit by a 2 stages hyper-exponential distribution according to the analysis in [69]. An exponential distribution is found to "reasonably" fit the *All-pairs-ping* data set in [75]. The descriptions of these data sets and their peers parameters and characteristics are reported in Table 2.2 in Section 2.11.1. We have solved numerically (3.10), (3.11), (3.14), (3.15), (3.18), (3.19), (3.23) and (3.24), given that all $s + r$ fragments of D are initially available, considering either Condor or PlanetLab context, using the same set of parameters values discussed and introduced in Section 2.11.1 of the previous chapter. We arbitrarily set the fragment upload rate value to $1/\beta = 6$ sec in the Condor context and $1/\beta \approx 21$ sec in the PlanetLab context. Results are reported partially in Table 3.1. 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 context considered, the distributed scheme yields a significantly smaller expected data lifetime than the centralized scheme, especially when the storage overhead, r/s, is high; cf. columns 3-4 in Table 3.1. The difference in performance is more pronounced in the Condor context. Regarding the expected number of available fragments, we again observe that the distributed scheme is less efficient than the centralized one. Observe how the performance deteriorates as peer churn becomes more important: compare for instance in Table 3.1 rows 6 vs. 20, and

Table 3.1: Expected lifetime and first availability metric: Centralized-repair scheme vs. Distributed-repair scheme

Condor context		$E[T^h_h(\mathcal{E}_{s+r})]$ (in days)		$M^h_{h,1}(\mathcal{E}_{s+r})$	
	$s=4$	cent. repair	dist. repair	cent. repair	dist. repair
$k=1$	$r=2$	0.48	5.99e-02	5.847	5.44
	$r=4$	19.49	0.244	7.737	6.761
	$r=5$	108.01	0.416	8.675	7.4
	$r=6$		0.656		8.01
	$r=10$	—	3.14	—	10.44
	$r=12$	—	8.123	—	11.41
$k=2$	$r=2$	0.109	1.92e-02	5.365	4.940
	$r=4$	5.11	9.62e-02	7.269	6.519
	$r=5$	31.044	0.187	8.225	7.290
$k=4$	$r=4$	0.186	1.86e-02	6.264	5.212
PlanetLab context		$E[T^e_h(s+r,0)]$ (in months)		$M^e_{h,1}(s+r,0)$	
	$s=8$	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

12 vs. 23 (these correspond to the same storage overhead and the same value of k). This is particularly true for the distributed recovery mechanism. We conclude that *when peers churn rate is high, only the centralized repair scheme can be efficient should the storage overhead be kept within a reasonable value (that is* $r/s \leq 2$*)*. As the distributed repair scheme *may involves* less upload traffics than the centralized one, it will be a good implementation choice in large networks where hosts have a good availability.

3.6.2 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 centralized and the context is similar to PlanetLab. We have picked one to two contour lines of each of the performance metrics studied in this paper and report them in Fig. 3.4. Consider point A which corresponds to $r = 6$ and $k = 1$ (recall $s = 8$). 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.

3.6.3 Impact of the size of blocks/fragments.

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 $P(T_h^e(s + r, 0) > 3 \text{ months}) = 0.89$ 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. We currently cannot say how does the bandwidth usage per recovery vary with the size of fragments. However, we know for sure that its effect will not be the same in both centralized and distributed schemes because of the additional upload stages in the centralized implementation. 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. A careful analysis of this issue is one objective of ongoing research.

3.7 Conclusions

We have proposed simple and general analytical models for evaluating the performance of two approaches for recovering lost data in distributed storage systems. Simple fluid model has been introduced under simple assumptions in order to have an explicit formula of the availability metric. We have analyzed the lifetime and the availability of data achieved by distributed-repair systems through markovian analysis considering realistic assumptions. 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. In contrast with the distributed-repair scheme in the case of highly dynamic environments, P2P storage systems with centralized-repair scheme (or "repair all missing fragment" policy) are efficient in any environment while the storage overhead is kept reasonable. Our analysis also suggests that the use of large size fragments reduces the efficiency of the recovery mechanism.

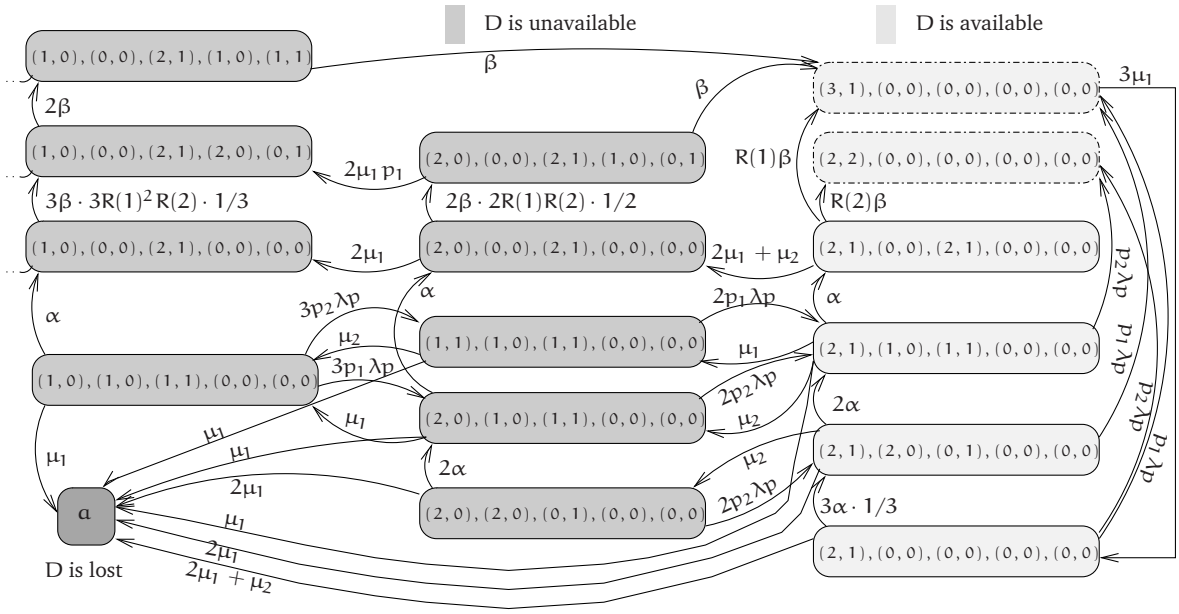


Figure 3.3: Some transitions of the Markov chain $\vec{\mathbf{W}}$ when $n = 2$, $s = 3$, $r = 1$, and $k = 1$.

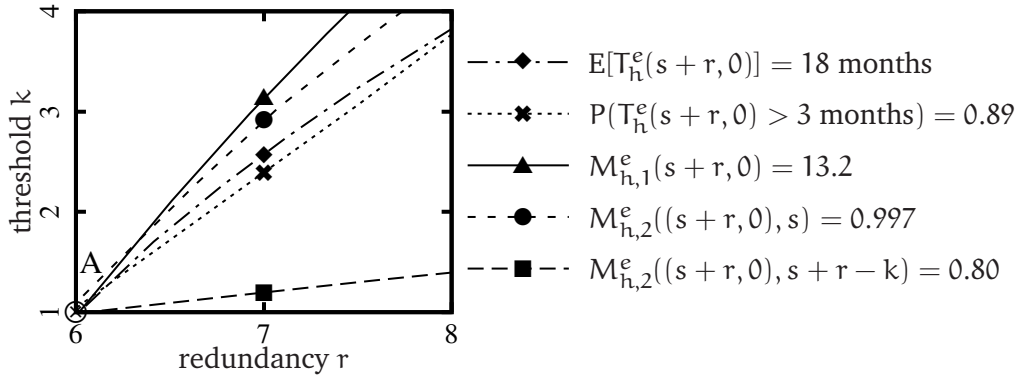


Figure 3.4: Contour lines of performance metrics (PlanetLab context, centralized repair).

4

Packet-level Simulation Model for Download and Recovery Processes

4.1 Introduction

The P2P paradigm has emerged as a cheap, scalable, self-repairing and fault-tolerant storage solution. We have shown in the first chapter, in Section 1.3, that recent modeling efforts, that address the performance evaluation of data lifetime and availability, have assumed the recovery process to follow an exponential distribution, an assumption made often because of the lack of studies characterizing the "real" distribution of the recovery process, and for the aim of simplification. This chapter aims at filling this gap and better understanding the behavior of these systems through simulation, while taking into consideration the impact of the heterogeneity of peers, the underlying network topologies, the propagation delays and the transport protocol. To that end, we implement the distributed storage protocol in the network simulator NS-2 [67] and run ten experiments covering a large variety of scenarios. As described in this chapter, our packet-level simulation model is realistic and captures the behavior of P2P storage systems. We show through experimental results how the recovery times distribution is impacted essentially by the inter-network links capacities, the volume of the background traffic, peers' bandwidth capacities, and the system workload. This distribution impacts, in turn, the modeling of data lifetime and availability as we have shown in the second and the third chapters.

We will distinguish between three general scenarios in which the download and the recovery processes have different distributions. In particular, in the first scenario, we show that the fragment download/upload time follows approximately an exponential distribution as long as

the total workload is equally distributed over the active peers, the core network has a good connectivity and the peers upload/download capacities are asymmetric. We have found that the successive download durations are weakly correlated in such a scenario. We also show that, as a consequence of the fragment download distribution and the weak coorelation, the block download time and the recovery time essentially follow a hypo-exponential distribution with many distinct phases (maximum of as many exponentials). In the second scenario, we will show that the fragment download/upload time follows a general phase type distribution but the block download time follows approximately an exponential distribution, under the following configurations: the peers upload/download capacities are symmetric and there are some bottlenecks in the backbone or among the local area network (LAN) routers. The characteristics of the third scenario are as follows. The peers are very heterogeneous and the volume of the non-P2P traffic is large with respect to the P2P traffic such that the total load in the system is very high. We found that, in such a scenario, both fragment and block successive download times are drawn from a general phase type distribution. Contrarily to the first scenario, the successive fragment download times in such a scenario are strongly correlated.

For all the experimental results in all the scenarios, we use expectation maximization and least square estimation algorithms to fit the empirical distributions. We also provide a good approximation of the number of phases of the hypo-exponential distribution that applies in several considered experiments. Last, we test the goodness of our fits using statistical (Kolmogorov-Smirnov test) and graphical methods.

The rest of this chapter is organized as follows. Section 4.2 introduces our motivation to do a simulation analysis with the packet-level simulator NS-2. Section 4.3 is devoted to the description of the model assumptions, selective implementation details and generating network topologies. In Section 4.4, we summarize the key settings of the experiments. In Section 4.5, we present the results of our simulations and the inference that we can draw from them. Last, Section 4.6 concludes this chapter.

4.2 Motivation

There have been recent modeling efforts focusing on the performance analysis of P2P backup and storage systems in terms of data durability and availability. In [75], Ramabhadran and Pasquale analyze backup systems that use *full replication* as redundancy mechanism. They develop a Markov chain analysis, then derive an expression for the lifetime of the replicated state and study the impact of bandwidth and storage limits on the system. This study relies on the assumption that the recovery process follows an exponential distribution. Observe that in replication-based systems, the recovery process lasts mainly for the download of one fragment of data that is equal to one block as the block here is not fragmented. In other words,

the authors of [75] are implicitly assuming that the fragment download time is exponentially distributed. In [28], Dalle et al. propose a stochastic model to characterize the expectation and the standard deviation of the data lifetime in a P2P backup system, that use *erasure coding* as redundancy mechanism, while assuming an exponential distribution on the recovery process.

In Sections 2.4 and 3.3 (appeared in [3]), we developed a more general model than that in [75], which applies to both replicated and erasure-coded P2P backup and storage systems. Also, unlike [75, 28], the model presented in [3] accounts for transient disconnections of peers, namely, the churn in the system. But we also assumed the recovery process to be exponentially distributed. However, this assumption can differ between replicated and erasure-coded systems, as in the latter systems the recovery process is much more complex than in the former systems. Furthermore, the recovery process differs from centralized to distributed recovery process implementation.

In all the models mentioned above, findings and conclusions rely on the assumption that the recovery process is exponentially distributed. However, this assumption is not supported by any experimental data. To the best of our knowledge, there has been no analytical or simulation study characterizing this process under realistic settings and assumptions.

It is thus essential to characterize the distribution of download and recovery processes in such systems. Evaluating these distributions is crucial to validate (or invalidate) some key assumptions made in some related studies. Moreover, simulation is critical to the building and better understanding of these systems with the presence of realistic topologies, underlying network protocols, underlying traffic, propagation delays and heterogeneous peers.

The main objective of this chapter is the description of the simulation model itself, and then the simulation analysis of download and recovery processes. The results show that (i) the fragment download time follows closely an exponential distribution and (ii) fragment download times are weakly correlated in some interesting scenarios. Given that in erasure-coded systems, the block download time consists of downloading several fragments in parallel, it follows that the recovery process should follow approximately a hypo-exponential distribution of several phases. (This is nothing but the sum of several independent random variables exponentially distributed having each its own rate [50]). We found that this is indeed the case in some interesting contexts. We realized that beside the fact that the total workload is equally distributed over the active peers, there are two main reasons for the weak correlation between concurrent downloads as observed in some scenarios: (i) the good connectivity of the core network and (ii) the asymmetry in peers upstream and downstream bandwidths. So, as long as the bottleneck is the upstream capacity of peers, the fragment download times are close to be independent.

Building on these results, we have incorporated into the models of Sections 2.4, 3.3 and 2.6 (appeared in [3, 29]) the assumption that fragment download and upload times respectively(instead of the block download times or the recover times) are exponentially distributed

with parameters α and β. The resulting models, whose descriptions are in Sections 2.5, 3.4, 2.7 and 3.5, characterize data lifetime and availability in P2P storage systems that use either replication or erasure codes, under more realistic assumptions in known scenarios.

4.2.1 Choice of Simulation

To collect traces of fragment download/upload times, of block download times and of recovery times, one can choose to perform simulations or experimentations either on testbeds or on real networks. We would like to consider situations where peers are either homogeneous or heterogeneous, different underlying network topologies, and different propagation delays in the network. Also, we would like to consider systems with a large number of peers. To achieve all this with experiments over real networks is very difficult. Setting up experiments over a dedicated network like Planet-Lab [74] would require a long time, and there will be limitations on changing the topology and the peers characteristics. In addition, measurement-based studies do not allow to evaluate performance in advance of building and deploying the system, hence the importance of simulations at reasonable scale for the thorough evaluation of P2P storage systems before their deployment.

4.2.2 Choice of NS-2

We find it most attractive to implement the distributed storage protocol in a well-known network simulator. We choose NS-2 as network simulator because it is an open source discrete event simulator targeted at networking research. NS-2 provides substantial support for simulation of TCP and routing and it is well known and well validated.

Note that in view of the backup and storage systems specification and objectives which are different, for instance, from file sharing, involving some hundreds to some thousands of peers in a simulation has to conclude realistic results and helps to understand the system behavior. However, this simulator can be used to validate principle algorithms or processes of a flow-level simulators (e.g. the flow-level algorithm presented in Chapter 5) that probably will neglect many factors such as the underlying network protocol, in order to have more scalable simulations where a very large of nodes are to be involved which is not feasible in a Packet-level simulator.

4.3 Simulation Assumptions and Network Topology

This section introduces the assumptions made in the simulation model and overviews the hierarchical structure of the network topologies used in the simulations. We implemented

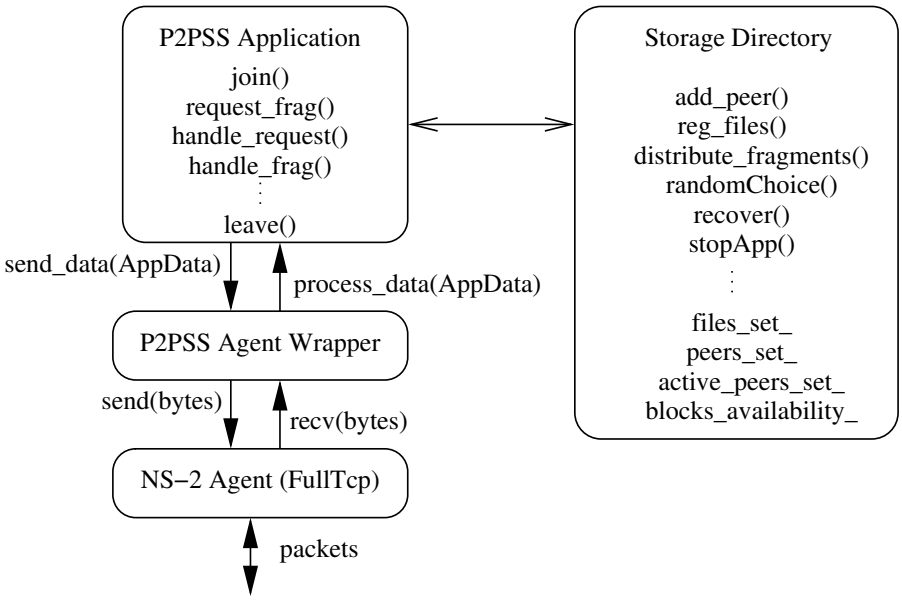


Figure 4.1: Simulator architecture.

the P2P storage application in NS-2 (versions 2.29 and 2.33) following the architecture depicted in Fig. 4.1. We will describe the base classes (P2P_Storage_Directory, P2P_Storage_App, P2P_Storage_Wrapper and data structure) and other implementation details in Appendix A.

We assume that there is a given number of stored files in the system and before that peers request data, the system directory object distributes the $s + r$ fragments of each block of data of all files over $s + r$ peers chosen uniformly among all the registered peers in the system. In fact, a DHT-like systems (e.g. [83, 54, 89]) do not choose randomly peers in the network to store the fragments of each block but the distribution of data depends on the identifier space, the identifier of each node (its position in the space) and the hash value of the block itself. However, it is proved (e.g. in [83, 54, 89]) that DHT makes the number of keys per node uniformly distributed with high probability. In other words, with high probability each node is responsible for $O(1/N)$ of the identifier space where N is the number of peers. It is proved as well that the cost of the lookup phase (e.g. in Chord-like protocol in [89]) grows as the logarithm of the number of nodes. As a result, and in view of the fact that we involve some hundreds to some thousands of nodes, we neglect the lookup cost with respect to the download or recovery times and we do not implement DHT to reduce the complexity. In other words, we use the same class of the system directory for both recovery process implementations and we assume that the system has a perfect knowledge of the data state.

We consider two different storage applications, a backup-like application and an e-library-like application ("e" stands for "electronic"). In the first, a file stored in the system can be

requested for retrieval only by the peer that has produced the file. In the second, any file can be downloaded by any peer in the system. In both applications, the storage protocol follows the description presented in Sections 2.2 and 3.2.

Two types of requests are issued in the system. The first type is issued by the users of the system: a user issues a request to retrieve its backup file in the backup-like application, or a public document in the e-library-like application. The second type consists of management requests. Usually, these are issued by the central authority (in the centralized implementation of the recovery process) or by a peer (in the distributed implementation) as soon as the threshold k is reached for any stored block of data. In the simulator, these management requests are issued by the system storage directory object.

File download requests are translated into (i) a request to the directory service to obtain, for each block of the desired file, a list of at least s peers that store fragments of this block, (ii) opening TCP connections with each peer in the said list to download one fragment, (iii) registering some statistical information such as the start and the completion time of the downloaded data. All download requests issued by a given peer form a Poisson process. This assumption is met in real networks as found in [47]. However, another random process can easily implemented.

Recovery requests are issued only in the scenarios where there is churn in the network. A recovery request concerning a given block translates into (i) a request from the storage directory service to a server in the centralized-repair scheme (we consider explicitly the first registered peer as the server in order to simulate the centralized implementation) or any active peer that is in charge of (ii) obtaining a list of at least s peers that store fragments of said block, (iii) opening TCP connections with each peer in the said list to download one fragment. Once all s fragments have been downloaded, the process proceeds with Steps 2 and 3, according to the implementation, as explained in Sections 2.2 and 3.2. Last, the storage directory updates the system state at the end of the operation, namely it increases the availability level of the blocks of interest and points to the right locations of its fragments or otherwise it adds the lost block in a black list if the operation failed.

Typically, applications access network services through sockets. NS-2 provides a set of well-defined API functions in the transport agent to simulate the behavior of the real sockets. Therefore, the P2P_Storage_Wrapper class handles calling the appropriate APIs when two applications want to communicate in order to (i) attache first the Full Tcp agent to both NS nodes via attach-agent and (ii) call then connect() instproc to set each agent's destination target to the other and last (iii) place one of them in LISTEN mode. We use in fact Full-Tcp agents since they support bidirectional data transfers.

Similar to what is done in the web cash application (see tcpapp.cc) we can model the underlying TCP connections as a FIFO byte stream, and then we will create same buffer management stuff. First, the P2P_Storage_Ms_Buf that contains a part of the messages such as the

Table 4.1: The basic prototypes of P2P_Storage_Msg_BufList class

Method	Functionality
void insert(P2P_Storage_Msg_Buf *d)	stores msgs of the sender until the reception of their acks
P2P_Storage_Msg_Buf* detach()	if the data is received by the destination, deletes them from the FIFO buffer
int size()	returns the current size of the buffer

Request message and the Fragment message. Second, P2P_Storage_Msg_BufList implements a FIFO queues that will store all the sent messages (requests or data) on the sender side until they correctly and completely arrive to the destination side. In other words, there is no support in the class "Agent" to transmit different applications data and messages. Instead, as all data are delivered in sequence, we can view the TCP connections as a FIFO pipes, and the transfer of the application data will be emulated as follows. We first provide buffer for the application data at the sender to store the messages to be sent, next we use the Agent's API "sendmsg(int nbytes, const char *flags = 0)" to send a stream of an equivalent data size of the stored messages, then we count the bytes received at the destination. When the receiver has got all bytes of the current data or message transmission (first message in the FIFO on the sender side toward the receiver), then the receiver gets the data directly from the FIFO's sender. The prototypes of the FIFO queues depicted in Table 4.1, where a FIFO queue is represented by the P2P_Storage_Msg_BufList class.

4.3.1 Network Topology

Having a representative view of enterprise networks or the Internet topology is very important for a simulator to predict the behavior of a network protocol or application if it were to be deployed. In fact, the simulated topology often influences the outcome of the simulations. Realistic topologies are thus needed to produce realistic simulation results. Most of existing simulation studies have used representations of a real topology (e.g. the Arpanet), simple models (e.g. a star topology), or random flat graphs (i.e. non-hierarchical) that are generated by Waxman's edge-probability function [96].

However, random models offer very little control over the structure of the resulting topologies. In particular, they do not capture the hierarchy that is present in the Internet. Recently, tools such as BRITE [65] and GT-ITM [18] have been designed to generate more complex random graphs, that are hierarchical, to better approximate the Internet's hierarchical structure.

To produce realistic topologies for our simulations, we use the tool GT-ITM [18] to generate

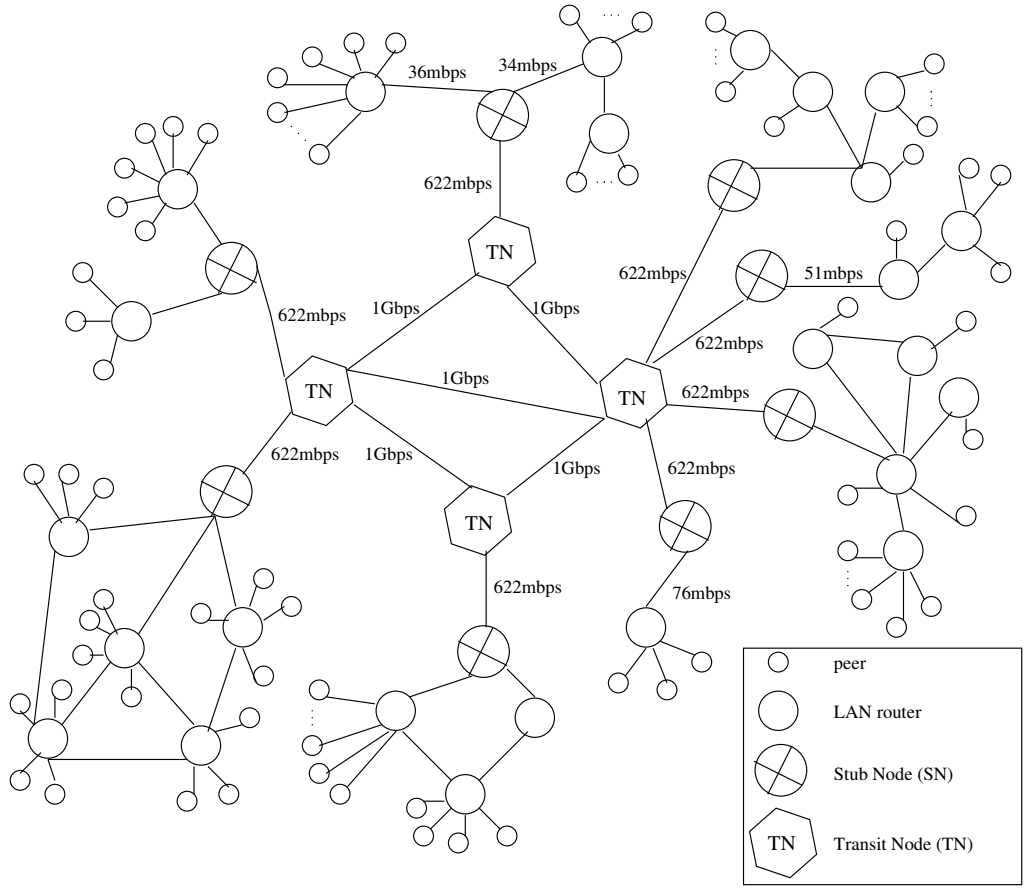


Figure 4.2: Three-level hierarchical random graph of Experiment 1.

random graphs. Three levels of hierarchy are used corresponding to transit domains, stub domains, and local area networks (LANs) attached to stub domains. Each graph has one transit domain of four nodes; each of the nodes is connected to two or three other transit nodes. Each transit node is connected on average to two stub nodes, and each stub node is in turn connected on average to four routers. Behind every router there is a certain number of fully-connected peers constituting a LAN. An example of these random graphes is that used in Experiment 1 which is depicted in Fig. 4.2, where we have used the notation TN for “transit node” and SN for “stub node”.

4.4 Experiments Setup

We ran a total of ten experiments. Experiments 1–5, 7 and 9–10 used the random graphs generated with the GT-ITM tool as detailed before, whereas a simple star topology is used

in Experiments 6 and 8. Regarding the intra- and inter-domain capacities, we rely on the information provided by RENATER [79] and GÉANT [44] web sites. In those networks, the links are well-provisioned. To have a more complete study, we will consider, in Experiments 4,5 and 10, links with smaller capacities, as can be seen in rows 4–6 of Table 4.2. Propagation delays over TN-SN edges vary from edge to edge as can be seen in row 7 of Tables 4.2.

Let C_u and C_d denote respectively the upload and download capacity of a peer. To set these values, we rely mainly on the findings of [47] and [53]. The experimental study of file sharing systems and of the Skype P2P voice over IP system [47] found that more than 90% of users have upload capacity C_u between 30Kbps and 384Kbps. However, the measurement study [53] done on BitTorrent clients in 2007 reports that 70% of peers have an upload capacity C_u between 350Kbps and 1Mbps and even 10% of peers have an upload capacity between 10Mbps and 110 Mbps. The capacities that we have selected in the simulations vary between the values of the ISDN and ADSL technologies; they can be found in rows 8–9 of Table 4.2. Observe that, except in Experiments 7,8 and 9, peers are heterogeneous. We will attribute, except in Experiment 8, the propagation delays over routers-peers edges randomly between 1ms and 10,20,25 1nd 150ms as can be seen in row 10 of Table 4.2.

In Experiments 1, 2, 4–5 and 9 (resp. Experiments 10), there exists a background traffic between three pairs of routers (resp. ten pairs of routers) across the common backbone. This traffic consists of random exponential and CBR traffic over UDP protocol and FTP traffic over TCP.

In each of the experiments, the amount of data transferred between routers and peers in the system during the observed time (that is from 4e+5 up to 6e+6 seconds) are, on average, 4.5–12 GB of P2P application traffic, and when applicable except in Experiment 10, 150–500 MB of FTP, 200–600 MB of CBR, and 250–900 MB of the exponential traffic. In Experiment 10 the amount of P2P application traffic is 15G.B for 8e+6 seconds simulation times and on average 1G.B of FTP, 1.5 G.B of CBR, and 800 MB of the exponential traffic.

Experiments 2 and 4 simulate a backup-like application whereas the other experiments simulate an e-library-like application. Churn is considered only in Experiments 4–6. As a consequence, redundancy is added and maintained only in these experiments. The storage overhead r/s is either 1 or 0.5. We consider the distributed implementation of the recovery process in Experiments 4 and 5, and the centralized implementation of the same in Experiment 6; the eager policy ($k = 1$) is considered in all three experiments. In other words, once a peer disconnects from the system, all fragments that are stored on it must be recovered.

Churn is implemented as follows. We assume that each peer alternates between a connected state, that lasts for a duration called “on-time”, and a disconnected state, that lasts for a duration called “off-time”. We assume in the simulations that the successive on-times (respectively off-times) of a peer are independent and identically distributed random variables with a

common exponential distribution function with parameter $\mu_1 > 0$ (respectively $\mu_2 > 0$). This assumption is in agreement with the analysis in [75]. We consider $1/\mu_1 = 3$ hours and $1/\mu_2 = 1$ hours.

Download requests are generated at each peer according to a Poisson process. This assumption is met in real networks as found in [47]. We assume all peers have the same request generation rate, denoted λ. We vary the value of λ across the experiments as reported in row 16 of Table 4.2.

The last setting concerns the files that are stored in the P2P storage system. Fragment sizes S_F (resp. block sizes S_B) in P2P systems are typically between 64KB and 4MB each (resp. between 4MB and 9MB each). We will consider in most of our experiments $S_F = 1$ or 2MB and $S_B = 8$MB, except in Experiment 4 where $S_F = 512$KB and $S_B = 4$MB. Therefore $s = 8$ in Experiments 1–7 and $s = 4$ in Experiments 8–10. As for the file size, we assume for now that it is equal to the block size. Therefore, the file download size is actually the block download size. We leave the case of more general file sizes to a future study. In fact, we consider this because the recovery process is related to the block download time and not to the file download time as already explained in Sections 1.2.1, 2.2 and 3.2.

Table 4.2 summarizes the key settings of the experiments.

Similarly to any simulation that uses NS, we define the network topology and the system parameters (such as maximum number of files, maximum number of peers, upload/download capacities, delays, block size, fragment size, TCP segemt size, amount of redundancy and the recovery threshold) at OTcl level in a TCL script (an example is provided in Appendix A).

4.5 Experimental Results

In this section, we present the results of our simulations and the inference that we can draw from them. For each experiment, we collect the fragment download time, the block download time and the recovery time when applicable. In Experiments 4 and 5 (distributed recovery), the two latter durations are collected to the same dataset as there is no essential difference between them. Having collected these samples, we compute the sample average and use MLE, LSE and EM algorithms to fit the empirical distributions and we perform the Kolmogorov-Smirnov test [62] on the fitted distribution. In the following, we will present selected results from Experiments 1, 5–6 and 8–10. The results of the other experiments are briefly reported in Tables 4.3–4.4.

4.5.1 Experiment 1

We have collected 76331 samples of the fragment download time (cf. column 2 of Table 4.3). The empirical cumulative distribution function (CDF) is depicted in Fig. 4.3(a). We can

see that it is remarkably close to the exponential distribution. Two exponential distributions are plotted in Fig. 4.3(a), each having a different parameter, derived from a different fitting technique. The two techniques that we used are MLE and LSE. The parameter returned by MLE is nothing but the inverse of the sample average and is denoted α; see row 2 of Table 4.3.

Beyond the graphical match between the empirical distribution and the exponential distribution, we did a hypothesis test. Let $\mathbf{X}$ be a vector storing the collected fragment download times. The Kolmogorov-Smirnov test compares the vector $\mathbf{X}$ with a CDF function, denoted `cdf` (in the present case, it is the exponential distribution), to determine if the sample $\mathbf{X}$ could have the hypothesized continuous distribution `cdf`. The null hypothesis is that $\mathbf{X}$ has the distribution defined in `cdf`, the alternative one being that $\mathbf{X}$ does not have that distribution. We reject the null hypothesis if the test is significant at the l% level. In Experiment 1, the null hypothesis with $\alpha = 1/40.35$ is not rejected for $l = 7$%.

Regarding the block download times, we have collected 9197 samples. The sample average is given in row 7 of Table 4.3). The empirical CDF is plotted in Fig. 4.3(b). We followed the same methodology and computed the closest exponential distribution using MLE. However, the match between the two distribution appears to be poor, and actually, the alternative hypothesis is not rejected in this case.

Of course, a general Phase-type distribution (more than two inter-related exponentials occurring in sequence, and/or in parrallel) can perfectly fit the collected data; where its parameters can be determined using an EM algorithm [34] (e.g. *EMpht* [71]). However, we would like to find a distribution that on the one hand will likely fit the empirical data, and on the other hand, will allow us to model such systems using some performance evaluation tools such as *Markov chains*. To that end, we make the following analysis. To get a block of data, s fragments, stored on s different peers, have to be downloaded. This is more efficiently done in parallel and this is how we implemented it in the simulator. We have seen that the download of a single fragment is well-modeled by an exponential random variable with parameter α.

Moreover, we have found that the concurrent downloads/uploads 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 [47]. So, as the bottleneck would be the upstream capacity of peers, the fragment download times are close to be iid rvs. Therefore, downloading s fragments in parallel is distributed like the maximum of s exponential random variables. Assuming these downloads to be mutually independent—assumption not necessarily met in the simulations—the maximum is then the sum of s independent exponential random variables with parameters $s\alpha, (s-1)\alpha, \ldots, \alpha$, due to the memoryless property (see also [50]).

This distribution is called the hypo-exponential distribution and its expectation is

$$E[T] = 1/\alpha \sum_{i=1}^{s} 1/i \tag{4.1}$$

where T denotes the block download time (or equivalently the distributed recovery duration).

In Experiment 1, $E[T] = 109.66$ seconds, while the sample average is equal to 102.75; cf. column 2 of Table 4.4. The relative error is 6.7%. The hypo-exponential distribution with s phases and parameters $s\alpha, (s-1)\alpha, \ldots, \alpha$ is plotted in Fig. 4.3(b). This distribution has a very good visual match with the empirical CDF of the block download time.

As a next step, we apply an EM algorithm [34] to find the best hypo-exponential distribution with s phases that fits the empirical data. In particular, we use *EMpht* [71], which is a program for fitting phase-type distributions to collected data. We do not plot the outcome of this program in Fig. 4.3(b) as it roughly overlaps with the hypo-exponential distribution with s phases and parameters $s\alpha, (s-1)\alpha, \ldots, \alpha$ that is already plotted there using the *PHplot.m* program associated with *EMpht*. After performing the Kolmogorov-Smirnov test, we find that the null hypothesis is not rejected for $l = 7\%$ (same significance level as for the fragment download times).

We conclude the analysis of the first experiment's results with three important points:

- The exponential assumption on the block download time is not met in realistic simulations.
- The fragment download time could be modeled by an exponential distribution with parameter α equal to the inverse of its average, and these download times are weakly correlated and close to be independent in an "Experiment 1"-like environment (similar peers and network configurations).
- As a consequence, the block download time could be modeled by a hypo-exponential distribution with s phases and parameters $s\alpha, (s-1)\alpha, \ldots, \alpha$.

4.5.2 Experiment 5

In this experiment, peers are not always connected. Each time a peer disconnects from the network, all the fragments that were stored on his disk will have to be recovered. The recovery process is implemented in a distributed way.

The empirical CDF of the fragment download time and that of the block download time or the recovery time are reported in Fig. 4.4. Following the same methodology as that used to analyze the results of Experiment 1, we reach the same conclusions. The relevant parameters are

reported in column 7 of Tables 4.3 and 4.4. However, the null hypothesis for the block download time or the recovery process is not always "not rejected". This is the case of Experiment 6, as seen next.

4.5.3 Experiment 6

Experiment 6 is the only one that uses a centralized recovery process. Also, it uses a simple star topology. In this experiment, the alternative hypothesis on the fragment download (resp. recovery) processes distribution is not rejected, even if graphically the exponential (resp. hypo-exponential) distribution fits reasonably the collected data.

There is a simple reason for that. We actually know that the download of a single fragment cannot be infinitely small, as suggested by the exponential distribution. Let t_m be the duration of the fastest fragment download among all s downloads. All other (slower) downloads are necessarily bounded by t_m. The effect of this minimum value can be neglected as long as t_m is negligible with respect to the average fragment download time. Otherwise, we need to consider that the fragment download/upload time is composed of two components: (i) a (constant) minimum delay t_m and (ii) a random variable distributed exponentially with parameter $\hat{\alpha}$ (resp. $\hat{\beta}$). This random variable models the collected data, shifted left by the value of t_m. The minimum delay can be approximated as $RTT + (S_F + \text{ Headers})/\max\{C_u\}$, where RTT stands for round-trip time.

The value of t_m is clearly visible in Fig. 4.4(c). We plot in this figure the empirical CDF of the fragment download time, the MLE exponential fit to the collected data and the MLE exponential fit to the shifted data. The null hypothesis is not rejected in the later case but is rejected in the former one (for the non-shifted data). This is the same case of the recovery process, whose empirical CDF is plotted in Fig. 4.4(d). Repeating the same analysis than in Section 4.5.1, and assuming that the fragment upload time follows an exponential distribution with parameter β, then the centralized recovery process, denoted T_c, would be modeled by a hypo-exponential distribution with $s + k$ phases ($k = 1$ in Experiment 6) having expectation

$$E[T_c] = 1/\alpha \sum_{i=1}^{s} 1/i + 1/\beta \sum_{j=1}^{k} 1/j\,. \tag{4.2}$$

Considering this distribution, we find that the null hypothesis of the Kolmogorov-Smirnov test for the collected data with parameters $1/\alpha = 40.72$ and $1/\beta = 6.22$ is rejected[1] for $l = 7\%$, while it is not rejected for the shifted data with parameters $1/\hat{\alpha} = 32.05$ and $1/\hat{\beta} = 5.11$.

[1]Even though it is rejected, this distribution is still much closer to the empirical data than the exponential distribution.

Equations (4.1) and (4.2) should then be replaced with

$$E[T] = t_m + 1/\hat{\alpha} \sum_{i=1}^{s} 1/i\,, \tag{4.3}$$

$$E[T_c] = t_m + 1/\hat{\alpha} \sum_{i=1}^{s} 1/i + 1/\hat{\beta} \sum_{j=1}^{k} 1/j\,. \tag{4.4}$$

The averages inferred from Eqs. (4.1)–(4.4) are listed in rows 3 and 5 of Table 4.4, and their relative errors with respect to the sample average are listed in rows 4 and 6 of the same table. Observe that the inferred average improves across all experiments when considering shifted data. The best improvement seen is that in Experiment 6. By considering that the *shifted* recovery time is hypo-exponentially distributed with $s+1$ phases and parameters $s\hat{\alpha}, (s-1)\hat{\alpha}, \ldots, \hat{\alpha}, \hat{\beta}$, the relative error on the inferred average drops from 30.1% to 2.6%.

The conclusion of this discussion is that the exponential assumption on fragments download/upload time is met in most cases. The same assumption does not hold on the block download time. The recovery time and the block download time are well approximated by a hypo-exponential distribution in "Experiments 1–6"-like environment (similiar peers/network configuration).

4.5.4 Experiments 8 and 9

In experiment 8, peers are homogeneous while they are hetregeneous in Experiment 9 and in both experiments, peers are always connected. The system workload is relatively big and peers's download/upload capacities are symmetric in Experiment 8 and close to be symmetric in Experiment 9 in such a way that the bottleneck can be the upstream or the downstream capacity of peers. In Experiment 9, there are background traffic. The concurrent fragment download processes are not independent but correlated. We see from Figures 4.5(a) and 4.5(a) that the fragment download time is remarkably not exponentially distributed in such configurations, even for the *shifted* data, but it follows a phase type distribution unlike the case of Experiments 1–6.

Regarding the block download time, we plot in Figures 4.5 the empirical CDF of the data download time and the MLE exponential fits to the *shifted* data. It is remarkable that the exponential distribution fits well the data distribution. In fact, the null hypothesis is rejected for the collected data but not rejected for the shifted data with $l = 6\%$ in Experiment 8 and with $l = 8\%$ in Experiment 9.

4.5.5 Experiment 10

In experiment 10, peers are very heterogeneous and always connected. There are an important volume of the background traffic, between ten pairs of LAN routers across the transit domain, with respect to the volume of the P2P application traffic so that the system workload is relatively big and peers's download capacities are twice bigger than the upload capacities. As a result, several bottlenecks can be created in the core of the network or at the end-users (peers). The concurrent and the successive fragment download time are strongly correlated. We see from Figure 4.6(a) that the fragment download time is remarkably not exponentially distributed in such configurations, even for the *shifted* data, but it follows a general phase type distribution. In fact, the null hypothesis is rejected for the collected data and for the shifted data even for $l = 10\%$

Moreover, concerning the block download process, we plot in Figure 5.6(b) the empirical CDF of the data download time and the MLE exponential fits to the *shifted* data and the hypo-exponential fit of s phases of the *shifted* data as well based on the MLE parameter of the fragment download time. It is remarkable that neither the exponential distribution nor the hypo-exponential fits match the collected data distribution. In fact, in such a scenario, modeling the P2P storage systems using the mathematical framework presented in Chapters 2 and 3 yields to under-estimation or over-estimation of the data lifetime and availability. However, such scenarios are not desired in nowadays networks and applications [73, 91, 40]. The Internet service providers have been divided into two groups: *infrastructure providers*, who manage the physical infrastructure, and *service providers*, who create virtual networks by aggregating resources from multiple infrastructure providers and offer end-to-end services to the end users. Such environment enables diverse network architectures to cohabit on a shared physical substrate. And so, we can imagine that the P2P storage application will have in distribution a predefined end-to-end bandwidth capacities where bottlenecks in the core of the network are rarely appeared.

4.6 Conclusions

This chapter describes a realistic simulation model of the P2P storage system and sketches its implementation on top of the Network Simulator NS (versions 2.29 and 2.33). We perform a simulation analysis of the download and recovery processes in P2P backup and storage systems. We set up ten simulations which enables us to collect fragment/block download times and recoveries times under a variety of conditions. We show that the exponential assumption on the block download time can be hold in some scenarios like Experiments 8 and 9. The same assumption on fragments download/upload time is met in most considered contexts implying

that both the block download time and the recovery process could be modeled by a hypo-exponential distribution with a pre-determined number of phases. The results of this chapter support key assumptions made in the models presented in Chapters 2 and 3.

Table 4.2: Experiments setup

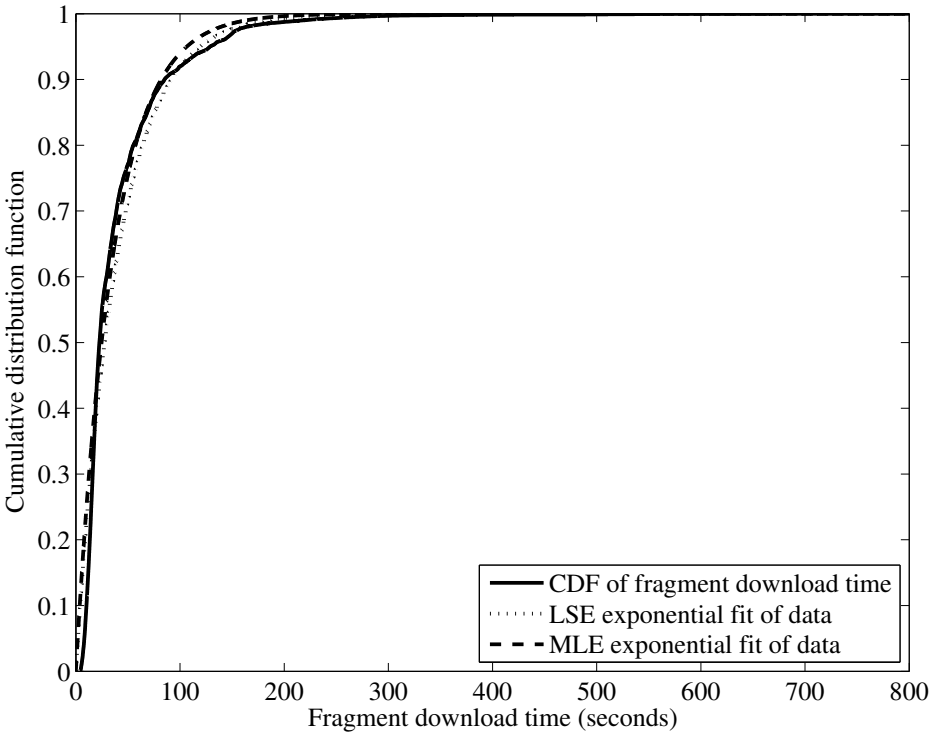
Experiment number	1	2	3	4	5
Topology	random	random	random	random	random
Number of peers	960	480	960	640	800
TN-TN capacities (Gbps)	1	1	1	1	1
TN-SN capacities (Mbps)	622	622	622	10–34	34–155
SN-routers capacities (Mbps)	34–155	34–155	34–155	4–10	10–34
TN-SN delays (ms)	5–25	5–75	5–50	5–25	5–25
C_u of peers (Kbps)	150–1000	128–1000	128–1000	256–700	256–1000
C_d of peers (Kbps)	$8 \times C_u$	$8 \times C_u$	$8 \times C_u$	$10 \times C_u$	$4 \times C_u$
routers-peers delays (ms)	1–20	1–20	1–20	1–10	1–25
Background traffic	yes	yes	no	yes	yes
Application type	e-library	backup	e-library	backup	e-library
Peers churn	no	no	no	yes	yes
Recovery process	—	—	—	dist.	dist.
r	—	—	—	s	s
$1/\lambda$ (min.)	80	80	144e-3	160	13
S_B (MB)	8	8	8	4	8
S_F (KB)	1024	1024	1024	512	1024
s	8	8	8	8	8
Experiment number	6	7	8	9	10
Topology	star	random	star	random	random
Number of peers	480	480	250	1225	1500
TN-TN capacities (Gbps)	—	1	—	1	1
TN-SN capacities (Mbps)	—	622	—	622	34–155
SN-routers capacities (Mbps)	—	34–155	—-	34–155	4–10
TN-SN delays (ms)	—	5–50	—	1–10	1–10
C_u of peers (Kbps)	256–700	256	384	512	30–1000
C_d of peers (Kbps)	2048	512	384	758	$2 \times C_u$
routers-peers delays (ms)	1–25	1–20	2	1–10	1–150
Background traffic	no	no	no	yes	yes
Application type	e-library	e-library	e-library	e-library	e-library
Peers churn	yes	no	no	no	no
Recovery process	cent.	—	—	—	—
r	$s/2$	—	—	—	—
$1/\lambda$ (min.)	16	8	1/60	1e-3	1.3e-4
S_B (MB)	8	8	4	8	8
S_F (KB)	1024	1024	1024	2048	2048
s	8	8	4	4	4

Table 4.3: Summary of experiments results

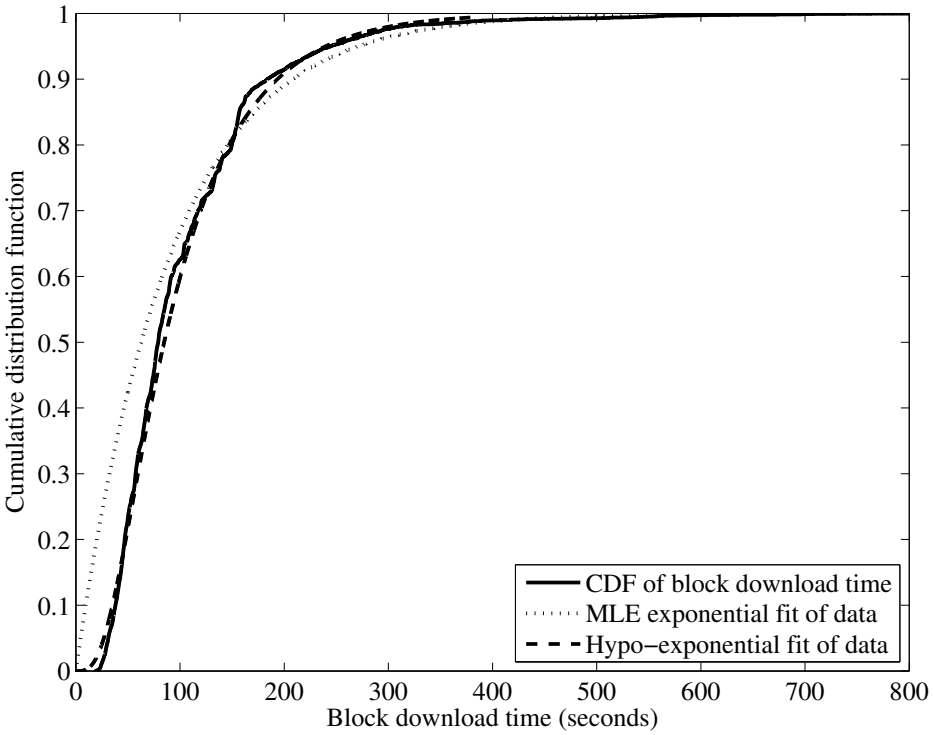
Experiment number	1	2	3	4	5
Average frag. down. time $= 1/\alpha$ (sec.)	40.35	44.89	30.66	34.74	108.86
Samples number	76331	12617	4851	9737	80301
t_m (sec.)	8.77	8.63	8.71	6.84	8.74
$1/\hat{\alpha}$ (sec.)	39.35	39.62	27.34	32.11	103.64
Av. of recovery or block down. time (sec.)	102.75	105.25	82.88	92.48	278.71
Samples number	9197	1516	602	589	10025
Experiment number	6	7	8	9	10
Average frag. down. time $= 1/\alpha$ (sec.)	40.722	141.89	135.87		
Samples number	4669	71562	37200		
t_m (sec.)	16.4	33.71	25.377		
$1/\hat{\alpha}$ (sec.)	32.05	124.61	110.49		
$1/\beta$, $1/\hat{\beta}$ (sec.)	6.22, 5.11	—	—	—	—
Av. of recovery or block down. time (sec.)	89.85	365.73	205.19		
Samples number	561	8938	9300		

Table 4.4: Block download time or recovery process: Validation of the approximations introduced in Eqs. (4.1)–(4.3)

Experiment number	1	2	3	4	5
Sample average	102.75	105.25	82.88	92.48	278.71
Inferred average from Eqs. (4.1), (4.2)	109.66	122.00	83.33	94.40	295.86
Relative error (%)	6.7	15.9	0.5	2.1	6.2
Inferred average from Eqs. (4.4), (4.3)	106.95	116.32	83.01	94.10	290.41
Relative error (%)	4.1	10.5	0.2	1.8	4.2
Experiment number	6	7	8	9	10
Sample average	89.85	365.73	205.19		
Inferred average from Eqs. (4.1), (4.2)	116.89	385.64	283.05		
Relative error (%)	30.1	5.4	37.95		
Inferred average from Eqs. (4.4), (4.3)	92.21	372.38	255.56		
Relative error (%)	2.6	1.8	24.55		

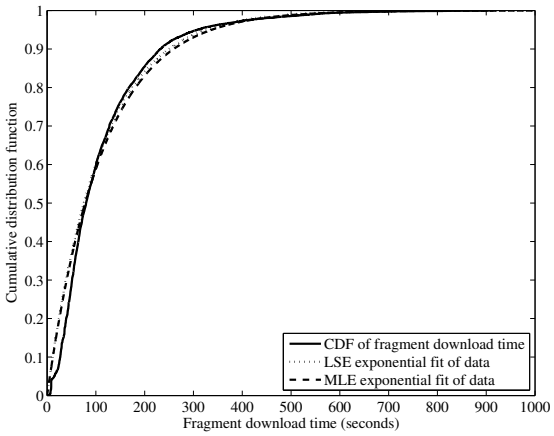


(a) Exponential fit of the fragment download time distribution

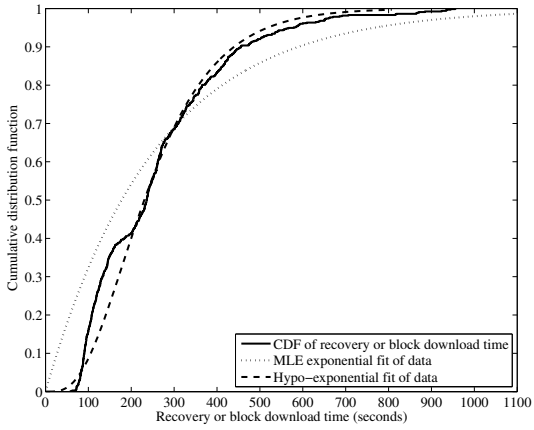


(b) Fitting of the block download time distribution

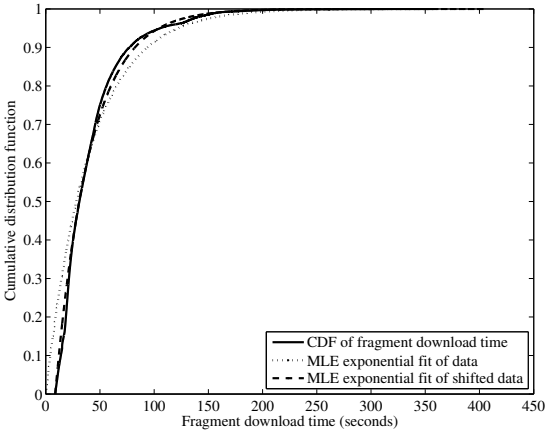
Figure 4.3: Experiment 1: Fragment and block download times.



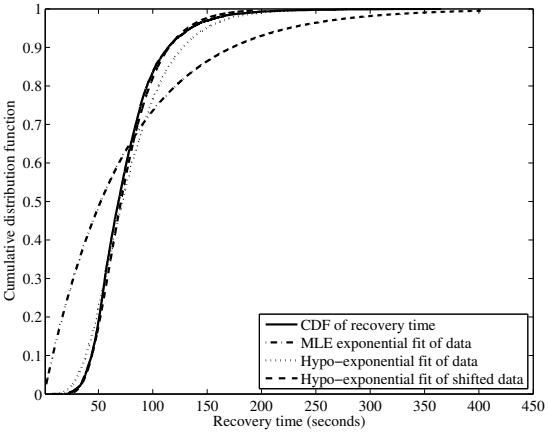
(a) Exponential fit of the fragment download time distribution



(b) Fitting of the recovery or block download time distribution

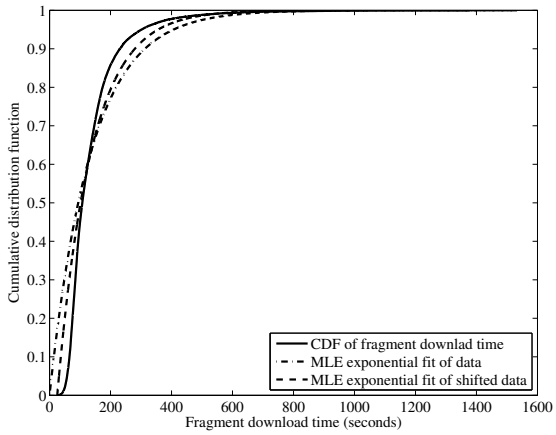


(c) Exponential fit of the fragment download time distribution

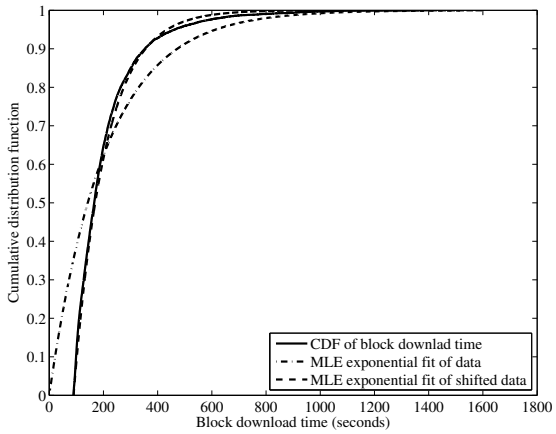


(d) Fitting of the recovery time distribution

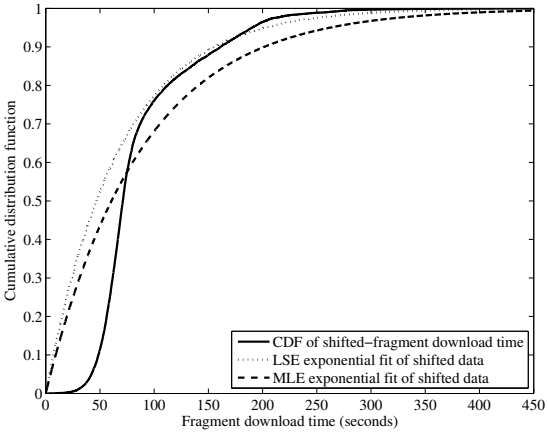
Figure 4.4: Experiment 5 (top a+b): Download and distributed recovery processes, Experiment 6 (down c+d): Fragment and recovery time, centralized recovery.



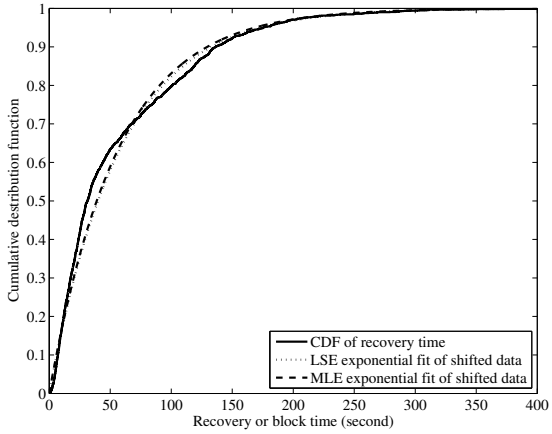
(a) Exponential fit of the fragment download time distribution



(b) Fitting of the block download time distribution

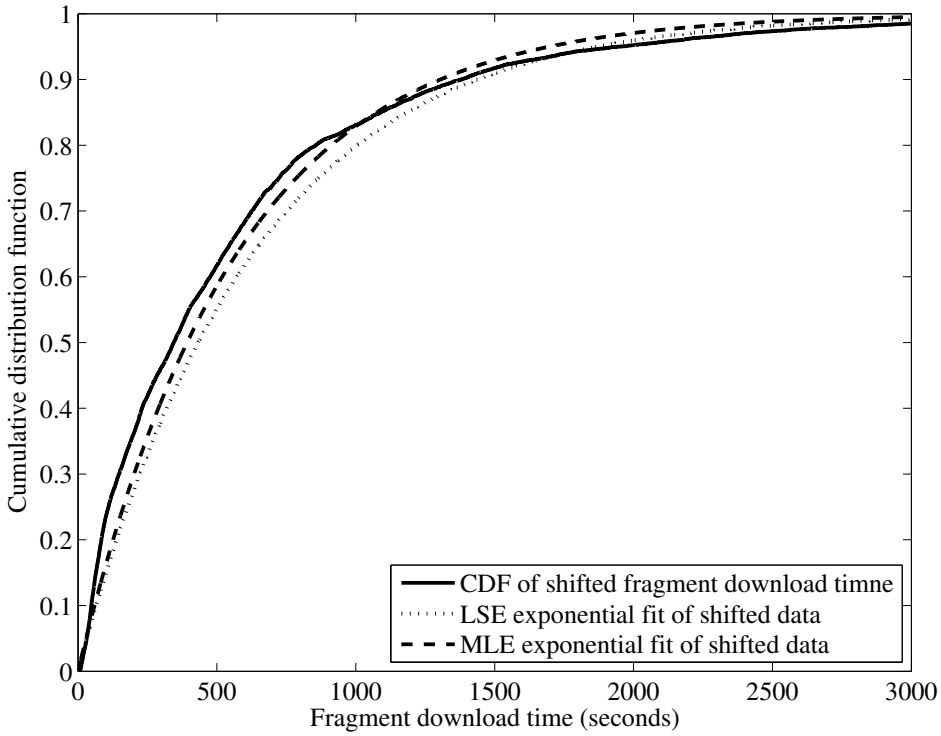


(c) Exponential fit of the fragment download time distribution

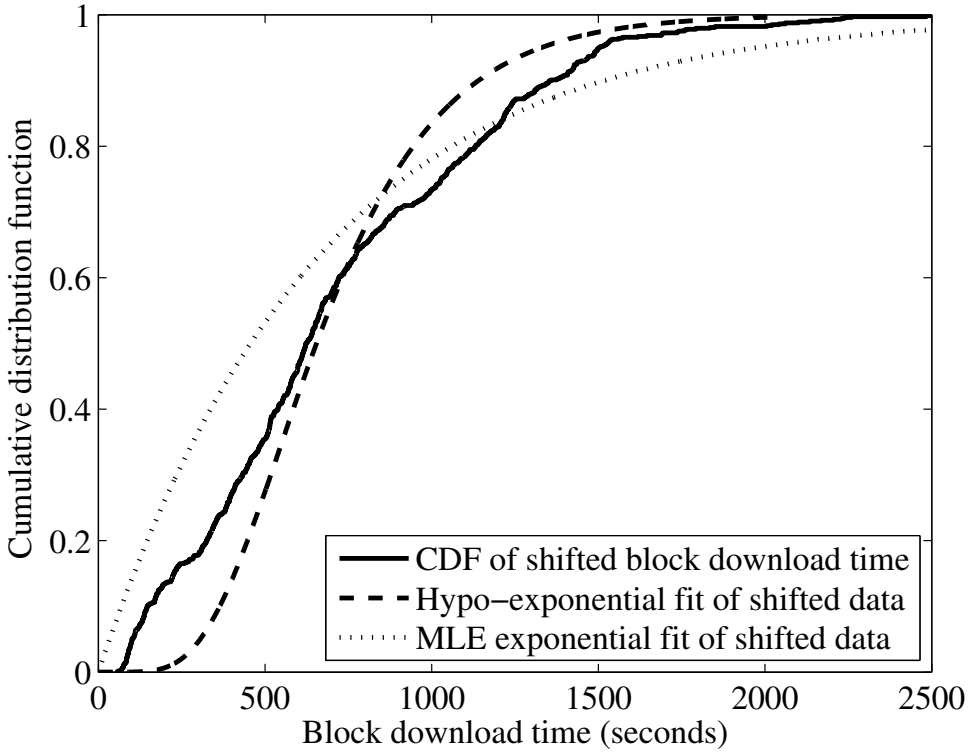


(d) Fitting of the block download time distribution

Figure 4.5: Experiment 8 (top a+b) and 9 (down c+d): Fragment and block download times.



(a) Exponential fit of the fragment download time distribution



(b) Fitting of the block download time distribution

Figure 4.6: Experiment 10: Fragment and block download times.

5

FLOW-LEVEL MODELING OF PARALLEL DOWNLOAD PROCESS: FIRST STEP TOWARD A SCALABLE P2P SIMULATOR

5.1 Introduction and related work

[1] 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 or peers 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, in contrast to the traditional client-server model where only servers supply, and computers consume. Applications that use this distributed network provides enhanced scalability and service robustness as all the connected computers or peers provide some services.

This distributed network model has proved to be an alternative to the Client/Server model and a cheap, scalable, self-repairing and promising paradigm for grid computing, grid delivery network (GDN), file sharing, voice over IP (VoIP), backup and storage applications.

Such distributed systems rely on data fragmentation and distributed storage. Files are partitioned into fixed-size blocks that are themselves partitioned into fragments. Fragments are

[1] joint work with Alain Jean-Marie (Research director at INRIA and LIRMM, CNRS/Université Montpellier 2)

usually stored on different peers. Given this configuration, a user wishing to retrieve a given block of data would need to perform multiple downloads, generally in parallel for an enhanced service. The transfer of sequences of packets on one long-term TCP connection (e.g. download a fragment of data between two peers in a P2P system or between a client and a server through FTP protocol) defines a "flow". A flow can as well refer to the sequences of packets that constitute a block of data and that follow several TCP connections simultaneously. In this work, we will consider the former definition.

One measure of the quality of the service given by the distributed storage/parallel download infrastructure is the time it takes to retrieve the complete document. This in turn depends on the throughput of the different flows created to obtain the fragments of this document. Their values are, *a priori*, a function of the demand and capacities of the complete network entities: clients, servers and links.

The basic problem of predicting the instantaneous shares of the bandwidth received by each flow of a TCP-based network has received quite some attention in the last 15 years, in connection with the notion of *fairness*; yet, *there is no clear consensus in the literature on a simple formula or algorithm* to give a reasonable solution of this problem. Such an algorithm would be extremely useful to build flow-level simulators and, possibly, to perform probabilistic performance calculations.

On the one hand, some authors have shown that the dynamics of TCP have been shown to be quite chaotic is some situations. Other authors on the other hand, have argued that TCP tends to share the bandwidth between flows reasonably. For instance, Heyman *et al.* [52], followed by Fredj *et al.* [43], have studied a single bottleneck link shared by a given number of identical sources that alternately send documents through the shared link and stop sending for a randomly thinking time. They showed through simulations that TCP shares *fairly* the bottleneck (that is, in equal shares) and they introduced analytical tools that can predict the expectation of the transmitting rate. Varki proposed in [94] a simple approximation for the expected response time based on the fork-join model. Massoulié and Roberts proposed in [63] a model similar to that of [52] where the inter-flows arrival times are iid exponentially distributed random variables. They studied the network as M/G/1 PS queue. In [22], Chiu and Eun, the authors have focused on the average download time of each user in a P2P network while considering the heterogeneity of service capacities of peers. They point out that the common approach of analyzing the average download time based on average service capacity is fundamentally flawed.

Other studies have put forward the concepts of max-min fairness, proportional fairness, balanced fairness and utility-based resource-sharing models (see *e.g.* [16] and the reference therein). One conclusion of these studies is that throughput allocations resulting from the use of the TCP protocol for infinitely long flows are usually *not* max-min fair. However, the results of

Bonald and Proutière [15] suggested that when the flows are dynamic (flows are continuously created and have a finite duration), the average throughput obtained by flows under various sharing mechanism tend to be similar. It is quite possible that, from a practical perspective, the predictions obtained with a max-min fair sharing mechanism may be "good enough".

The purpose of this chapter is to assess whether max-min fairness for the allocation throughput is a proper model when evaluating response times of parallel downloads.

Given the variety of situations to be studied, we begin with the simplest scenario: a symmetric network in which we assume that capacity constraints are located at the client/server nodes, and not inside the network. We also assume that all RTTs are equal. [2]

We use an algorithm which calculates an instantaneous throughput for each individual flows in a certain set of flows, given the upload and download capacities of the client and server nodes. This algorithm can be seen as a variant of the "progressive filling" [16] or "water filling" algorithm of [7]: we name it as the Progressive Filling Flow Level Algorithm (PFFLA). The validation of this algorithm consists in characterizing the response time of parallel downloads in a distributed storage system, through simulations. We have implemented the PFFLA in a flow-level simulator of parallel downloads, and we have programmed a similar model over NS2. The response times in the flow level simulator have been compared to that of the packet-level simulations in NS (both distributions and averages). This experimental setting is, to the best of our knowledge, original in at least three features. First, we consider finite flows related to downloads in parallel, which are synchronized when they are created. In addition, the performance metric is the globale response time, not that of individual flows. Second, we consider that the possible bottlenecks for flows occur only at the edge of the network, never inside. Finally, we consider large numbers of nodes (up to 500) and flows (up to $4{\cdot}10^5$).

Our results show that the relative error between PFFLA and NS-2 for the expected value is less than 2% for relatively large loads in the system (e.g. $\rho = 70\%$) and less than 1% for low loads in the symmetric up/down case and less than 5% respectively for relatively large loads in the system (e.g. $\rho = 50\%$) and less than 1% for low loads in the asymmetric case. We conclude that PFFLA is a reliable mechanism to analyze the service response time in many systems based on P2P and Grid computing concepts such as Storage Systems and Grid Delivery Networks.

The rest of this paper is organized as follows. Section 5.2 overviews the system assumptions and notation. Section 5.3 describes the flow-level simulation algorithm "PFFLA". In Section 5.4, comparisons between packet-level and PFFLA are introduced and discussed. Last, Section 5.5 concludes the paper and highlights some future directions.

[2] The question of how to handle different round trip times is left for future work.

5.2 System description and notation

In the following, we will distinguish the *servers*, which are computers that provide a storage service, from the *clients* whose objective is to retrieve data from the servers to account for the fact that flows (transfer of sequences of data units from a server to a client) have a direction. In the terminology of P2P-based systems, each "peer" has the role of both a client and a server. It is usual that the communication link from the network to the peer (upload link) and the one from the peer to the network (download link) are not shared. Their capacity may actually not even be the same, as with ADSL network accesses. In that case, the entities client and server can be considered as two distinct nodes. On the other hand, if the network access is indeed shared between input and output, the peer is represented by one node. In the following, we shall only consider the first situation.

In this study, we are interested in systems where blocks of data are partitioned into several equally sized fragments stored randomly over different servers. We will consider both homogeneous (upload/download capacities are identical) and asymmetric situations.

We consider a distributed system in which the following assumptions and notations will be enforced throughout the paper.

Network assumptions:

- The considered network consists in a set of nodes $\mathcal{N}$. In a P2P context, there will be $\mathcal{N}/2$ peers according to this notation. In other words, we will have $\mathcal{N}/2$ servers and $\mathcal{N}/2$ clients.
- The logical structure of the network is that of a star, with an infinite-capacity central node. In other word, the interconnection network underlying the parallel download application is assumed not to introduce capacity constraints. Only the upload or download links (the branches of the star) have a limited capacity.

  The capacity of upload links (from servers to the network) is C_u, the capacity of download links (from network to clients) is C_d.
- The temporal distance (measured as the round-trip time, RTT) is assumed to be the same between pairs of nodes (clients or server).

Data and traffic assumptions:

- Each block of data D of size S_B is partitioned into s fragments of size S_F.
- We assume that the s servers that hold fragments of a given block of data D are uniformly selected over all servers in the system, and are all distinct.

- Each download request of a block of data issued by a client will generate s parallel requests toward s servers to retrieve s distinct fragments of the requested block. A request generates s flows.

The assumption on the uniform distribution of the blocks of some document corresponds to the situation where a very large number of documents exist, and/or each fragment of each document has been replicated a large number of times. In that situation, it is unlikely that the set of blocks needed by two distinct requests will be correlated. The network being symmetric, it is reasonable to assume that fragments have been uniformy distributed. The assumption that different fragments of some document are stored on different peers is common in P2P-based systems: it results mainly from privacy and data ownership issues.

5.3 Description of the algorithm

Before describing the algorithm we have used, we recall the principle of max-min fairness, and the algorithm (referred to as the "Progressive Filling" algorithm in [16]).

The notion of max-min (or maximin) fairness originates from the field of political philosophy and economics, and was introduced in the context of networking by Bertsekas and Gallager [7, ch. 6] as a design objective for communication networks, in particular, the design of flow control schemes. The main idea of max-min fairness is to maximize the allocation of each flow f subject to the constraint that an incremental increase in f's allocation does not cause a decrease in some other flow's allocation that is already as small as f's or smaller [7, p. 526].

For the purpose of formalizing the description of the PFFLA algorithm, introduce the following notation. A node (server or client) will be represented by the link that connects it to the network core. The network is assumed to be made of a set $\mathcal{A}$ of links. Each link a has a capacity C_a. The traffic is formed by a set $\mathcal{F}$ of flows. We assume that flows cannot be split between several routes of the network. This implies that we can assume that each flow f has a throughput $\theta_f \geq 0$, and crosses certain links of $\mathcal{A}$. We write $f \nabla a$ to denote the fact that f crosses a.

Using this notation, the total flow on link a of the network is then given by:

$$F_a = \sum_{f \nabla a} \theta_f .$$

The capacity constraint for the network is then:

$$F_a \leq C_a, \quad \forall a \in \mathcal{A} . \tag{5.1}$$

A vector of throughputs $\{\theta_f\}$ satisfying these constraints is said to be *feasible*.

The existence of a max-min fair allocation of throughputs is not entirely obvious. Indeed, satisfying the capacity constraints implies that the increase of some flow's throughput may result in the decrease of another flow's throughput, and conversely.

Bertsekas and Gallager have proved that there exists one unique feasible allocation of throughputs which is max-min fair. It can be constructed using the following algorithm.

The idea is to start with an all-zero rate vector and to increase the rates on all paths together until $F_a = C_a$ for one or more links a. At this point, each flow using a saturated link has the same throughput at every other flow using that link. Thus, these saturated links serve as bottleneck links for all flows using them.

At the next step, all flows not using the saturated links are incremented equally in rate until one or more new links become saturated. The newly saturated links serve as bottleneck links for those flows that pass through them but do not use the previously saturated links. The algorithm continues until all flows pass through at least one saturated link. This process is often visualized as progressively augmenting the throughput until capacities are "filled", hence the name of "progressive filling". Information flows are also sometimes visualized as some "fluid" which is poured into the network. For this reason, the algorithm is also referred to as a "water filling" algorithm.

We have used in our analysis the following version of this algorithm.

Data: A set of links $\mathcal{A}$ with their capacities C_a, and a set of flows $\mathcal{F}$
Result: A throughput value for each flow, satisfying the throughput constraint (5.1)
begin
  Remove from $\mathcal{A}$ nodes without flows ;
  while $\mathcal{A}$ *not empty* **do**
    foreach $a \in \mathcal{A}$ **do**
      $N_a \leftarrow \#\{f \in \mathcal{F} | f \nabla a\}$;
    end
    calculate $\theta^* = \min_{a\in\mathcal{A}} C_a/N_a$;
    calculate $a^* = arg\min_{a\in\mathcal{A}} C_a/N_a$;
    foreach f, $f\nabla a^*$ **do**
      set $\theta_f = \theta^*$;
      foreach a, $f\nabla a$ **do**
        $C_a \leftarrow C_a - \theta^*$
      end
      remove f from $\mathcal{F}$;
    end
    remove from $\mathcal{A}$ links without flows ;
  end
  return $\{\theta_f\}$
end

Algorithm 1: Algorithm PFFLA

The fact that this algorithm produces a max-min allocation can be checked the same way as for the Progressive Filling Algorithm: according to [7, p. 527], max-min solutions are characterized by the "bottleneck" property:

$$\forall f \in \mathcal{F}, \quad \exists a \in \mathcal{A}, \quad f\nabla a \quad \text{and} \quad \sum_{g\in\mathcal{F}, g\nabla a} \theta_g = C_a \quad \text{and} \quad \forall h \in \mathcal{F}, \quad h\nabla a, \theta_h \geq \theta_f \,. \quad (5.2)$$

This property can be checked almost by construction on our algorithm.

More remarks concerning this algorithm:

- The algorithm eventually stops because at least one link is removed from $\mathcal{A}$ at each loop. The number of loops is therefore bounded by the cardinal of the initial set of links. Since several links can be removed in each loop, the algorithm may actually stop faster.
- When $arg\min_{a\in\mathcal{A}} C_a/N_a$ contains more than one element, it does not matter which one is chosen, in the sense that the outcome of the algorithm does not depend on that particular choice.

This results, by contradiction, from the fact that the max-min allocation is unique. It can also be proved that other links of the set are still in the "argmin" at the next step, so that each of them will be chosen eventually. Equivalently, one may remove simultaneously from the network all flows that are connected to links in the "argmin" set.

- It is possible to add constraints on the throughput of flows. For instance, the throughput of a TCP flow on a lossless connexion with RTT τ and maximum window size w is always less than w/τ.

In our situation, we have made the assumption that the network can be represented by a star, and the flows cross exactly two links: one upload link and one download link. The algorithm is capable to handle general situations however.

5.4 Experimental results

5.4.1 Parameter values

We ran a total of seventeen experiments; ten in symmetric peers download/upload capacities scenarios (homogeneous networks) and seven in asymmetric scenarios.

The set-up of the simulation parameters is summarized in Table 5.1. The capacities that we have selected in the simulations vary between the values of the ISDN and ADSL technologies (384, 576 and 1500 kbps). In experiments 1–10, nodes are homogeneous: they have all the same network access capacity. In Experiments 11–17, capacities of clients and servers are asymmetric.

Download requests at each client node arrive according to some Poisson process of given rate λ. The different request processes are independent. This assumption is reasonable in practice: Guha *et al.* have shown in [47] that in real networks, and when the number of clients N_c is large, the request arrival process can be reasonably modeled by a Poisson process. We vary the value of the request generation rate across the experiments such that the total load in the system ρ (see below) varies from low (e.g. 6%) up to high value (e.g. 70%) as reported in Table 5.1.

The last setting concerns the blocks and fragments sizes that are stored in the system. Fragment sizes S_F (resp. block sizes S_B) in P2P systems, for instance, are typically between 256KB and 4MB each (resp. between 4MB and 9MB each). We will consider in most of our experiments $S_F = 2$MB and $S_B = 8$MB, except in Experiment 1 where $S_F = 1$MB and $S_B = 4$MB. Therefore $s = 4$ in all experiments. In the asymmetric scenarios, we have chosen the two capacities values 1500/384 kbps, except in Experiment 17 where the capacities values are 2000/384 kbps. So, in all the asymmetric experiments, except in the last one, the capacity of a server is slightly larger than $1/s$ times the capacity of a client.

For the packet-level simulation details, we consider a fixed constant value of 2ms for the link propagation delays. The main TCP configurations are as follow: we use TCP segment size (S_{pkt}) of 1460 Bytes, the upper bound on the advertised window for the TCP connection is set to 40, the initial size of the congestion window on slow-start is 2, and the TCP/IP header size (h_{ip}) is 40 Bytes. The P2P application layer header (h_a), which is implemented over the NS transport layer, is 13 Bytes for each fragment. The queue management type used in the links is "DropTail" with size of 500 packets. The maximum window size is left to NS2's default of 64kB. Given our assumptions on propagation delay, this gives a maximum TCP throughput of $64\text{kB}/8\text{ms} = 4\text{MB/s}$, largely superior to the capacity of the links. Therefore, maximum window effects are not expected to restrict the throughput of file transfers.

In the flow-level simulation, and when calculating the total amount of data sent in the TCP flows, we neglect the fact that one data packet may be incomplete after segmentation. We also neglect the packets sent during the opening and the closing of the TCP connection, and we assume that no retransmission occurs. The total amount of data transported during the download of one document is then calculated by multiplying the application data size by the overhead factor due to packet headers, that is:

$$L(\text{bits}) \;=\; s \;\times\; (S_F(\text{bits}) + h_a(\text{bits})) \;\times\; (1 + h_{ip}/S_{pkt})\;. \tag{5.3}$$

Consider a client node with link capacity C. The time to download a complete document would be, when no interferences from other downloads occur:

$$\sigma \;=\; s \;\times\; \frac{(S_F(\text{bits}) + h_a(\text{bits})) \;\times\; (1 + h_{ip}/S_{pkt})}{C(\text{bps})}\;. \tag{5.4}$$

On the other hand, if the global arrival rate of document requests is λ, the rate of requests arriving at a particular client is $\lambda/(\mathcal{N}/2)$. Accordingly, the load factor of a client link of capacity C in the network is:

$$\rho \;=\; \lambda s \;\times\; \frac{(S_F(\text{bits}) + h_a(\text{bits})) \;\times\; (1 + h_{ip}/S_{pkt})}{C(\text{bps})\mathcal{N}/2}\;. \tag{5.5}$$

Consider now a server node with link capacity C. Given our assumption on the uniform repartition of blocks on servers, the rate of arrivals of fragment requests at the servers is $\lambda s/(\mathcal{N}/2)$. The duration of one request should be σ/s since only one fragment is concerned. Finally, the load factor of the server's link is given by Equation (5.5) also.

In the homogeneous cases, this value of ρ can also be interpreted as the load of the whole network (ratio of global data requests to global transfer capacity). In the asymmetric cases, we take ρ as the load of the links with the smallest capacity.

5.4.2 Simulators and Metrics

We have developed a packet-level simulator and a flow-level simulator for our model. The packet-level simulator is build using NS2. Its implementation details can be found in [31].

The flow-level simulator consists in the embedding of Algorithm 1 into a discrete-event simulator handling the arrival and the departure of flows. The principle is that every time the set of flows present in the network changes, the bandwidth shares are re-computed with the algorithm, and it is assumed that these throughputs are obtained instantaneously. The program keeps track of the remaining quantities to be downloaded in each flow, and can compute the date of the next event: arrival or end of download.

Both simulators are instrumented so as to produce response times for fragments and complete documents.

The metric we are interested in is the *download time* of a document. For a given request, this is the maximum between the download time of the s fragments of the document. Of course, this is a random variable, and we measure its empirical distribution and empirical average. The empirical average obtained with the packet-level and flow level simulators are denoted with $E[T_{NS}]$ and $E[T_{FLA}]$, respectively. In the view of the fact that the flow-level simulator ignores the delay for establishing and closing the TCP connections and propagation delays, there will be, for any experiment, a very small difference between the minimum values obtained from both simulators. Therfore, we denote by $\hat{E}[T_{NS}]$ the measured download time for NS, corrected by a constant value so that the minimal values for both simulators are the same. This last metric will be used later to compare the average response times in both simulators. However, we have not corrected this systematic error in the figures, presented later in this paper, for illustrative purpose.

In addition, we have compared the average document download times with the average response time in a simple queueing system. The rationale for this is that, if the throughput of the connections were limited only by the client's capacity, then the link would behave as a Processor Sharing queue. This is because the size of the fragments is the same, so that the response time of all s fragments is the same, and all s fragments can be actually considered as a single "customer". The client's bandwidth is then shared between different requests. Since requests arrive according to a Poisson process, the model is that of a $M/D/1$ processor sharing (PS) queue. This model is expected to work well when the load is small: indeed, in that case it is unlikely that flows will be limited on the server side. Since the average response time in this queue, denoted by $E[T_{PS}]$, can be computed with a simple formula, we can easily test this conjecture.

The average response time in the $M/D/1/PS$ queue is known to be, in seconds, as follow

Table 5.1: Experiments setup

Experiment number	$\mathcal{N}/2$ peers	C_d/C_u kbps	S_B/S_F MB	$1/\lambda$ sec.	ρ %	samples	Required time hours
1	25	384/384	4/1	60	6	45000	20
2	25	576/576	8/2	39.88	12	77500	18
3	250	1500/1500	8/2	1.536	12	25000	15
4	250	1500/1500	8/2	1.024	18	25000	13
5	250	576/576	8/2	1.913	25	30000	33
6	250	1500/1500	8/2	0.734	25	37000	31
7	250	1500/1500	8/2	0.510	36	40000	30
8	250	1500/1500	8/2	0.367	50	40000	36
9	250	1500/1500	8/2	0.306	60	67500	58
10	250	1500/1500	8/2	0.262	70	280000	264
11	25	1500/384	8/2	59.81	12	30000	23
12	250	1500/384	8/2	5.98	12	30000	54
13	500	1500/384	8/2	2.99	12	30000	25
14	250	1500/384	8/2	1.99	36	55000	54
15	500	1500/384	8/2	0.996	36	25000	11
16	500	1500/384	8/2	0.718	50	40000	18
17	500	2000/384	8/2	0.718	50	40000	19

(see *e.g.* [57]):

$$E[T_{PS}] \;=\; \frac{\sigma}{1-\rho} \;=\; \frac{s(\cdot S_F(\text{bits}) + h_a(\text{bits})) \times (1 + h_{ip}/S_{pkt})}{C(\text{bps}) \times (1-\rho)} \quad (\text{s})\,. \tag{5.6}$$

We will compute the relative error (RR) between $\hat{E}[T_{NS}]$ and $E[T_{PS}]$, on one hand, and between $\hat{E}[T_{NS}]$ and $E[T_{FLA}]$ on the other hand. The relative error, for instance between results from NS-2 and FLA is calculated as follow:

$$RR(NS, FLA) \;=\; \frac{\hat{E}[T_{NS}] - E[T_{FLA}]}{\hat{E}[T_{NS}]}$$

Known results on the PS queueing model also include the distribution of the response time. The relevant formulas are provided in the Appendix B.

5.4.3 Results

We have run both the flow-level simulator and the packet-level simulator on the seventeen sets of values described in Table 5.1.

For the flow-level simulations, we have collected 100000 samples of the document download time in every case. The number of samples collected with the packet-level simulation is reported in Table 5.1.

We also report in Table 5.1 the execution time of the packet-level simulations, it varies between some hours and some days. The experiments were run on a machine with the following principal characteristics: multithreaded processor Intel(R) Core(TM)2 Duo of 2.66GHz, 4GB RAM + 4GB swap running Fedora Core 5. The analysis of number of samples issued by unit time of computation (figure not reported) reveals that this number decreases with the number of nodes. Interestingly, it tends to slowly decrease when the load ρ of the network increases, but it is actually increasing for high values of ρ. We do not have an explanation for this observation.

The execution times for the flow-level simulation are not reported in details. It varies between one second and several minutes on a machine with characteristics: processor Intel(R) Core(TM)2 Duo of 2.00GHz, 2GB RAM running Fedora Core 5 (slightly less power than the machine used for packet-level simulations). The simulation time per sample increases with the load of the system, due to the fact that the load sharing must be re-computed every time a flow starts or stops. For a given load, it also increases with the number of nodes.

We conclude that the flow-level algorithm is very efficient in terms of time. However, one question remains: how good is it in term of accuracy?

To answer this question, we first depict in Figures 5.1–5.10 the empirical complementary cumulative distribution function (CCDF) of the block download time obtained form both simulators, and for all the experiments. We report then in Table 5.2 the expected download time obtained from both simulation levels and from the PS formula (5.6). Table 5.2 reports as well the relative error between results.

The results show that for small system load, the download time predicted by the PFFLA fits exactly that of the NS-2. The relative error between the average values is very small as shown in Table 5.2. The average value calculated from the PS formula is also very close but the relative error between average values of PS and NS-2 is slightly larger than that between PFFLA and NS-2. This confirms that the prediction of the duration of TCP flow is accurate. Indeed, since these are long flows, the slow start phase can be easily neglected. Other phenomena which typically perturb the throughput of TCP (packet losses, buffer fluctuations) probably happen very rarely in this case. The flow sharing algorithm apparently provides a very good approximation, for average response times as well as for distributions.

When ρ is relatively large, some buffers can fill up more frequently, and then some flows tend to be relatively long in the NS-2 simulation. However, the relative errors between average values of PFFLA and NS-2 are slightly more important in this case but still very small, in particular, RR is less than 2% in the symmetric case and less than 5% in the asymmetric case. It is clear from Figures 5.5– 5.6 that the distributions measured by both simulators are different in

Table 5.2: Measurements for the PFFLA and the packet-level simulation; comparison with the PS model

Experiment number	$\mathcal{N}/2$ peers	ρ %	$\hat{E}[T_{NS}]$ sec.	$E[T_{FLA}]$ sec.	RR% (NS, FLA)	$E[T_{PS}]$ sec.	RR% (NS, PS)
1 symm.	25	6	96.062	95.45	0.6%	95.44	0.6%
2 symm.	25	12	135.04	136.071	-0.7%	141.59	-4.8%
3 symm.	250	12	51.77	52.089	-0.6%	52.195	-0.8%
4 symm.	250	18	55.57	55.96	-0.7%	56.015	-0.8%
5 symm.	250	25	161.252	160.196	0.6%	166.132	-3%
6 symm.	250	25	61.068	61.517	-0.7%	61.243	-0.3%
7 symm.	250	36	73.547	73.346	0.2%	71.7692	2.4%
8 symm.	250	50	99.501	97.75	1.7%	91.864	7.6%
9 symm.	250	60	129.066	127.691	1%	114.83	11%
10 symm.	250	70	176.45	180.05	-2%	153.107	13.2%
11 asymm.	25	12	61.137	62.901	-2.8%	52.19	17%
12 asymm.	250	12	64.738	64.935	-0.3%	52.19	19.3%
13 asymm.	500	12	65.298	65.182	-0.18%	52.19	20%
14 asymm.	250	36	103.70	110.231	-6.2%	71.76	30.8%
15 asymm.	500	36	107.18	110.396	-3%	71.76	33%
16 asymm.	500	50	144.615	149.213	-3.1%	91.865	36%
17 asymm.	500	50	142.1	149.213	-5%	68.45	51.8%

the symmetric case for very high load, but average values turn out to be almost identical. The same observation holds for asymmetric cases, see Figures 5.8(b) to 5.10.

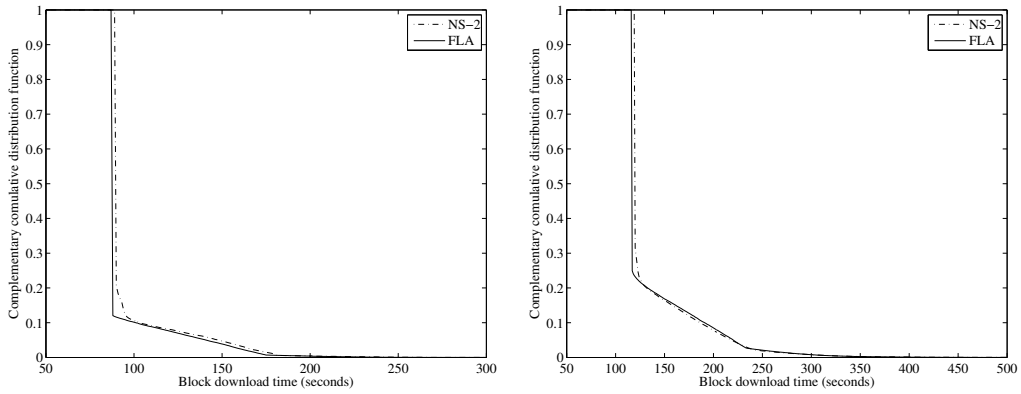
The accuracy of the PS approximation for the average download time is acceptable for symmetric cases up to $\rho = 36\%$, and degrades above $\rho = 50\%$. The accuracy for the complete distribution can be assessed on Figure 5.4(b) for a load of $\rho = 36\%$. In the asymmetric cases, the approximation is bad at low loads, and very bad at large loads. The explanation for this is the following. The download of a block at a client can be slowed down by two phenomena. The first one is that a second request arrives at the node. This is taken into account by the PS model. The second one is that one TCP flow is slowed down at the server side. This requires that at least sC_u/C_d blocks are downloaded simultaneously from the server. In the symmetric cases, this value is 4 and the event rarely happens, even for moderate loads. In the asymmetric case, this value is 1 and this is much more frequent. See the Appendix B for more comments on the PS approximation. In order to understand better the differences between flow-level and packet-level distributions, a careful analysis of the congestion avoidance mechanism in

congested networks and an extension of the algorithm to account for the overloaded system (ρ around one) are the objective of ongoing research.

Another observation is that larger the network size, better the performance of PFFLA and worse the performance of PS model as illustrated by Experiments 11, 12 and 13 (resp. 14 and 15). The number of peers in these experiments are 25, 250 and 500 respectively (resp. 250 and 500) for same load and capacities. However, the relative error occurred in Experiment 14 is less than that of 13 which is, in turn, less than that of Experiment 12. Respectively, the relative error occurred in Experiment 16 is less than that of 15. Indeed, the performance does not depend only on the system load but also on the number of peers and their capacities.

Clearly, larger the buffer sizes, better the performance in real networks. To address this point, we depict in Figure 5.11 the CCDF of block download time for two values of the queue limit (100 and 500 packets) in NS-2 simulation for 25 peers (50 nodes), $C = 1500\text{kbps}$ and $\rho = 70\%$. The relative error between the two expected download time is 6.56% (159.575 seconds for 500 packets and 170.038 for 100 packets) and then the buffer size can affect the performance of the system with high load.

We conclude that PFFLA is a reliable mechanism to analyze the service response time in many non-overloaded systems based on P2P and Grid computing concepts such as Storage Systems and Grid Delivery Networks. In particular, when the size of networks is relatively large, PFFLA predictions are very accurate as long as the system is not overloaded or close to be overloaded.



(a) $\rho = 6\%$, C=384kbps, $\mathcal{N}$=50, S_B=4MB, S_F=1MB (b) $\rho = 12\%$, C=576kbps, $\mathcal{N}$=50, S_B=8MB, S_F=2MB

Figure 5.1: Experiments 1 (left) and 2 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2

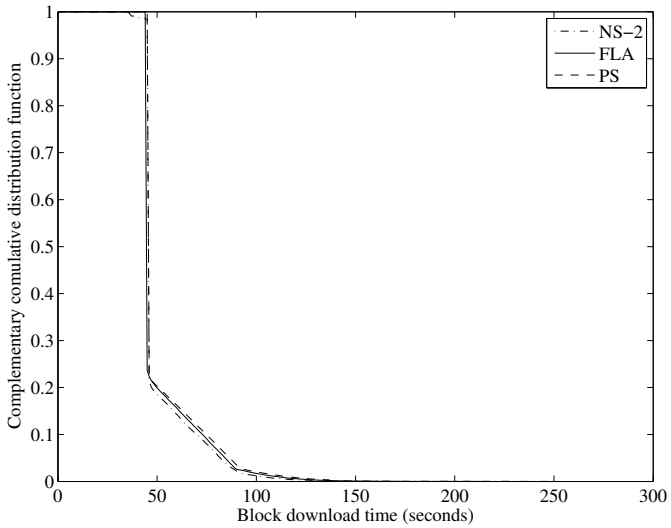


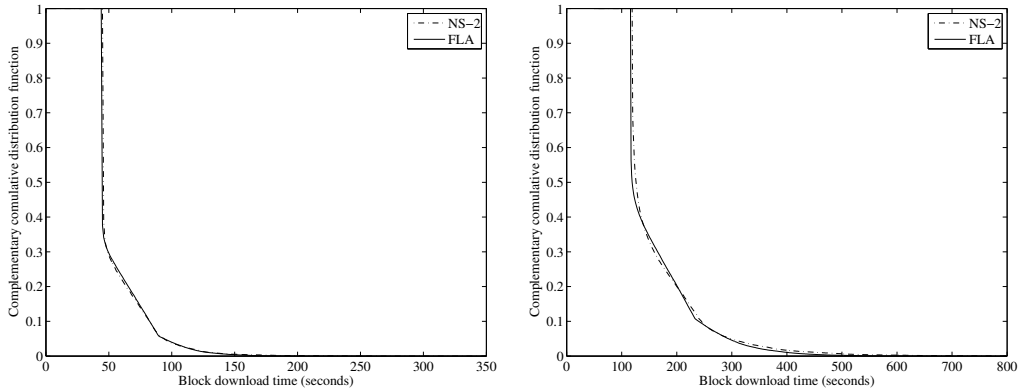
Figure 5.2: Experiment 3: Packet-level simulation NS-2 vs progressive-filling flow-level algorithm FLA & PS for $\rho = 12\%$, C = 1500kbps, $\mathcal{N} = 500$, $S_B = 8$MB.

5.5 Conclusion and future work

In this report, we have proposed and analyzed the PFFLA algorithm. The algorithm is quite simple and uses the concept of "Progressive-Filling" (or max-min fairness). We have implemented the PFFLA in a flow-level simulator of parallel downloads, and we have programmed a similar model over NS2. The response times in the flow level simulator have been compared to that of the packet-level simulations in NS (both distributions and averages).

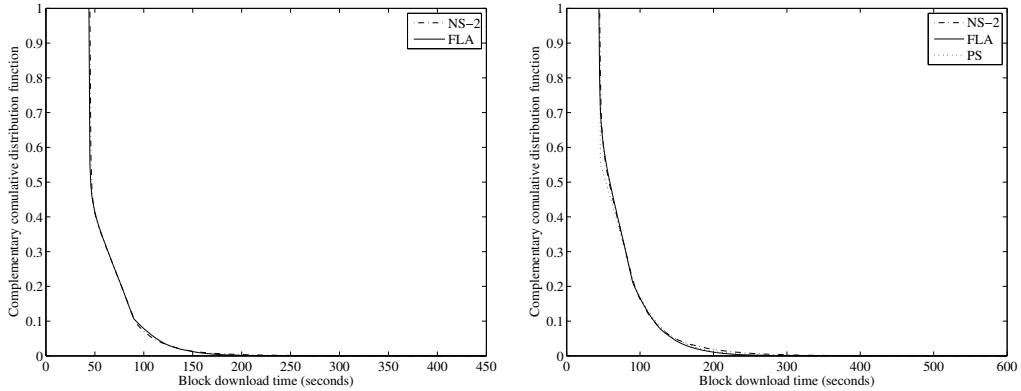
Our results conclude that PFFLA is a reliable mechanism to analyze the service response time in many systems based on P2P and Grid computing concepts such as Storage Systems and Grid Delivery Networks.

A conclusion from the literature is that different RTTs do introduce some "unfairness" in bandwidth allocations. Our next step will therefore be to find a simple yet efficient modification of Algorithm 1 to handle this situation.



(a) $\rho = 18\%$, C=1500kbps, $\mathcal{N}$=500, S_B=8MB. (b) $\rho = 25\%$, C=576kbps, $\mathcal{N}$=500, S_B=8MB.

Figure 5.3: Experiments 4 (left) and 5 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.



(a) $\rho = 25\%$, C=1500kbps, $\mathcal{N}$=500, S_B=8MB. (b) $\rho = 36\%$, C=1500kbps, $\mathcal{N}$=500, S_B=8MB.

Figure 5.4: Experiments 6 (left) and 7 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.

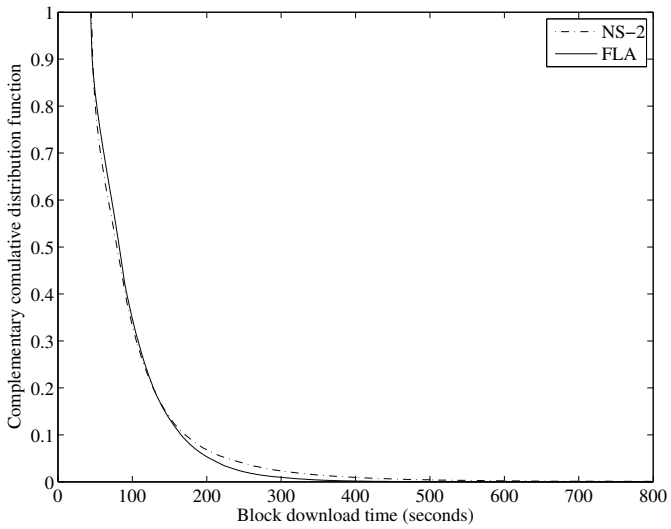
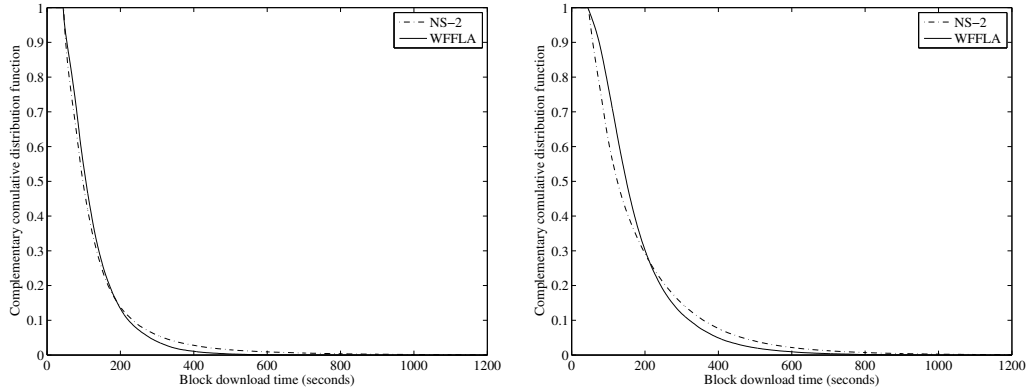
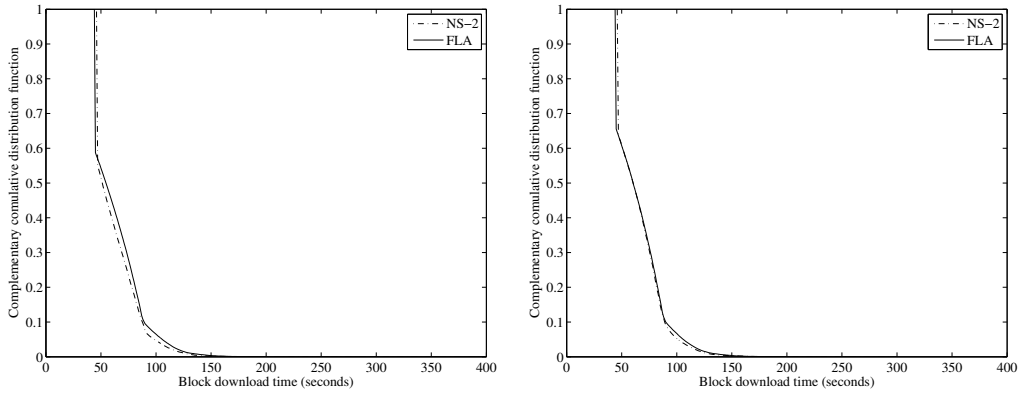


Figure 5.5: Experiment 8: progressive-filling flow-level algorithm FLA vs Packet-level simulation NS-2 for $\rho = 50\%$, $C = 1500$kbps, $\mathcal{N} = 500$, $S_B = 8$MB.



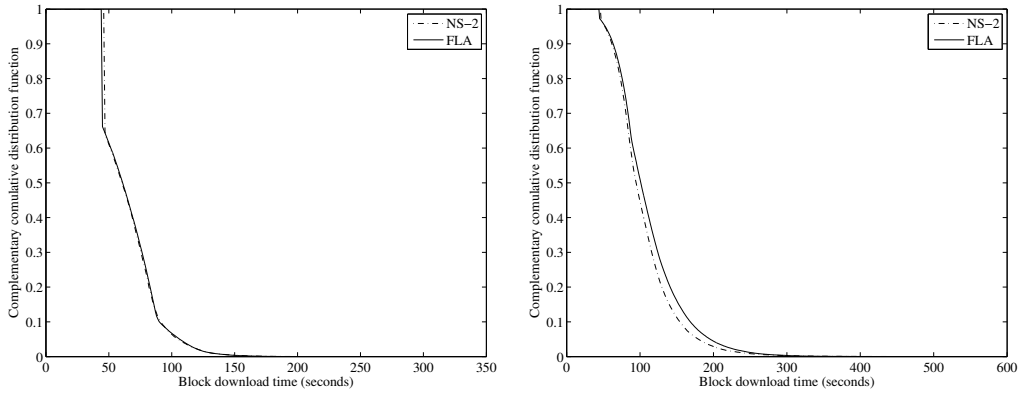
(a) $\rho = 60\%$, C=1500kbps, $\mathcal{N}$=500, S_B=8MB. (b) $\rho = 70\%$, C=1500kbps, $\mathcal{N}$=500, S_B=8MB.

Figure 5.6: Experiments 9 (left) and 10 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.



(a) $\rho = 12\%$, C_d=1500kbps, C_u=384kbps, $\mathcal{N}$=50, S_B=8MB. (b) $\rho = 12\%$, C_d = 1500kbps, C_u = 384kbps, $\mathcal{N}$ = 500, S_B = 8MB.

Figure 5.7: Experiments 11 (left) and 12 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.



(a) $\rho = 12\%$, C_d=1500kbps, C_u=384kbps, $\mathcal{N}$=1000, S_B=8MB. (b) $\rho = 36\%$, C_d=1500kbps, C_u=384kbps, $\mathcal{N}$=500, S_B=8MB.

Figure 5.8: Experiments 13 (left) and 14 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.

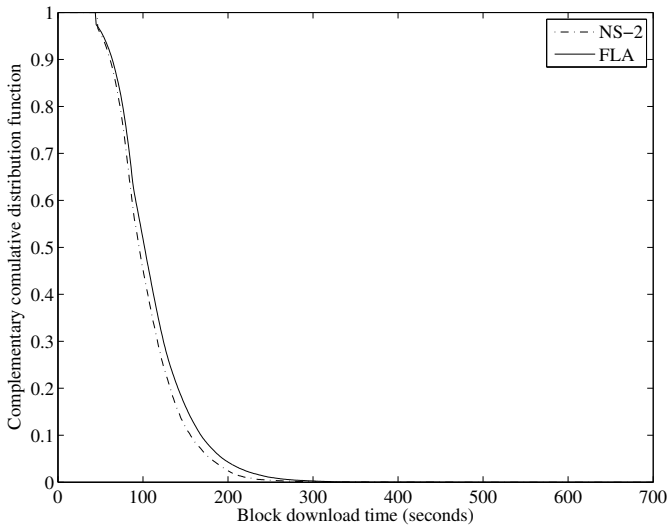
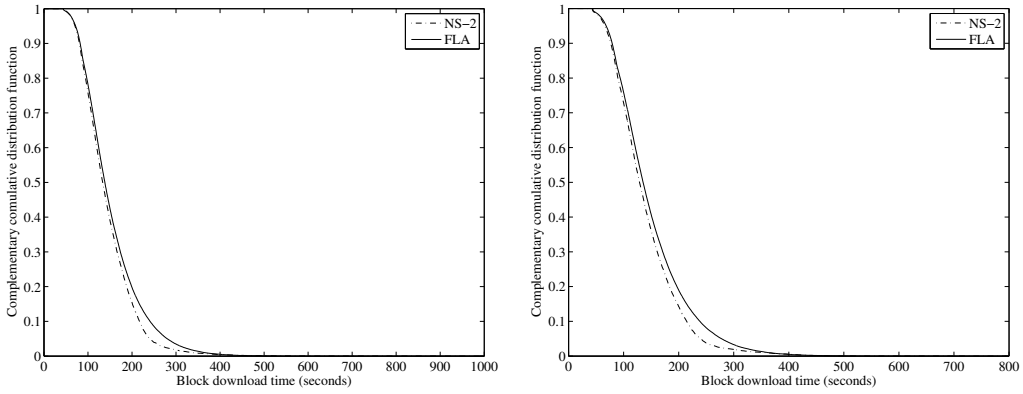


Figure 5.9: Experiment 15: progressive-filling flow-level algorithm FLA vs Packet-level simulation NS-2 for $\rho = 36\%$, $C_d = 1500$kbps, $C_u = 384$kbps, $\mathcal{N} = 1000$, $S_B = 8$MB.



(a) $\rho = 50\%$, C_d=1500kbps, C_u=384kbps, $\mathcal{N}$=1000, S_B=8MB.

(b) $\rho = 50\%$, C_d=2000kbps, C_u=384kbps, $\mathcal{N}$=1000, S_B=8MB.

Figure 5.10: Experiments 16 (left) and 17 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.

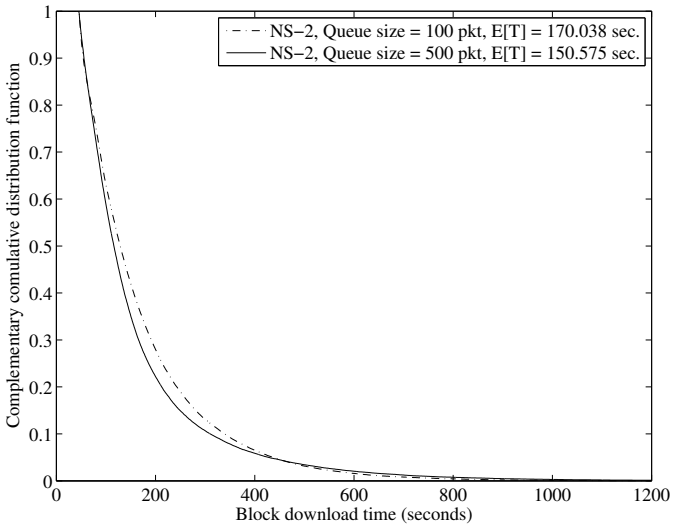


Figure 5.11: Queue size effect in Packet-level simulation NS-2 for $\rho = 70\%$, $C = 1500$kbps, $\mathcal{N} = 50$, $S_B = 8$MB.

6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

The growth of storage volume, bandwidth, and computational resources for PCs has allowed to build a new network paradigm (called overlay) on top of 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, in contrast to the traditional client-server model where only servers supply, and computers consume. Applications that use this distributed network provides enhanced scalability and service robustness as all the connected computers or peers provide some services.

This distributed network model has proved to be an alternative to the Client/Server model and a cheap, scalable, self-repairing and promising paradigm for grid computing, grid delivery network (GDN), file sharing, voice over IP (VoIP), backup and storage applications.

We have addressed in this thesis the analysis and the optimization of the performance of peer-to-peer backup and storage systems in terms of the delivered data lifetime and data availability. In such systems, data are no longer stored on expensive magnetic tapes but on much cheaper hard disks. This systems rely on data fragmentation and distributed storage. Files are partitioned into fixed-size blocks that are themselves partitioned into fragments. Since in a P2P network, peers are free to leave and join the system at any time, ensuring high availability of the stored data is an interesting and challenging problem. To ensure data reliability, redundant data is inserted in the system. Redundancy is achieved, in practice, either by replication or by using erasure codes. For the same amount of redundancy, erasure codes provide higher availability of data than replication. A new class of codes has been proposed recently in [35], the

so-called Regenerating Codes. This new redundancy scheme can be considered a generalization of the erasure coding, that reduces the communication cost of erasure coding by slightly increasing the storage cost.

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 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 were 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 were referred to as *distributed-recovery systems*. Regardless of the recovery mechanism used, two repair policies can be adopted. 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 stored it on new peer upon recovery. The second policy is called *lazy* whose explicit goal is to delay repair work for as long as possible. This alternative policy 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.

Most of the existing P2P storage or backup 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 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.

In this thesis, we aimed at addressing fundamental design issues such as: *how to tune the system parameters so as to maximize data lifetime and availability while keeping a low storage overhead and achievable bandwidth use?*

For instance, Sébastien Choplin from UbiStorage informed us recently that UbiStorage is moving to the decentralized architecture to overcome the single point of failure problem and to obtain a more scalable backup and storage solution. They are implementing the storage protocol over FreePastry with the simplest erasure coding scheme (ReedSolomon [78]). He confirmed us that the choices of the key system parameters (s, r, and k) and the analysis of the recovery process are required to optimise the performance of the system and to better use its resources, especially when not only data backup but also data storage is wanted.

6.1.1 Delivered data lifetime and data availability

In Chapter 2, we have characterized the performance of P2P backup and storage systems in terms of the delivered data lifetime and data availability, when the *distributed-recovery* scheme

is enforced. Both repair policy (eager and lazy) were modeled and analyzed. We have investigated the distribution of the *lifetime* of data in a P2P system. We have evaluated two *availability* metrics: the expected number of available redundant fragments, and the fraction of time during which the number of available redundant fragment exceeds a given threshold. The impact of each system parameter on the performance was evaluated. Guidelines have been derived on how to engineer the system and tune its key parameters in order to provide desired lifetime and/or availability of data.

In particular, we have proposed two simple models in which the peer availability is considered to follow an exponential distribution. The recovery process was considered to follow an exponential distribution in the first simple model, and a hypo-exponential distribution in the second simple model. We have then extended the two simple models to more general ones by assuming that peers on-times durations, in both extended models, are hyper-exponentially distributed. Doing so, our modeling is general, realistic and valid under different distributed environments. A simple fluid model has been introduced under simple assumptions in order to have an explicit formula of the first availability metric.

Numerical computations have been performed to illustrate several issues of the performance. We found that, in stable environments such as local area or research laboratory networks where machines are usually highly available, the distributed-repair scheme (or "repair one missing fragment at a time" policy) offers a reliable, scalable and cheap storage/backup solution. This is in contrast with the case of highly dynamic environments, where the distributed-repair scheme, using erasure code (EC) or replication, is inefficient for the storage solution as long as the storage overhead is kept reasonable but it can offer the backup solution without any guarantee for the reconstruction time of the backup files. P2P storage systems may be applicable in highly dynamic environments with the distributed-repair scheme (or "repair one missing fragment" policy) by using the regenerating codes (RC) [35], whose description is introduced in Section 1.2.1, instead of erasure codes (e.g. [78]). However, the analysis of the overhead cost resulting from the different redundancy schemes with respect to their advantages (e.g. simplicity of replication, less storage space in both widely used EC: ReedSolomon [78] and Tornado [17], and less bandwidth cost of maintenance in RC), is an interesting issue to be addressed in the future.

In Chapter 3, we have focused our study on the quality of service delivered to each block of data in *centralized-repair* P2P backup and storage systems, in terms of data lifetime and availability. We evaluated such systems through Markovian models under similar assumptions of those made in Chapter 2. The impact of each system parameter on the performance is evaluated, and guidelines are derived on how to engineer the system and tune its key parameters in order to provide desired lifetime and/or availability of data. Our analysis also suggests that the use of large size fragments reduces the efficiency of the recovery mechanism. 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. A careful analysis of this issue is one objective of ongoing research.

6.1.2 Validation of key assumptions

The key assumptions made in the models presented in Chapter 2 and Chapter 3, in particular on the on-time durations of peers and on the recovery durations respectively, were validated through real traces collected from different distributed environments and intensive packet-level simulations.

Our findings in Chapter 4 came to support the assumptions made on the recovery process in the models presented in Chapter 2 and 3. The aim of Chapter 4 was the understanding of the behavior of P2P storage systems through simulation, while taking into consideration the impact of the heterogeneity of peers, the underlying network topologies, the propagation delays and the transport protocol. To that end, we have implemented the distributed storage protocol in the network simulator NS-2 [67] and run ten experiments covering a large variety of scenarios. We have shown through experimental results how recovery times distribution is impacted essentially by the demand and capacities of the complete network entities: peers upload/download, routers and links, in addition to the volume of the background traffic.

We have distinguished between three general scenarios in which the download and the recovery processes are modeled by a different distribution. In particular, in the first scenario, we have shown that the fragment download/upload time follows approximately an exponential distribution as long as the total workload is equally distributed over the active peers, the core network has a good connectivity and the peers upload/download capacities are asymmetric. We have found that the successive download durations are weakly correlated in such a scenario. We also show that, as a consequence of the fragment download distribution and the weakly correlation, the block download time and the recovery time essentially follow a hypo-exponential distribution with many distinct phases (maximum of as many exponentials). In the second scenario, we have shown that the fragment download/upload time follows a general phase type distribution but the block download time follows approximately an exponential distribution, under the following configurations: the peers upload/download capacities are symmetric and there are some bottlenecks in the backbone or among the local area network (LAN) routers. The characteristics of the third scenario are as follows. The peers are very heterogeneous and the volume of the non-P2P traffic is large with respect to the P2P traffic such that the total load in the system is very high. We found that, in such a scenario, both fragment and block successive download times are drawn from general phase type distribution. Contrarily to the first scenario, the successive fragment download times in such a scenario are strongly correlated. For all the experimental results in all the scenarios, we have used expectation maximization

and least square estimation algorithms to fit the empirical distributions. We also provided a good approximation of the number of phases of the hypo-exponential distribution that applies in several considered experiments. We tested the goodness of our fits using simple statistical (Kolmogorov-Smirnov test) and graphical methods.

6.1.3 First step toward a scalable simulator of the whole storage/backup system

Although the NS-based simulator can accurately predict the behavior of the download and the recovery processes while considering the impact of several constraints such as the heterogeneity of peers and the the underlying network topologies, this simulator requires however very long time and can not be used to simulate the whole storage system; data life-time and availability. To overcome this scalability limitation, we have proposed and analyzed, in Chapter 5, an algorithm, that we called the "progressive-filling flow-level algorithm" or PFFLA. The algorithm is efficient in time and quite simple and uses the concept of "Progressive-Filling" (or max-min fairness), hence the name. We have implemented the PFFLA in a flow-level simulator of parallel downloads. The response times in the flow level simulator have been compared (both distributions and averages) to that of the packet-level simulations in NS using a modified model of the simulator presented in Chapter 4. Our results concluded that PFFLA is a reliable algorithm to analyze the service response time in many systems based on P2P and Grid computing concepts such as Storage Systems and Grid Delivery Networks.

6.2 Future work

Several markovian models are proposed in this thesis where most of the assumptions are validated through the analysis of data collected from real environments or from the output of stochastic simulations. Using an appropriate model for a given context, we can evaluate the performance of the storage system in terms of data lifetime and availability and tune its parameters to provide desired requirements. However, we would like to address several issues in the next steps.

First, we have seen in Section 3.6.3 that increasing s, using smaller fragments size, increases the efficiency of the recovery mechanism and then the performance of the whole storage application. This observation should be moderated by the fact that different fragments size yielding a different bandwidth usage per recovery. We would like to search how does the bandwidth usage per recovery vary with the size of fragments while fixing the size of blocks of data. Moreover, when the number of fragments increases, more coordination and control overhead have to be handled by the system. A careful analysis of this issue is still required in order to optimize the system performance.

Another interesting issue is the analysis of the overhead cost resulting from the different redundancy schemes with respect to their advantages; understanding the complexity versus the advantages of *Tornado* [17] and *regenerating codes* [35] with respect to *replication* and the simplest erasure coding scheme *ReedSolomon* [78]. This can be done through real experiments using *Tahoe-LAFS* implementation [95] or by collaborating, for example, with *UbiStorage* that has shared with us several discussions about some theoritical and practical issues related to the P2P backup applications. An other method is to use a well-validated and scalable simulator for these systems. Let us hereafter explain our prespective.

Deploying the system over a large number of peers and achieving real experiments over real networks are very useful in order to carefully verify that the system performs as expected from modeling or simulation. Such experiments will also help us to better understand the impact of several neglected factors in modeling and simulation such as the complexity overhead to maintain the state of the system. However, to collect traces of fragment download/upload times, of recovery times, and of data durability in such particular applications would require a very long time, and there will be limitations on changing the topology and the peers characteristics. Hence, the importance of using a "very realistic" simulator at reasonable scale (e.g. simulating the systems using a "real implementation"), but how to do that? And is it possible?

Typically, simulators re-implement protocols from scratch, leading to a costly software effort and divergence from actual implementation code. Furthermore, simulation code often does not interact well with real implementations. Most commonly, simulation implementations of protocols are rewritten for use as implementation code, often because the simulation code makes use of abstractions and simplifications which are not present in real systems. Lacage is working on a project [61] to realize one of the NS-3 goals [51] that facilitates, on one hand, the reuse of existing softwares and applications under some compilation's conditions (e.g. a release of a storage protocol such as UbiStorage or Tahoe), and, on the other hand, provides Interfaces to allow users to easily migrate between simulation and network emulation environments. In particular, the goal of this project is to allow the integration of unmodified POSIX application binaries in ns-3. This requires the implementation of the relevant parts of the POSIX API used by the target applications but also the implementation of a specialized ELF loader to load multiple per-node applications within the same simulation process. ELF stands for "Executable and Linking Format", formerly called Extensible Linking Format which is a common standard file format for executables, object code, shared libraries, and core dumps.

In summary, we can validate a scalable stochastic simulator (e.g. a modified version of the Algorithm PFFLA 1 that introduced in Chapter 5) through a real implementation using NS-3. A stochastic simulator will be then used to simulate, in an efficient and reasonable time, the storage protocol under any given topology and network/peers characteristics. For the moment, we are planning to extend the current version of the Algorithm PFFLA 1 that was designed for

flows with identical RTTs. In fact, a conclusion from the literature is that different RTTs do introduce some "unfairness" in bandwidth allocations. Our next step is to find a simple yet efficient version of Algorithm 1 to handle the situation of different RTTs that introduces some "unfairness" in bandwidth allocations.

Another issue is to introduce server and peer selection policies in the algorithm for an enhanced service. Bernard and Le Fessant have concluded in [6] that by carefully selecting the peers on which backup data is stored, repairing cost can be highly reduced while providing high durability level. We are planning to use the technique of "fluid-diffusive" for providing simple formulas of some performance metrics for the P2P storage systems that use some placement policies. The "fluid-diffusive" approach has been proposed recently to be an alternative efficient, in term of time, technique to model large P2P systems. Recently, Carofiglio *et al*. in [19] have used this approach to analyze and to model general P2P systems. Unlike the first-order fluid models (e.g. those proposed in Sections 2.4.2 and 3.3.2), the model proposed in [19] captures the impact of stochasticity on the system dynamics based on a set of partial differential equations. For more accurate prediction, we have to account for the time needed to locate a specific content in the system, in particular when the system is large. One can rely on the work done by Oechsner and Tran-Gia [70] in this direction.

A

PACKET-LEVEL SIMULATION MODEL: SOME IMPLEMENTATION DETAILS

This appendix describes the base classes P2P_Storage_Directory, P2P_Storage_App, P2P_Storage_Wrapper and data structure. In fact, we follow the same methodology as the Web cash application presented in the NS Manual (cf. [39, Chap. 40]), and use some of the technical ideas presented in [39, Chap. 39,41] of the NS Manual and [37]. Therefore, we will discuss some selected pieces of code and sketch the description of the basic APIs, through which applications find data and request services from underlying transport NS agents. We implemented the P2P storage application in NS-2 (versions 2.29 and 2.33) following the architecture depicted in Fig. A.1. P2P_Storage_App class emulats a P2P storage application that takes care of associating peers to NS-2 Nodes, handling messages, choosing files to be downloaded, requesting list of peers holding desired data from P2P_Storage_Directory object, sending requests to those peers, registering some information (e.g. download start times, finish times, total sending data, effective load) in logs files, and joining/leaving the system. P2P_Storage_Wrapper object is an intermediate class that takes care of creating the TCP connections between applications and passing data in an appropriate type between the FullTcp transport agent object in NS-2 and the P2P_Storage_App class. P2P_Storage_Directory object is the "God" of the simulator. It registers

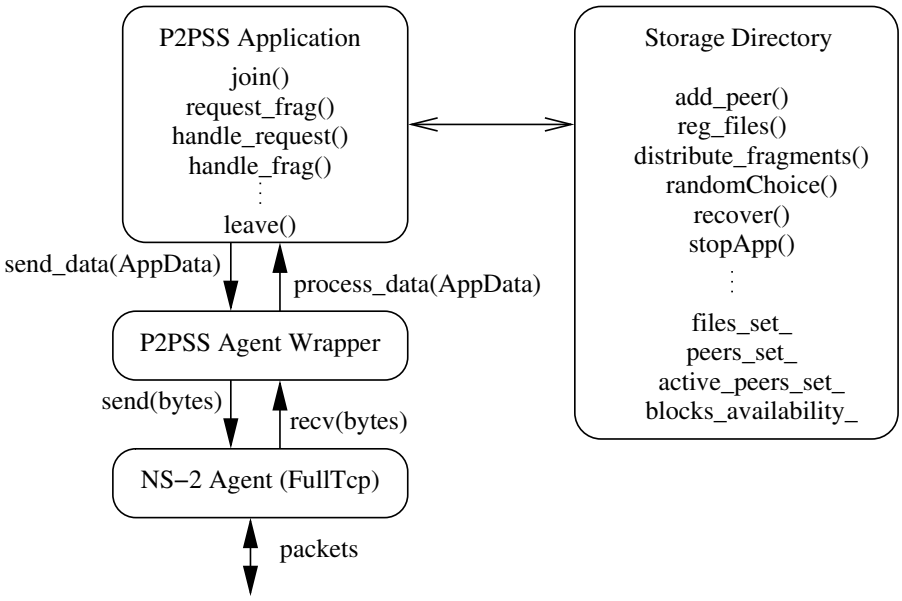


Figure A.1: Simulator architecture.

applications (peers) and files, distributes their fragments in the system, memorize the locations of fragments of each block for every registred files, calculates the availability of each block and maintains a list of the active peers (running applications).

Similarly to any simulation that uses NS, we define the network topology and the system parameters (such as maximum number of files, maximum number of peers, upload/download capacities, delays, block size, fragment size, TCP segemt size, amount of redundancy and the recovery threshold) at OTcl level in a TCL script as follows.

Listing A.1: Simulation scenario setup

```
set NS [new Simulator]
# Number of peers
        set N_P 1000
        set Inter_arrival_req 3
# Number of files
        set N_F 10000
        set max_request 1000
```

```
# Overhead storage r/s
        set oh_st 1.5
# Recovery threshold k
set k_th 1
# Application type, e-library-like=1
# Buckup-like type = 0
set app_type 1
# Data unit sizes
        set S_F_KB [expr 16 *1024]
        set S_b_KB [expr 4 *1024]
        set S_Frag_KB [expr 1 *1024]
# Set MSS for TCP
Agent/TCP/FullTcp set segsize_ 1460
# Peers upload capacities in bps
set C_up_max [expr 384 * 1000]
set C_up_min [expr 64 * 1000]

# Peers download capacity is higher than upload capacity
set C_down_max [expr $C_up_max * 8]
set C_down_min [expr $C_up_min * 8]
# Bandwidth (BW) between routers of AS 34Mbps to 155Mbps
set linkBW_min [expr 34 * 1000000]
set linkBW_max [expr 155 * 1000000]
# BW between bacbone is 1 Gbps
set backbone [expr 1000 * 1000000]
# BW between backbone and transit acess to routers is 622 Mbps
set transit_access [expr 622 * 1000000]
# Create instance of system directory
```

```
set dir [new P2P_Storage_Directory $N_P $N_F S_F_KB S_b_KB S_Frag_KB
                                        $oh_st $Inter_arrival_req]
proc create_router_topology {} {
        global ns linkBW_min linkBW_max transit_access backbone
        .
        .
        .
}
#Creat peers of each sub domain
proc creat_peers_topology {router_index peer_index} {
        global ns peer n C_up_min C_up_max outdir
        set DelayMin 1
        set DelayMax 25
        # random delay choices
        set D [new RNG]
        set Delay [expr round([$D uniform $DelayMin $DelayMax])]
        .
        .
        .
}
        .
        .
        .
# Start applications
for {set i 0} {$i < $N_P} {incr i} {

        $ns at [$ns now] "$app($i) start"
}
```

```
# new MashInspector
$ns at 500000 "finish"
# Run the simulation
$ns run
```

We instantiate then from P2P_Storage_Directory class, which is implemented as a child class of TclObject, as shown in Listing A.2, the system directory object. The basic function members of the class P2P_Storage_Directory are found in Table A.1.

Listing A.2: Definition of the system directory class

```
class P2P_Storage_Directory : public TclObject
{

public:

// The Constructor of the class
        P2P_Storage_Directory(int num_peers,int num_files,
                long file_size, long block_size, long fragment_size,
                double storage_overhead, double mean_interval););
        ...
protected:
// Tcl command interpreter
        int command(int argc,const char*const* argv);
        void reg_peer(Node* peer);
        void reg_file(file_list_entry* file);
        void del_peer(Node* peer);
        void stopApps();
        void go_off(long id, Node * node_);
        void go_on(long id, Node * node_);
```

```
        int randomChoice( int min, int max );
        double exponential(double mean_periode);
        virtual void distribut_fragments();
        void modify_peer_state(int peer_index,int changed_value);
        ...
};
```

To make it possible to create an instance of the system directory object in OTcl, we have to define a linkage object that must be derived from TclClass. This is illustrated in Listing A.3. In fact, once NS is started, it executes the constructor for the static variable "class_p2p_storage_directory", and thus an instance of "P2P_Storage_DirectoryClass" is created.

Listing A.3: The linkage object P2P_Storage_DirectoryClass between OTCL and C++ class P2P_Storage_Directory

```
static class P2P_Storage_DirectoryClass :
                        public TclClass
{
public:
        P2P_Storage_DirectoryClass() :
        TclClass("P2P_Storage_Directory") {}
        TclObject* create(int argc, ...)
                {
                        if (argc != 11)
                                return NULL;
                        else
                                return (new P2P_Storage_Directory(
                                atol(argv[4]),atol(argv[5]),
                                atol(argv[6]), atol(argv[7]),
                                atol(argv[8]), atof(argv[9]),
```

```
                                    atof(argv[10]) ) );
                ...
                }
} class_p2p_storage_directory;
```

We assume that there is a given number of stored files in the system and before that peers request data, the system directory object distributes the $s + r$ fragments of each block of data of all files over $s + r$ peers chosen uniformly among all the registered peers in the system. This is the task of the member functions "reg_file()" and "distribute_fragments()", where the system directory has a private vector containing pointers to the meta-data of the stored files. Listings A.4 and A.5 depict the details of the meta-data (file structure) of any stored file and the member files_set_ respectively. We use the same class of the system directory for both recovery process implementations and we assume that the system has a perfect knowledge of the data state.

Listing A.4: The file_list_entry data structure

```
typedef struct file_list_entry {
     int file_id;
     long file_size;
     long block_size;
     long frag_size;
     int N_blocks; // file_size/block_size
     int N_frags; // s
     int total_N_frags;//s+r

/* Map between each block' id (key) and a list of s+r peer's id,
   on which the fragments are stored */
     map<int,int *> >block_node_id_list_;
     map<int,vector<Node*> >block_node_list_;
     map<int,int> blocks_availability;
```

```
    map<int, bool> black_list;//false==lost
};
```

Listing A.5: The private member files_set_ of the P2P_Storage_Directory class

```
vector<file_list_entry*> files_set_;
```

After creating the instance of the storage directory at the OTcl level, we allocate next the NS nodes and we create the underlying network topology by using for example the GT-ITM tool [18] (see more details in Section 4.3.1). We instantiate from P2P_Storage_App class the applications (peers) where a pointer at the Node class must be set to the attached application running on that Node (Agent) which will be used to pass data from an Agent to an Application. In fact, we did minor changes to the files: tcp-full.cc, tcp-full.h, node.cc, node.h, agent.cc and agent.h to support the collaboration between nodes, agents and the P2P storage systems applications.

Listing A.6: OTcl level, creating nodes and application

```
set node($i) [$ns node]
        ...
set app($i) [new P2P_Storage_App $dir
                $node($i) $app_type $C_up($i)
                $C_down($i) $Inter_arrival_req max_request]
        ...
```

We consider in fact two different storage applications, a backup-like application and an e-library-like application (“e” stands for “electronic”). In the first, a file stored in the system can be requested for retrieval only by the peer that has produced the file. In the second, any file can be downloaded by any peer in the system. In both applications, the storage protocol follows the description presented in Sections 2.2 and 3.2.

Two types of requests are issued in the system. The first type is issued by the users of the system: a user issues a request to retrieve its backup file in the backup-like application, or a public document in the e-library-like application. The second type consists of management requests. Usually, these are issued by the central authority (in the centralized implementation of

the recovery process) or by a peer (in the distributed implementation) as soon as the threshold k is reached for any stored block of data. In the simulator, these management requests are issued by the system storage directory object.

File download requests are translated into (i) a request to the directory service to obtain, for each block of the desired file, a list of at least s peers that store fragments of this block, (ii) opening TCP connections with each peer in the said list to download one fragment, (iii) registering some statistical information such as the start and the completion time of the downloaded data. All download requests issued by a given peer form a Poisson process. This assumption is met in real networks as found in [47]. However, another random process can easily implemented.

Recovery requests are issued only in the scenarios where there is churn in the network. A recovery request concerning a given block translates into (i) a request from the storage directory service to a server in the centralized-repair scheme (we consider explicitly the first registered peer as the server in order to simulate the centralized implementation) or any active peer that is in charge of (ii) obtaining a list of at least s peers that store fragments of said block, (iii) opening TCP connections with each peer in the said list to download one fragment. Once all s fragments have been downloaded, the process proceeds with Steps 2 and 3, according to the implementation, as explained in Sections 2.2 and 3.2. Last, the storage directory updates the system state at the end of the operation, namely it increases the availability level of the blocks of interest and points to the right locations of its fragments or otherwise it adds the lost block in a black list if the operation failed.

The P2P storage application uses many timers to handle events. In particular, a timer for scheduling the next file's request, a timer for scheduling the next failure moment once a peer becomes on line, and a timer for scheduling the next moment to rejoin the system once a peer becomes off line. We define the FileRequestTimer, OffLineTimer and OnLineTimer classes that are derived from the "TimerHandler" class, and write their "expire()" member functions to call file_request(), leave() and join() APIs respectively. Then, we included an instance of each timer object as a private member of P2P_Storage_App object. Listings A.7 and A.8 show the example of FileRequestTimer and its expire member function implementation.

Listing A.7: FileRequestTimer implementation

```
class FileRequestTimer:public TimerHandler
{
protected:
        P2P_Storage_App *app_;

public:
       FileRequestTimer(P2P_Storage_App* app):
                TimerHandler(), app_(app) {}
       inline virtual void expire(Event *);

};
```

Listing A.8: Expire function of FileRequestTimer

```
void FileRequestTimer::expire(Event *) {
        app_->file_request();
}
```

Typically, applications access network services through sockets. NS-2 provides a set of well-defined API functions in the transport agent to simulate the behavior of the real sockets. Therefore, the P2P_Storage_Wrapper class handles calling the appropriate APIs when two applications want to communicate in order to (i) attache first the Full Tcp agent to both NS nodes via attach-agent and (ii) call then connect() instproc to set each agent's destination target to the other and last (iii) place one of them in LISTEN mode. We use in fact Full-Tcp agents since they support bidirectional data transfers.

Similar to what is done in the web cash application (see tcpapp.cc) we can model the underlying TCP connections as a FIFO byte stream, and then we will create same buffer management stuff. First, the P2P_Storage_Ms_Buf that contains a part of the messages such as the Request message and the Fragment message. Second, P2P_Storage_Msg_BufList implements a FIFO queues that will store all the sent messages (requests or data) on the sender side until

they correctly and completely arrive to the destination side. In other words, there is no support in the class “Agent” to transmit different applications data and messages. Instead, as all data are delivered in sequence, we can view the TCP connections as a FIFO pipes, and the transfer of the application data will be emulated as follows. We first provide buffer for the application data at the sender to store the messages to be sent, next we use the Agent’s API “sendmsg(int nbytes, const char *flags = 0)” to send a stream of an equivalent data size of the stored messages, then we count the bytes received at the destination. When the receiver has got all bytes of the current data or message transmission (first message in the FIFO on the sender side toward the receiver), then the receiver gets the data directly from the FIFO’s sender. These are the tasks of the functions “send_data()” and “send()” on the sender application side and “process_data()” and “recv()” on the receiver side as shown in Fig.A.1 and described in Table A.3, which use in turn the prototypes of the FIFO queues depicted in Table A.2, where a FIFO queue is represented by the P2P_Storage_Msg_BufList class.

Table A.1: The basic prototypes of P2P_Storage_Directory class

Method	Functionality
map <int,vector<Node*≫ get_peer_list_(int file_ID)	gets a list of peers (s usually) for each block to download a specific file
void add_peer(Node* peer)	adds new peer to the directory
void reg_file(file_list_entry* file)	adds the file entry to the files_set_
void stopApps()	stops all the applications and frees the memory when the maximum simulation time or the maximum number of requests are reached
void del_peer(Node* peer)	deletes peer from active_Peer_set_ when leaving the system
void go_off(long id, Node * node_)	reduces the blocks availability, checks the recovery threshold, del_peer
void go_on(long id, Node * node_)	increases the blocks availability if not recovered, add_active_peer
int randomChoice(int min, int max)	chooses an active peer randomly
virtual void distribute_fragments()	distributes the fragments of blocks of the registered files
bool recovery(int block_id, int missing)	recovers missing fragments of a given block

Table A.2: The basic prototypes of P2P_Storage_Msg_BufList class

Method	Functionality
void insert(P2P_Storage_Msg_Buf *d)	stores msgs of the sender until the reception of their acks
P2P_Storage_Msg_Buf* detach()	if the data is received by the destination, deletes them from the FIFO buffer
int size()	returns the current size of the buffer

Table A.3: The basic prototypes of P2P_Storage_App and P2P_Storage_Wrapper classes

Method	Functionality
virtual void start()	after calling the constructor, App starts requesting files with an inter-request times chosen from an exponential distribution
double exponential(double lambda)	generates a random number from an exponential distribution
int create_conn(Node *dst,int file_id, int block_id, int frag_id)	establishes a connection with the destination
virtual void send_data(P2P_Storage_Msg m, int s_id, int dst_id)	Application sends msg to the wrapper agent
virtual void send(int nbytes)	wrapper agent calls sendmsg() of the tcp agent
void recv(int nbytes, int socket_id)	the NS agent announces the App each time a packet arrives
void process_data(P2P_Storage_Msg msg, int s_id)	handles the received data
void file_request(int file_id)	creates connections and sends requests to get the file after calling the Directory member function get_peer_list_(int file_id)
void request_frag(int conn_id, int f_id, b_id, int fr_id, int dst_id)	requests a fragment from a peer
void handle_request(P2P_Storage_Msg m, int conn_id)	handles a request, creates a fragment message and sends it
void handle_frag(P2P_Storage_Msg m, int conn_id)	called when the receiver gets all bytes of the current transmission, updates the related members, increases the number of completed fragments, calls reg_frag_traces() and frag_downloaded()
void reg_frag_traces(int file_id, int b_id, int fr_id, int dst_id)	registers the information about a completely received fragment
void frag_downloaded(int f, int b, int frag, int conn,int dst)	after downloading a fragment, calls close_connections()
void join()	informs the directory the active state

B

Approximations with Processor Sharing

B.1 Distribution of the response time in the M/D/1/PS queue

According to Yashkov and Yashkova [100, Corollary 2.11] the distribution of the response time in a M/D/1/PS queue with arrival rate λ and service time d, say $V(d)$, is given by its Laplace transform as:

$$E(e^{-sV(d)}) = (1-\rho)\,\frac{(s+\lambda)^2 e^{-d(s+\lambda)}}{s^2+\lambda(s+(s+\lambda)(1-\rho))e^{-d(s+\lambda)}},$$

where the load factor is $\rho = \lambda d$. The inversion of this Laplace transform yields the series:

$$\begin{aligned} & P(V(d) \leq d + t) \qquad (B.1) \\ = \; & (1-\rho)e^{-\rho} \sum_{n=0}^{\infty} (-1)^n e^{-n\rho} 1_{\{t \geq nd\}} \\ & \sum_{m=0}^{n} \binom{n}{m} (2-\rho)^m (1-\rho)^{n-m} \frac{[\lambda(t-nd)]^{2n-m}}{(2n-m)!} \\ & \left[1 + 2\lambda \frac{t-nd}{2n-m+1} + \frac{\lambda^2 (t-nd)^2}{(2n-m+1)(2n-m+2)} \right] . \end{aligned}$$

For every $t \in [0, 2d]$, this formula reduces to:

$$\begin{aligned} & P(V(d) \leq d + t) \qquad (B.2) \\ = \; & (1-\rho)e^{-\rho} \left[1 + 2\lambda t + \frac{\lambda^2}{2} t^2 \right] \\ & + (1-\rho)e^{-2\rho} 1_{\{t \geq d\}} \left\{ (1-\rho) \frac{\lambda^2}{2} (t-d)^2 \left[1 + \frac{2}{3}\lambda(t-d) + \frac{\lambda^2}{12}(t-d)^2 \right] \right. \\ & \left. (1-2\rho)\lambda(t-d) \left[1 + \lambda(t-d) + \frac{\lambda^2}{6}(t-d)^2 \right] \right\} . \end{aligned}$$

B.2 Small load approximations

We briefly discuss now approximations that can be performed when the load or the arrival rate is small.

Consider a client downloading s flows in parallel. The nominal duration of each flow is σ/s, so that the total duration is σ if the flows are not disturbed.

Assume that the arrival of flows is a Poisson process of rate λ at all nodes: at the client node and the server nodes. Given some "tagged" download request, the probability that another request interferes with it *at the client* is $e^{-\lambda \times (2d)} = e^{-2\rho}$ because the request interferes if it arrives less than d units of time before or after the arrival of our tagged request. The same probability holds at each server.

The result of a request interfering *at the client* is that the tagged download is longer. If the arrival date of the interfering request relative to the tagged request is u, the additional response time of the tagged request is $d - |u|$.

The result of a request interfering *at the server* depends on the capacity of the server. If the capacity is enough, the interference will *not* slow down the flow, and the response time will not change. This is the case for the symmetric cases in our experiments. If the capacity is not enough, the flow will be slowed down. Take the case where the capacity of servers is precisely that of one of the s parallel flows. This is the case for asymmetric cases in our experiments. If an interference occurs at the server, the flow will be slowed down to half its throughput. The result is then exactly the same as when two requests interfere at the client.

Suppose now that the arrival rate λ and the load ρ are small. Ignoring the events that happen with probability $o(\rho)$, only three events are to be considered: a) no interferences; b) one single interference at the client, none at the server; c) one single interference at the server, none at the client. According to the discussion above, we can calculate the statistics of the response time T in the two situations:

Large server capacity: the probability that $T = d$ is the probability of events a) and c), since c) does not have an influence on T. This probability is: $e^{-2\rho} = 1 - 2\rho + o(\rho)$. For $x \in [0, d]$, $P(T > d + x) = P(\text{ b) and } |u| < d - x) = 2\rho(1 - x/d)$, and $E[T] = d(1 + \rho)$. These formulas are in accordance with Equations B.2 and (5.6). The prediction that $P(T > d) \sim 2\rho$ can be observed on Figures 5.1 to 5.4. The linear behavior of the distribution of $P(T > x)$ for $x \in [d, 2d]$ is also clear on these figures.

Minimal server capacity: the probability that $T = d$ is the probability of event a), that is, $(e^{-2\rho})^2 = 1 - 4\rho + o(\rho)$. For $x \in [0, d]$, $P(T > d + x) = P(\text{ b) or c) and } |u| < d - x) = 4\rho(1 - x/d)$, and $E[T] = d(1 + 2\rho)$. These formulas are not in accordance anymore with the PS queueing model. This explains the bad results of the approximation in Table 5.2 for asymmetric cases. At the same time, this suggests a possible correction for the PS approximation formula. The prediction that $P(T > d) \sim 4\rho$ can however be observed on Figures 5.7 and 5.8(a) (with $\rho = 12\%$). The almost-linear behavior of the distribution of $P(T > x)$ for $x \in [d, 2d]$ is also clear on these figures.

C

Présentation des Travaux de Thèse en Francais

C.1 Introduction

La croissance du volume de stockage, de la bande passante et de la puissance de calcul des ordinateurs personnels a changé fondamentalement la méthode de conception des applications. Depuis près de dix ans, un nouveau paradigme pour les réseaux a été proposé, où les ordinateurs peuvent constituer un réseau virtuel (appelée réseau de recouvrement ou un overlay) au-dessus d'un autre réseau ou d'une architecture existante (par exemple Internet). Ce nouveau paradigme a été appelé réseau distribué pair-à-pair (P2P). Un pair dans ce paradigme est un ordinateur qui joue le double rôle de fournisseur et de consommateur de ressources, contrairement au modèle traditionnel client-serveur où les serveurs fournissent les ressources, et les clients les consomment. Conceptuellement, les applications utilisant les réseaux P2P présentent une amélioration notable dans la robustesse des services et permettent un passage à l'échelle stable sans infrastructure supplémentaire puisque tous les pairs connectés fournissent

quelques ressources et services. Un pair dans le réseau de recouvrement (l'overlay) peut être considéré comme étant relié par des liens virtuels ou logiques, chacun d'eux correspondant à un chemin qui peut-être composé de plusieurs liens physiques, dans le réseau sous-jacent. Comme déjà mentionné, chaque pair demande/propose un service de/vers les autres pairs à travers le réseau overlay; des exemples de ces services sont le calcul (partage de la capacité de son unité centrale), le téléchargement des données (partage de sa capacité de bande passante), le stockage des données (partager son espace de stockage gratuit), ainsi que l'aide pour trouver des ressources, services et autres pairs.

Ce modèle P2P s'est avéré être une alternative au modèle Client/Serveur et être un paradigme prometteur pour le calcul sur la grille "grid computing", le partage de fichiers, la voix sur IP, les applications de sauvegarde et de stockage. Toutefois, le partage de fichiers est l'application P2P dominante sur l'Internet (voir [66, 55, 41, 38, 24, 25]), permettant aux utilisateurs de contribuer, rechercher et obtenir le contenu facilement. En effet, depuis l'apparition du format mp3 en 1991, les applications P2P présentaient une solution efficace pour les partager gratuitement, d'où la popularité croissante de ces applications. De plus, l'apparition des formats de vidéos comprissées (divx) a encore augmenté leurs intérêts.

Afin de fournir une base appropriée, nous allons décrire brièvement, dans la section C.2, la taxonomie de base du réseau overlay P2P. Etant donnée la grande popularité des applications P2P de partage de fichiers, nous allons introduire certaines d'entre elles comme des exemples de l'utilisation des architectures P2P, même si elles ne font pas l'objet d'une étude plus approfondie dans cette thèse. Les techniques de sauvegarde et de stockage des systèmes de P2P seront présentées alors avec quelques exemples existants dans la Section C.3.3. La Section C.4 présente l'état de l'art et les motivations. Enfin, la Section C.5 présente brièvement notre contribution décrite dans cette thèse.

C.2 Architectures de réseau de recouvrement P2P

En consiérant la façon dont les pairs dans le réseau de recouvrement sont liés les uns aux autres au dessus de la topologie physique du réseau, et la façon dont les services sont partagés et localisés, nous pouvons classer les réseaux P2P en deux classes de topologies: les réseaux non-structurés et les réseaux structurés.

C.2.1 Réseau P2P non-structuré

Un réseau P2P non structuré organise les pairs ou les noeuds dans une topologie de graphe aléatoir et utilise les techniques d'inondation ou les marches aléatoires pour découvrir les données stockées par les noeuds qui sont connectés à l'overlay. En d'autres termes, les pairs se connectent à l'overlay sans préoccuper des noms ou des identifiants de leurs voisins. Cette approche prend en charge des requêtes arbitrairement complexes et n'impose pas de contrainte sur la topologie de l'overlay ou sur le placement des données.

En général, trois topologies non structurées peuvent être distinguées.

Premièrement, il y a les systèmes P2P entièrement distribués, comme le protocole original de **Gnutella** [41], où tous les pairs sont parfaitement égaux et il n'existe aucune autorité centrale. Dès qu'un pair rejoint le système, il établit plusieurs connexions avec quelques autres pairs, appelés voisins. Pour rechercher une entité dans le système, un pair envoie une requête à ses voisins. Si un voisin connaît l'entité demandée, il répond au demandeur. Sinon, il transmet la requête à ses propres voisins, et ainsi de suite jusqu'à une profondeur donnée. Cette profondeur est similaire au temps de vie (TTL) des paquets dans les réseaux IP. Ce type de recherche est appelé inondation. Cependant, le coût de l'inondation du réseau augmente de façon linéaire avec le nombre de pairs ce qui limite le passage à l'échelle du système si la profondeur est élevée. En outre, il n'y a aucune garantie sur le temps de réponse, en particulier, pour les fichiers non populaires.

Deuxièmement, les systèmes P2P hybrides, tels que **Kazaa** [55], utilisent le concept de supernoeuds: les noeuds qui gèrent l'indexation des pairs, la distribution et la localisation

des blocs de données. Des supernoeuds sont élus de manière dynamique en fonction de la capacité de bande passante et de la puissance de traitement des noeuds. Toutes les requêtes sont initialement transmises aux supernoeuds pour recevoir un service. Par conséquent, le temps de découverte des resources et des services est réduit par rapport à des systèmes entièrement décentralisés. Il n'y a pas de point unique de défaillance comme dans le cas des systèmes centralisés (expliqué ci-dessous) et il n'est pas nécessaire d'acheminer des messages par des inondations comme dans le cas des systèmes entièrement distribués.

Troisièmement, les systèmes centralisés, comme **Napster** [66], reposent sur un serveur central pour les fonctions d'indexation et pour le "bootstrap" de l'ensemble du système. En fait, Napster a été le premier système P2P de partage de fichiers, utilisé notamment pour le partage des fichiers de musique. Bien que considéré comme un système P2P, ce modèle suit le modèle client-serveur, car il utilise un serveur central pour maintenir une liste des répertoires et des fichiers partagés qui sont stockés sur les pairs, et pour trouver des ressources et router les demandes entre les pairs. Toutefois, le téléchargement se produit de manière P2P; les pairs se connectent les uns aux autres pour télécharger des fragments de données. Cette topologie souffre du fait qu'elle a un point de défaillance (l'autorité centrale) et ne peut pas passer à grande échelle.

C.2.2 Réseau P2P structuré

Dans les réseaux P2P structurés, les noeuds se voient attribuer un nodeId aléatoire unique (identifiant de noeud) à partir d'un espace d'identification assez large. Des identifiants uniques, appelés "clés" sont assignés aux objets de données. Ils sont choisis dans le même espace d'identification. Chord [89], Tapestry [101] et Pastry [83] utilisent un espace d'identification circulaire de n-bits entiers modulo 2^n-bits ($n = 160$ pour Chord et Tapestry, et $n = 128$ pour Pastry). Si n est grand, l'overlay attribue dynamiquement chaque clé à un unique noeud actif avec une probabilité très élevée. Ce noeud est appelé la racine de la clé, ou le noeud responsable de la clé. Afin de délivrer des messages de manière efficace à la racine, chaque noeud maintient une table de routage comprenant les nodeIds et les adresses IP des noeuds vers lesquels le

noeud local maintient des liens virtuels (liens d'overlay). Les messages sont envoyés à travers ces liens virtuels aux noeuds dont les nodeIds sont progressivement de plus en plus près de la clé dans l'espace d'identification.

Les réseaux P2P structurés utilisent une fonction de hachage (e.g. SHA-1 [87, 86]) pour allouer une adresse globale ou un espace d'identification à tous les noeuds et à toutes les clés. Contrairement aux réseaux P2P non structurés, le concept principal dans les réseaux structurés est le routage à base de clé. Le routage à base de clé signifie qu'un ensemble de clés est associé à des "valeurs" (adresses des données) dans l'espace d'adressage. un réseau P2P structuré est souvent considéré comme une table de hachage distribuée (DHT), qui est un dictionnaire distribué dans lequel chaque entrée est composée d'une "clé" et d'une "valeur" associée qui indique l'endroit du contenu de la clé demandée.

Dans les applications P2P, trois approches d'organisation de données peuvent être considérées. D'abord, une "clé" peut être l'identifiant de l'ensemble du fichier de données (valeur de hachage de son nom, ou de son titre, ou de son contenu) comme dans PAST [84], un utilitaire de stockage persistant qui a été construit en utilisant Pastry [83]. Dans d'autres applications P2P, les fichiers sont fragmentés en fragments de taille égale, et une "clé" est l'identifiant d'un fragment de données d'un fichier comme dans le cas de CFS [27], un système de fichiers coopératif qui a été construit à l'aide de Chord [89] afin de fournir des services de stockage. Un troisième type d'organisation de données consiste à diviser les fichiers en blocs de données de même taille, chaque bloc est ensuite fragmenté en plusieurs fragments de même taille, et la "clé", dans cette dernière approche, sera l'identifiant d'un bloc de données comme dans le cas de UbiStorage [92], un système P2P de sauvegarde de données. Chacune de ces approches d'organisation des données a ses avantages et ses inconvénients. Les objectifs du système, le temps de téléchargement des données, la disponibilité et la mise en oeuvre du système et les problèmes de conception peuvent favoriser une approche plutôt qu'une autre selon les cas particuliers.

C.3 Systèmes P2P de stockage et de sauvegarde des données

Parallèlement à l'évolution des systèmes P2P de partage de fichiers, les systèmes P2P de stockage et de sauvegarde des données ont été développés. Ils sont moins populaires parce qu'ils ne sont pas consacrés exclusivement au partage de musique ou de vidéos et parce que les gens ne font pas confiance au P2P pour stocker leurs données privées. Pour une utilisation ultérieure, nous allons définir deux mesures importantes, la disponibilité des données, et la durée de vie des données ou longévité des données.

Dans le temps, un pair ou un noeud peut être soit *connecté* soit *déconnecté* du système de stockage. Nous faisons allusion à *on-time* (resp. *off-time*), comme un intervalle de temps pendant lequel un pair est toujours connecté (resp. déconnecté). Pendant une période donnée, nous pouvons représenter la disponibilité d'un noeud par le pourcentage de la somme des durées *on-time* durant cette période. Donc, à n'importe quel moment de cette période, un pair ne peut être disponible qu'avec une certaine probabilité. Au cours d'une durée off-time d'un pair, les objets de données, qui sont stockés sur ce pair, sont momentanément indisponibles pour les utilisateurs du système. En conséquence, un objet de données peut être disponible à tout moment avec une certaine probabilité qui est liée à la disponibilité du noeud qui le stocke. Pour pouvoir télécharger un élément de données, un noeud qui stocke une copie complète de celui-ci ou un nombre suffisant de noeuds qui stockent ses fragments distincts doivent être actifs (connectés au système) pendant un certain temps.

Certains éléments de données (ou des fragments de ces éléments) peuvent être perdus du système à cause de départs définitifs de certains noeuds ou de défaillances des disques. Nous définissons la durée de vie des données comme la période allant jusqu'au moment où les données sont considérées être perdues (ne peuvent plus être téléchargées, ou reconstruites complètement). Ainsi, avant que les données ne soient perdues, ces données peuvent être disponibles ou non disponibles temporairement, mais ells sont durables (pas perdues définitivement).

Nous distinguons les systèmes de sauvegarde des systèmes de stockage. Les systèmes P2P

de sauvegarde visent à fournir la longévité des données sans contrainte sur le temps de reconstruction des données (la disponibilité). En d'autres termes, les données doivent être durablement stockées, mais pas nécessairement immédiatement disponibles pour le téléchargement, au contraire des systèmes de stockage. Pour cette raison, les concepteurs de systèmes de sauvegarde sont intéressés par les départs permanents des pairs plutôt que par les déconnexions intermédiaires, même si les durées de déconnexions sont longues. Dans cette thèse, nous allons fournir des modèles qui permettent d'évaluer l'impact de chaque paramètre du système sur les performances. En particulier, nous montrons comment nos résultats peuvent être utilisés pour garantir la qualité de service de la durée de vie des données et/ou de la disponibilité. Certains des efforts récents pour construire des systèmes extrêmement disponibles et durables basés sur le paradigme P2P incluent Intermemory [45, 21], Freenet [24], OceanStore [59], CFS [27], PAST [84], Farsite [14, 1], Total Recall [10], Wua.la [99] and Allmydata [2]. Bien que ces systèmes de stockage soient évolutifs, tolérants contre les catastrophes inattendues et économiquement attrayants par rapport au traditionnel système client/serveur, ils posent de nombreux problèmes tels que la fiabilité, la confidentialité et la disponibilité.

Dans ces systèmes, les pairs sont libres de partir ou de rejoindre le système à tout moment. En raison de la disponibilité intermittente des pairs, assurer une disponibilité élevée des données stockées est un problème intéressant et utile. Pour garantir la fiabilité et la disponibilité des données dans ces systèmes dynamiques, des données redondantes sont insérées dans le système. La redondance peut être réalisée par example par réplication, ou en utilisant des codes correcteurs d'erreurs (CCE).

Cependant, utilisert des mécanismes de redondance sans récupérer les données perdues n'est pas efficace, puisque le niveau de redondance diminue lorsque les pairs quittent le système. En conséquence, les systèmes P2P de stockage de données doivent compenser la perte de données en stockant en permanence des données redondantes supplémentaires sur de nouveaux pairs.

Dans les sections suivantes, nous allons introduire les mécanismes de redondance et de recouvrement de données, qui sont les deux principales techniques utilisés par les système P2P

de stockage et de sauvegarde des données.

C.3.1 Les mécanismes de redondance

La redondance est un mécanisme essentiel dans tous les systèmes de stockage afin d'assurer un certain niveau de fiabilité, de disponibilité et de durabilité des données. Il a d'abord été utilisé, dans le stockage des données, en 1987 dans les systèmes RAID (Redundant Arrays of Inexpensive Disks) [72]. Les systèmes RAID permettent aux ordinateurs d'atteindre des niveaux élevés de fiabilité de stockage à partir de composants peu coûteuses et peu fiables comme les disques durs des ordinateurs personnels, en disposant ces dispositifs dans des tableaux pour la redondance. La redondance permet la tolérance de panne, de sorte que tout ou partie des données stockées dans le tableau peut être récupérée en cas de défaillance des disques.

Il existe trois approches en RAID pour gérer les données stockées: (i) la réplication (mirroring) sur plus d'un disque, (ii) l'entrelacement (striping), la séparation des données à travers plus d'un disque, (iii) et l'utilisation des codes correcteurs d'erreurs (CCE) [49]. L'idée de base est de combiner deux ou plusieurs disques durs physiques en une seule unité logique, et selon la manière dont les données sont gérées (splittées, codées ou répliquées sur les disques), on peut distinguer sept niveaux de systèmes RAID.

Il existe plusieurs mécanismes disponibles pour la production des données redondantes. Toutefois, dans le cadre des systèmes P2P de stockage et de sauvegarde, nous nous concentrerons sur deux mécanismes, la réplication et le CCE.

La réplication

Il existe deux niveaux de réplication utilisés dans les systèmes P2P de stockage et de sauvegarde de données:

- *Réplication complète du fichier*. Un fichier f est répliqué r fois sur r pairs différents (comme PAST [84]) de sorte que la tolérance aux pannes ou AUX départs des pairs est égal à r. En d'autres termes, r est le nombre de pairs stockant des copies de données objet qui peuvent quitter le réseau sans perdre l'objet de données. Le rapport $1/\{\mathrm{r}+1\}$ définit l'espace de

stockage utile dans le système. Ci-après, nous ferons référence à ce niveau de réplication par “réplication”.

- *La réplication au niveau de fragments*. Ce niveau de réplication consiste à diviser un fichier f en s fragments de même taille, puis à faire r copies de chacun d'entre eux, comme dans CFS [27].

Code correcteur d'erreur CCE (Erasure coding)

Cette approche consiste à diviser le fichier f en b blocs de même taille. Chaque bloc de données est partitionné en s fragments de même taille aussi, en utilisant un code correcteur d'erreur, r fragments redondants sont ajoutés comme le montre la figure C.1. L'espace de stockage utile dans le système est défini par le rapport $s/(s+r)$. Plusieurs études [97, 8, 10]

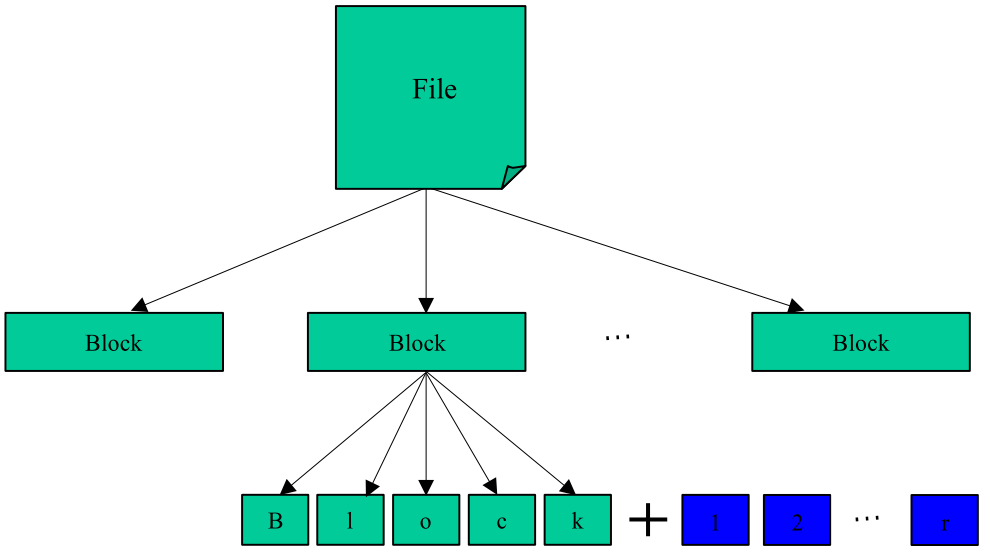


Figure C.1: L'organisation des données dans les systèmes qui utilisent CCE.

ont montré que l'approche CCE est plus efficae que les deux niveaux de réplication présentés précédement pour les systèmes de stockage car il réduit le trafic de la réplication en utilisant la puissance de calcul.

C.3.2 Les politiques et les mécanismes du processus de recouvrement

Les systèmes P2P de sauvegarde et de stockage nécessitent de compenser la perte de données, suite au départ des pairs, en stockant en permanence des données supplémentaires sur d'autres pairs redondants afin de pouvoir garentir un niveau élevé de durabilité des données et/ou de disponibilité.

En fait, les systèmes P2P de sauvegarde peuvent s'appuyer sur une autorité centrale qui reconstitue des fichiers ou des fragments quand c'est nécessaire. Ces systèmes seront dénommés *systèmes de récupération centralisés*. Alternativement, d'une façon distribuée, des agents sécurisés fonctionnant sur certains noeuds actifs peuvent reconstruire eux-mêmes les données qui seront stockées sur les disques des noeuds. Ces systèmes seront dénommés *systèmes de récupération distribués*.

Les politiques du processus de recouvrement

Indépendamment du mécanisme de recouvrement utilisé, deux politiques de réparation peuvent être appliquées: la politique *eager* et la politique *lazy*. Dans la politique **eager**, lorsque le système détecte que l'un des pairs a quitté le réseau, il lance immédiatement la reconstruction de toutes les données hébergées par les pairs qui sont en panne ou déconnectés, et les stocke sur d'autres pairs en guise de récupération. En utilisant cette politique, les données deviennent non disponibles seulement quand la vitesse de disparition des pairsest plus rapide que la détection des départs des pairs et a réparation de leurs données. Cette politique est simple, mais ne fait aucune distinction entre les départs permanents des données qui ont besoin d'être récupérées, et les déconnexions transitoires pour lesquelles le recouvrement n'est pas forcément utile.

En prenant en considération que les connexions peuvent être temporaires et pas toujours permanents, on peut retarder la réparation jusqu'à ce que le nombre de fragments indisponible d'un bloc D de données atteigne un seuil donné, noté k. Dans ce cas, on doit avoir $k \leq r$ puisque D est perdu si plus de r fragments sont manquants dans le système de stockage. Avec cette politique, le processus de recouvrement utilise moins de bande passante qu'avec la politique *eager*. Toutefois, il est évident qu'une quantité supplémentaire de redondance est nécessaire

pour tolérer les départs des pairs pour de longues périodes. Cette politique est appelée **lazy** parce que l'objectif explicite est de retarder les processus de réparation aussi longtemps que possible.

Les deux politiques de réparation peuvent être représentées par le paramètre de seuil $k \in \{1, 2, \ldots, r\}$, où k peut prendre n'importe quelle valeur dans l'ensemble $k \in \{2, \ldots, r\}$ dans la politique *lazy*, et $k = 1$ dans la politique *eager*.

Mécanisme de récupération centralisé

Prenons un bloc D de données et supposons que le système a perdu k fragments (seuil de récupération), de sorte que les fragments perdus doivent être récupérés.

Dans la mise en oeuvre centralisée, une autorité centrale: (1) télécharge *en parallèle* s fragments de D à partir des pairs actuellement disponibles, (2) reconstruit en une fois tous les fragments non disponible, et (3) retourne (upload en anglais) les fragments reconstruits *en parallèle* sur de nouveaux pairs pour le stockage. L'autorité centrale actualise la base de données enregistrant les placements des nouveaux fragments dès que la récupération est entièrement terminée. En fait, l'étape 2 s'exécute dans un temps négligeable par rapport au temps d'exécution des étapes 1 et 3. L'exécution de l'étape 1 (resp. l'étape 3) se termine lorsque le téléchargement (resp. le renvoi) du dernier fragment est terminé.

Mécanisme de récupération distribué

Dans l'implémentation distribuée, un agent sécurisé sur un nouveau pair est informé de l'identité *d'un fragment* parmi les k fragments indisponibles à reconstituer. Après la notification, l'agent (1) télécharge *en parallèle* s fragments de D à partir des pairs actuellement disponibles, (2) reconstruit le fragment spécifié et le stocke sur le disque du nouveau pair, (3) supprime par la suite les s fragments téléchargés afin de satisfaire la contrainte que seul un fragment d'un bloc de données peut être tenu par un pair. Cette opération est itérée jusqu'à ce que moins de k fragments soient indisponibles. Le temps d'exécution de l'étape 1 est considéré comme le temps du recouvrement d'un fragment dans cette implémentation distribuée. Nous allons

donc considérer que le processus de récupération se termine avec le téléchargement du dernier fragment parmis les s nécessaires.

C.3.3 Examples de systèmes P2P de stockage et de sauvegarde

"Cooperative File System" (**CFS**) [27] est un système P2P de stockage de données qui offre des garanties vérifiables pour l'efficacité, la robustesse, et la distribution équitable de la charge de stockage des fichiers. L'architecture de CFS est totalement décentralisée. En CFS, plusieurs fournisseurs de contenu coopérent pour stocker leurs données et servir chacun d'autres eux. Chaque fichier est divisé en fragments qui sont stockés sur des pairs différents. CFS a trois couches: (i) le système de fichiers (FS) qui interprète les fragments sous forme de fichiers et présente une interface de système de fichiers aux applications, (ii) le DHash (Distributed Hash), couche qui effectue la récupération et la distribution des fragments de et sur les serveurs, DHash trouve des fragments en utilisant (iii) le protocole de localisation *Chord* [89]. CFS utilise le mécanisme de réplication au niveau fragment avec la politique *eager* pour augmenter la disponibilité des données dans le système. Cependant, cela ne explore pas les compromis de coût et de résilience. DHash met les fragments redondants (d'un fragment donné) sur les r successeurs (serveurs) du pair responsable du fragment considéré dans l'anneau Chord.

TotalRecall [10] est un système de stockage P2P qui garantit un niveau prédéfini de haute disponibilité en adaptant automatiquement le niveau de redondance et la fréquence des réparations à la distribution des échecs des pairs. Il utilise une version modifiée de la *DHash [27]* pour l'emplacement des objets. Après l'estimation de la disponibilité de ses pairs, TotalRecall applique un mécanisme de *réplication* dans les environnements tres stables et un mécanisme de CCE dans les environnements à faible disponibilité. TotalRecall a été l'un des premiers systèmes qui exploitent le fait que la plupart des pairs quittent temporairement le système, et donc utilisent une politique *lazy*, contrairement au CFS par exemple.

La société française **UbiStorage** [92] a été créée au début des années 2006 et se penche actuellement sur le marché de sauvegarde en ligne pour les petites et moyennes entreprises. Elle utilise le prototype Ubiquitous Storage **US** [76, 88] qui vise à fournir un dispositif de

stockage virtuel à chaque utilisateur, et qui assure la durabilité des données. Le principal mécanisme de redondance utilisé pour assurer la durabilité des données est basé sur le code correcteur d'erreur CCE. US utilise une autorité centralisée pour contrôler le système et localiser les données. Toutefois, les efforts actuels tentent de contrôler et d'administrer le système d'une façon distribuée pour peremttre un passage à grande échèlle.

C.4 L'état de l'art et les motivations

Bien que l'état de l'art sur l'architecture des systèmes de fichiers et de sauvegarde distribuée est abondant, la plupart de ces systèmes sont configurés de façon statique pour offrir durabilité et/ou disponibilité des données avec seulement une connaissance superficielle de la façon dont la configuration aura un impact sur la performance globale. Certains systèmes permettent aux données d'être reproduites et mises en cache sans contraintes sur le coût de l'espace de stockage. Ces configurations conduisent à gaspiller la bande passante et le volume de stockage et ne fournissent pas un niveau prédéfini et clair de durabilité et de disponibilité. D'où l'importance de l'évaluation approfondie des systèmes de stockage P2P avant leur mise en service.

C.4.1 La disponibilité des pairs

Un problème majeur dans toute application P2P est que les pairs sont libres de joindre et de quitter temporairement (un temps long ou court) le système à tout moment. Certains pairs peuvent échouer à cause de problèmes matériels ou logiciels, puis ils quittent définitivement le système. Ce phénomène de quitter de de se connecter de/au réseau est nommé *churn*. En général, rejoindre le système n'a aucun impact remarquable sur le système. Toutefois, le départ et les événements d'échec ont un impact négatif important parce qu'ils peuvent causer des pertes de données.

BinzenhöFer et Leibnitz [11] ont proposé un algorithme distribué pour estimer le taux de *churn* dans les systèmes DHT (overlay structuré) en échangeant des observations de mesures entre une liste de voisins (e.g. list de successeurs dans Chord ou leafs dans Pastry).

Ramabhadran et Pasquale ont analysé, dans [75], le *All-pairs-ping* data set [90] (trace de données), qui rapporte des mesures pour le temps de disponibilité et le temps d'indisponibilité pour les noeuds de PlanetLab [74]. En traçant la fonction de distribution de chaque durée (temps de disponibilité/indisponibilité), ils ont trouvé qu'une distribution exponentielle est un ajustement "fit" raisonnable à la fois pour les durées de disponibilité et d'indisponibilité des noeuds de PlanetLab.

Caractériser la disponibilité des machines dans des environnements locaux et étendus a été l'objectif de [69]. Dans ce papier, Nurmi, Brevik et Wolski ont analysé un ensemble de trois traces de données (data set en anglais), ou chaque trace mesure la disponibilité des machines dans un contexte différent. Ils ont ajusté les distributions empiriques avec quatre distributions statistiques sur chaque trace de données et ils ont évalué aussi la qualité de leur ajustement "fit" par des outils statistiques. Ils ont trouvé que le modèle hyper-exponentiel correspond plus exactement aux durées de disponibilité des machines que l'exponentiel, Pareto, ou la distribution de Weibull. Cette étude supporte une hypothèse principale dans nos modèles.

C.4.2 La durée de vie et la disponibilité des données

Peu d'études ont développé des modèles analytiques pour les systèmes P2P de sauvegarde et de stockage dans l'objectif de comprendre, d'une part, les compromis possibles entre la disponibilité et la durée de vie des données, et d'autre part, la redondance impliquée dans le stockage des données et la fréquence de réparation. En plus, des modèles qui capturent le comportement des deux politiques du processus de recouvrement (eager et lazy), et les deux mécanismes de réplication dans la modélisation, tout en incluant, à la fois, les déconnexions temporaires et permanentes des pairs ne sont pas encore bien étudiés.

Dans [8], Bhagwan, Savage et Voelker ont proposé une analyse probabiliste de l'efficacité de la réplication au niveau de l'ensemble du fichier et au niveau des fragments, ainsi que pour l'efficacité de CCE. Ils ont étudié le coût pour maintenir un niveau donné de disponibilité à long terme, par le recouvrement des données manquantes régulièrement après chaque instant t (par exemple dix mois). Ils ont montré que l'utilisation de CCE rend le système plus évolutif que les deux niveaux de la réplication.

Cependant, cette étude ne donne que la disponibilité prévue d'un fichier quelconque stocké dans le système basée uniquement sur le facteur de réplication et sur la disponibilité des pairs. En outre, les auteurs négligent le facteur de largeur de bande et ne considèrent que les coûts de stockage.

L'objectif principal de [75] est l'analyse d'un système de stockage qui utilise la réplication

pour assurer la fiabilité des données. Cette analyse ne s'applique pas aux systèmes basés sur CCE.

Duminuco, Biersack et En-Najjary [36] ons proposé une méthode proactive pour réduire le coût de maintenance, en particulier l'utilisation de bande passante, basée sur une estimation du taux de départ des noeuds qui stockent des données.

Dans [6], Bernard et Le Fessant ont proposé une technique pour estimer la fiabilité des systèmes P2P de sauvegarde des données et optimiser leurs performances en introduisant un nouveau critère, "l'âge des pairs". Plus le pair reste connecté au système, plus on peut supposer qu'il restera en ligne. En sélectionnant soigneusement les pairs sur lesquels des données sont stockées, les coûts de la réparation peuvent être réduits de façon importante tout en assurant un niveau élevé de durabilité. Les auteurs ont décrit une méthode pour estimer l'âge des pairs et ils valident leur méthode par des simulations.

Dans [28], Dalle et al. ont développé un modèle stochastique basé sur une approximation "fluid" pour caractériser la moyenne et l'écart type de la durée de vie des données dans un système P2P de sauvegarde des données, tout en tenant compte du fait que de nombreux blocs de données sont perdus au même moment lorsqu'un pair quitte définitivement le système. Ils ne considèrent pas le taux de "churn" et n'étudient pas la disponibilité des données. Ils ont étudié un système qui ne produit jamais de fragments redondants à la suite d'une déconnexion temporaire. Un mécanisme de recouvrement est alors déclenché afin de récup'erer les fragments manquants d'un bloc de données, après un échec, si son disponibilité est inférieure à un seuil prédéfini. Le processus de récupération tend à réparer le plus vite possible tous les fragments manquants du bloc considéré de données, dès qu'un nombre suffisant de fragments sont disponibles dans le système.

C.5 Contribution de la thèse

Nous abordons dans cette thèse, la durée de vie des données et leur disponibilité dans des systèmes distribuée P2P de stockage et de sauvegarde. Dans de tels systèmes, les données ne sont plus stockées sur des bandes magnétiques très robustes, fiables et chères, mais sur les disques durs des ordinateurs (pairs) disponibles dans le réseau. Bien que peu coûteux, ces systèmes de stockage posent de nombreux problèmes de fiabilité, de confidentialité, de disponibilité, de routage, de performance, etc.

Cette thèse évalue et compare les performances de systèmes de stockage de données sur des réseaux de pairs en termes de longévité des données et de leur disponibilité. Deux mécanismes de récupération de données perdues sont considés. Le premier mécanisme est centralisé et repose sur l'utilisation d'un serveur pouvant récupérer plusieurs données à la fois alors que le second mécanisme est distribué.

Notre première contribution à l'analyse de la durée de vie des données et de leur disponibilité dans ces systèmes est [3]. Dans cette étude, des hypothèses simples ont été considérées. En particulier, nous avons supposé dans [3] que la disponibilité des machines et le processus de récupération sont exponentiellement distribués, en suivant les hypothèses et les résultats de [75, 11, 31]. Bien que les modèles soient simples, ils intègrent em même temps le comportement des deux politiques de la réparation (eager et lazey), et les deux mécanismes de recouvrement (réplication et CCE), et ils prennent en compte, les déconnexions temporaires et permanentes des pairs. Afin d'avoir des formules simples et explicites, nous avons introduit des approximations "fluid", sous des hypothèses simples, qui estiment le nombre moyen de fragments disponibles dans les systèmes P2P de stockage et de sauvegarde basés sur la réplication ou CCE.

Cependant, le processus de recouvrement dans les systèmes basés sur CCE peut différer de celui utilisé dans les systèmes répliqués. Dans les systèmes basés sur CCE, les fragments inaccessibles sont constamment récupérés, en exigeant le téléchargement en parallèle de plusieurs fragments (constituant "un bloc"). L'hypothèse que le processus de récupération suit une dis-

tribution exponentielle, faite dans des efforts récents de modélisation y compris notre première contribution, est faite surtout faute des études caractérisant la distribution “réelle” du processus de récupération. Cette thèse vise à remplir ce manque par une étude empirique. A cette fin, et pour comprendre comment le processus de récupération et le processus de téléchargement peuvent être mieux modélisés, nous avons implémenté ces processus dans le simulateur de réseau au niveau packets NS-2 [39]. Les détails d'implémentation ont été présentś dans [31]. Nous menons également plusieurs expériences couvrant une grande variété de scénarios. Nous montrons que le temps de téléchargement des fragments suit approximativement une distribution exponentielle dans le plus part des experiences faites. Nous montrons aussi que le temps de téléchargement des blocs et le temps de réparation suivent essentiellement une distribution hypo-exponentielle ayant plusieurs phases distinctes, cf. [32].

S'appuyant sur les conclusions de [32], nous avons développé dans [30] des modèles markoviens pour étudier la durée de vie des données et leur disponibilité dans des systèmes de stockage P2P en supposant que le temps de téléchargement des *fragments* suit une distribution exponentielle et que le processus de *recouvrement* suit une distribution hypo-exponentielle. Les modèles en [30] sont donc plus généraux et réalistes que ceux considérés dans [3].

Pour chaque mécanisme de recouvrement nous avons considéré que la disponibilité des machines est exponentiellement distribuée dans les modèles en [3, 30]. Cependant, dans [69], il est montré que la disponibilité des machines est mieux modélisée avec une distribution hyper-exponentielle qu'avec une distribution exponentielle, Pareto, ou de Weibull. C'est pourquoi, nous avons proposé dans [29] et [33] respectivement des modèles plus élaborés que ceux présentés dans [3] et [30] où la disponibilité des machines est hyper-exponentiellement distribuée. Nos modèles s'appliquent à différents environnements distribués. Ils permettent d'évaluer l'impact de chaque paramètre du système sur les performances. En particulier, nous montrons comment nos résultats peuvent être utilisés pour garantir la qualité de service. Les principales hypothèses faites dans nos modèles sont validées soit par des simulations au niveau paquet soit par des traces réelles recueillies dans différents environnements distribués.

Bien que notre modèle de simulation au niveau packet [31] soit capable de prédire avec

précision le comportement des processus de recouvrement et de téléchargement, tout en considérant l'impact de plusieurs contraintes telles que l'hétérogénéité des pairs et la topologie du réseau physique, le temps de simulation peut devenir excessivement long pour de très grands réseaux. Pour surmonter cette restriction de passage à l'échelle nous proposons et analysons un algorithme efficace au niveau flux, que nous avons appelé le "progressive-filling flow-level algorithm" or PFFLA. L'algorithme est simple et utilise le concept de "remplissage d'eau" (ou l'équité min-max), d'où le nom. Il permet de caractériser le temps de réponse des téléchargements en parallèle dans un système de stockage distribué. Cet algorithme a ete validé par simulations.

BIBLIOGRAPHY

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *in Proc. of the 5th Symposium on Operating Systems Design and Implementation (OSDI), USENIX*, Boston, MA, USA, December 2002. 7, 163

[2] Allmydata: Unlimited Online Backup, Storage, and Sharing. `www.allmydata.com/`. 7, 163

[3] 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. 22, 23, 89, 173, 174

[4] Azureus Wiki. `http://azureuswiki.com/index.php/DistributedTrackerAndDatabase`. 5

[5] 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. 31

[6] S. Bernard and F. Le Fessant. Optimizing peer-to-peer backup using lifetime estimations. In *Proc. of 2nd International Workshop on Data Management in Peer-to-Peer Systems (Damap'09)*, Saint-Petersburg, Russia, March 22 2009. 21, 135, 172

[7] D. Bertsekas and R. Gallager. *Data Networks, 2nd ed.* Prentice Hall, New Jersey, 1992. 111, 113, 115

[8] 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. 11, 16, 20, 165, 171

[9] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proc. of 2nd IPTPS*, Berkeley, California, February 2003. 52

[10] 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. 7, 9, 11, 12, 16, 52, 163, 165, 168

[11] A. Binzenhöfer and K. Leibnitz. Estimating churn in structured p2p networks. In *Proc. of 20th International Teletraffic Congress (ITC)*, volume 4516 of *LNCS*, pages 630–641, Ottawa, Canada, 17–21 June 2007. 18, 22, 170, 173

[12] Bittorrent. `http://www.bittorrent.com`, 2001. 5

[13] C. Blake and R. Rodrigues. High availability, scalable storage, dynamic peer networks: Pick two. In *Proc. of HotOS IX*, Lihue, Hawaii, May 2003. 21

[14] W.J. Bolosky, J.R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. *SIGMETRICS Perform. Eval. Rev.*, 28(1):34–43, 2000. 7, 163

[15] T. Bonald and A. Proutière. Insensitive bandwidth sharing in data networks. *Queueing Systems*, 44:69–100, 2003. 20, 111

[16] Jean-Yves Le Boudec. *Rate adaptation, Congestion Control and Fairness: A Tutorial*. Ecole Polytechnique Fédérale de Lausanne (EPFL), Dec 2008. 20, 110, 111, 113

[17] J.W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. *SIGCOMM Comput. Commun. Rev.*, 28(4):56–67, 1998. 9, 11, 131, 134

[18] K. Calvert, M. Doar, and E.W. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, June 1997. 93, 144

[19] G. Carofiglio, R. Gaeta, M. Garetto, P. Giaccone, E. Leonardi, and M. Sereno. A fluid-diffusive approach for modelling p2p systems. In *MASCOTS '06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pages 156–166, Washington, DC, USA, 2006. IEEE Computer Society. 135

[20] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proc. of OSDI '00*, New Orleans, Louisiana, February 1999. 15

[21] Y. Chen, J. Edler, A. Goldberg, A. Gottlieb, S. Sobti, and P. Yianilos. A prototype implementation of archival Intermemory. In *Proc. of ACM DL '99*, pages 28–37, Berkeley, California, August 1999. 7, 9, 14, 163

[22] Y. Chiu and D.Y. Eun. Minimizing file download time in stochastic peer-to-peer networks. *IEEE/ACM Trans. Netw.*, 16(2):253–266, 2008. 19, 110

[23] B.G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M.F. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association. 11, 13

[24] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of Workshop on Design Issues in Anonymity and Unobservability, Berkeley, California*, volume 2009 of *Lecture Notes in Computer Science*, pages 46–66, July 2000. 3, 7, 14, 158, 163

[25] B. Cohen. The BitTorrent protocol specification. `http://wiki.theory.org/BitTorrentSpecification`, January 2001. 3, 158

[26] Condor: High throughput computing. `http://www.cs.wisc.edu/condor/`, 2007. 23, 52

[27] 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. 6, 7, 9, 10, 12, 13, 15, 16, 161, 163, 165, 168

[28] 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. 21, 89, 172

[29] A. Dandoush, S. Alouf, and P. Nain. P2p storage systems modeling, analysis and evaluation. Technical Report RR-6392, INRIA Sophia Antipolis, December 2007. 23, 89, 174

[30] 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. 23, 174

[31] 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. 22, 23, 29, 65, 118, 173, 174

[32] 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. 23, 29, 30, 65, 174

[33] 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. 23, 174

[34] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J. Royal Statist. Soc.*, 39(1):1–37, 1977. 97, 98

[35] 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. 9, 22, 27, 129, 131, 134

[36] A. Duminuco, E. Biersack, and T. En-Najjary. Proactive replication in distributed storage systems using machine availability estimation. In *Proc. of the 2007 ACM CoNEXT conference*, pages 1–12, New York, NY, USA, 2007. ACM. 21, 172

[37] K. Eger, T. Hobfeld, A. Binzenhofer, and G. Kunzmann. Efficient simulation of large-scale P2P networks: Packet-level vs. flow-level simulations. In *Proc. of UPGRADE-CN'07*, Monterey, California, June 2007. 137

[38] eMule. `http://www.emule-project.net`, 2002. 3, 158

[39] K. Fall and K. Varadhan. The NS manual, the VINT project, UC Berkeley, LBL, USC/ISI, and Xerox PARC. `http://www.isi.edu/nsnam/ns/ns-documentation.html`, November 2008. 22, 137, 174

[40] N. Feamster, L. Gao, and J. Rexford. How to lease the internet in your spare time. *ACM SIGCOMM Computer Communication Review*, 37:61–64, 2007. 101

[41] A.A. Fisk. Dynamic Query Protocol. `http://www.ic.unicamp.br/~celio/peer2peer/gnutella-related/gnutella-dynamic-protocol.htm`, 2001. 3, 4, 10, 158, 159

[42] U.C. Frank, B. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *In Proc. of NSDI*, pages 45–58, 2006. 16

[43] S. Ben Fredj, T. Bonald, A. Proutiere, G. Régnié, and J.W. Roberts. Statistical bandwidth sharing: a study of congestion at flow level. *SIGCOMM Comput. Commun. Rev.*, 31(4):111–122, 2001. 19, 110

[44] GÉANT: a pan-European backbone which connects Europe's national research and education networks. `http://www.geant.net/server/show/nav.159`. 95

[45] A.V. Goldberg and P.N. Yianilos. Towards an archival Intermemory. In *Proc. of ADL '98*, pages 147–156, Santa Barbara, California, April 1998. 7, 9, 14, 163

[46] C. Grinstead and J. Laurie Snell. *Introduction to Probability*. American Mathematical Society, 1997. 33, 39, 48, 68, 73, 81

[47] 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. 30, 54, 92, 95, 96, 97, 116, 145

[48] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: highly durable, decentralized storage despite massive correlated failures. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 143–158, Berkeley, CA, USA, 2005. USENIX Association. 12, 16

[49] R.W. Hamming. Error detectmg and correcting codes. *Bell System Technical Journal*, XXVI(2):147–160, April 1950. 8, 164

[50] 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. 26, 30, 36, 64, 65, 70, 89, 97

[51] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. ns-3 project goals. In *WNS2 '06: Proceeding from the 2006 workshop on ns-2: the IP network simulator*, page 13, New York, NY, USA, 2006. ACM. 134

[52] D.P. Heyman, T.V. Lakshman, and A.L. Neidhardt. A new method for analysing feedback-based protocols with applications to engineering web traffic over the internet. In *SIGMETRICS '97: Proceedings of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 24–38, New York, NY, USA, 1997. ACM. 19, 110

[53] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Leveraging Bittorrent for end host measurements. In *Proc. 8th Passive and Active Measurement Conference*, Louvain-la-neuve, Belgium, April 2007. 95

[54] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, New York, NY, USA, 1997. ACM. 6, 91

[55] Kazaa media desktop. `http://www.kazaa.com`, 2001. 3, 4, 158, 159

[56] L. Kleinrock. *Queueing Systems, Vol. 1*. J. Wiley, New York, 1975. 34

[57] L. Kleinrock. *Queueing Systems, Vol. 2*. J. Wiley, New York, 1975. 119

[58] 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. 31

[59] 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. 7, 9, 11, 15, 163

[60] T.G. Kurtz. Solutions of ordinary differential equations as limits of pure jump markov processes. *Journal of Applied Probability*, 7(1):49–58, April 1970. 35, 69

[61] Mathieu Lacage. Ns-3 unmodified posix api. http://code.nsnam.org/mathieu/ns-3-simu/. 134

[62] F.J. Massey. The kolmogorov-smirnov test for goodness of fit. *J. Am. Statist. Assoc.*, 46(253):68–78, 1951. 96

[63] L. Massoulié and J.W. Roberts. Bandwidth sharing and admission control for elastic traffic. *Telecommunication Systems*, 15(1-2):185–201, 2000. 19, 110

[64] J. McCaleb. eDonkey2000. `http://www.eDonkey.com/`, 2000. 10

[65] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: Boston University representative Internet topology generator. `http://www.cs.bu.edu/brite/`. 93

[66] Napster. `http://www.napster.com`, 1999. 3, 5, 10, 158, 160

[67] The network simulator project, UC Berkeley, LBL, USC/ISI, and Xerox PARC. `http://nsnam.isi.edu/nsnam/index.php/Main_Page`. 87, 132

[68] M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models. An Algorithmic Approach*. John Hopkins University Press, Baltimore, 1981. 33, 38, 46, 47, 67, 72, 73, 80

[69] 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. 19, 23, 26, 29, 51, 52, 53, 54, 59, 64, 82, 171, 174

[70] S. Oechsner and P. Tran-Gia. Performance evaluation of a reliable content mediation platform. In *Proc. of 20th International Teletraffic Congress (ITC)*, volume 4516 of *LNCS*, pages 154–165, Ottawa, Canada, 17–21 June 2007. 135

[71] M. Olsson. The EMpht-programme. Technical report, Department of Mathematics, Chalmers University of Technology, 1998. 97, 98

[72] D.A. Patterson, G. Gibson, and R.H. Katz. A case for redundant arrays of inexpensive disks (raid). In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 109–116, New York, NY, USA, 1988. ACM. 8, 164

[73] L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. In *Proc. of the 3rd ACM Workshop on Hot Topics in Networks (HotNets-III*, 2004. 101

[74] PlanetLab. An open platform for developing, deploying, and accessing planetary-scale services. `http://www.planet-lab.org/`, 2007. 18, 23, 52, 90, 170

[75] S. Ramabhadran and J. Pasquale. Analysis of long-running replicated systems. In *Proc. of IEEE Infocom*, Barcelona, Spain, April 2006. 18, 19, 21, 22, 26, 29, 52, 53, 64, 82, 88, 89, 96, 170, 171, 173

[76] C. Randriamaro, O. Soyez, G. Utard, and F. Wlazinski. Data distribution in a peer to peer storage system. *Journal of Grid Computing*, 4(3):311–321, September 2006. 17, 168

[77] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM. 6

[78] I.S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of SIAM*, 8(2):300–304, June 1960. 9, 27, 130, 131, 134

[79] Renater: Le Réseau National de télécommunications pour la Technologie, l'Enseignement et la Recherche. `http://www.renater.fr`. 95

[80] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. In *Proc. of the USENIX Annual Technical Conference*, Boston, MA, June 27–July 2 2004. Awarded best paper. 18

[81] R. Rodrigues and B. Liskov. High availability in dhts: Erasure coding vs. replication. In *Peer-to-Peer Systems IV 4th International Workshop IPTPS 2005*, Ithaca, New York, USA, February 2005. 21

[82] J. Rothenberg. Ensuring the longevity of digital documents. *Scientific American*, pages 642–47, 1995. 14

[83] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *In Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 2218 of *LNCS*, pages 329–350, Heidelberg, Germany, November 2001. 5, 6, 91, 160, 161

[84] 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. 6, 7, 9, 10, 16, 161, 163, 164

[85] 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. 30, 54

[86] Fips 180-1. secure hash standard. u.s. department of commerce nist, national technical information service, springfield, va. `http://www.itl.nist.gov/fipspubs/fip180-1.htm`, 1993. 6, 161

[87] RFC 3174 - US Secure Hash Algorithm 1 (SHA1). `http://www.faqs.org/rfcs/rfc3174.html`. 6, 14, 161

[88] O. Soyez. *Stockage dans les systèmes pair à pair*. PhD thesis, Jules Verne University of Picardie, France, November 2005. 17, 168

[89] 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. 5, 6, 15, 91, 160, 161, 168

[90] J. Stribling. PlanetLab - All Pairs Pings. `http://pdos.csail.mit.edu/~strib/pl_app`, 2005. 18, 52, 170

[91] J. Turner and D. Taylor. Diversifying the internet. In *In Proc. IEEE GLOBECOM*, pages 755–760, 2005. 101

[92] UbiStorage. `http://http://www.ubistorage.com`. 6, 9, 13, 17, 161, 168

[93] G. Utard and A. Vernois. Data durability in peer to peer storage systems. In *Proc. of IEEE/ACM CCGRID 2004 (GP2PC 2004)*, pages 90–94, Chicago, Illinois, April 2004. 11, 16, 21

[94] E. Varki. Response time analysis of parallel computer and storage systems. *IEEE Trans. Parallel Distrib. Syst.*, 12(11):1146–1161, 2001. 19, 110

[95] B. Warner, Z. Wilcox-O'Hearn, and R. Kinninmont. Tahoe: A Secure Distributed Filesystem. `http://allmydata.org/~warner/pycon-tahoe.html`. 17, 134

[96] B.M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988. 93

[97] 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. 11, 165

[98] R.W. Wolff. *Stochastic Modeling and the Theory of Queues*. Prentice-Hall, 1989. 40

[99] Wula: Secure Online Storage. `http://www.wuala.com/`. 7, 163

[100] S. F. Yashkov and A. S. Yashkova. Processor sharing: A survey of the mathematical theory. *Automation and Remote Control*, 2007. 153

[101] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, 2004. 5, 14, 160