A Survey of Cooperative Backup Mechanisms

Marc-Olivier Killijian, Ludovic Courtès, David Powell

A Survey of Cooperative Backup Mechanisms [*]

Marc-Olivier Killijian [*], Ludovic Courtès, David Powell

Universités de Toulouse, Laboratoire d'Analyse et d'Architecture des Systèmes, Toulouse

Abstract

Storage capacity, like computing power, follows its Moore's law and grows dramatically. Consequently, the need for data backup services increases. The domain of data backup was recently hit by the peer-to-peer wave: several projects propose to use a cooperative approach to the backup problem. In this paper we survey these projects, the various problems they face and the techniques they propose to tackle them.

Key words: Ad-hoc, Peer-to-peer, Cooperative systems, Data backup

1 Introduction and Motivations

Storage capacity, like computing power, follows its Moore's law and grows dramatically, for instance disk density grows at an impressive annual rate of 100% [1]. At the same time, this new storage capacity is consumed by the production of new data. Consequently, the need for backup capacity increases but so does the space available for backup.

Our concern here is on *cooperative* backup services, in which resources belonging to multiple users are pooled as a distributed storage system for backing-up

	File type	Writer multiplicity	Reader multiplicity
File sharing systems	Static	Single	Multiple
File backup systems	Static	Single	Single
General file systems	Dynamic	Multiple	Multiple

Fig. 1. Typical characteristics distinguishing various distributed storage

each others' data. Such a cooperative backup service must be distinguished from other forms of distributed storage such as file sharing systems or general file systems.

A file system can be defined as a support for the storage of data on non-volatile medium, typically a hard disk. Data is stored on files that encompass both the data and its associated metadata (name and other attributes such as date of modification, etc.). Usually a file system also provides a directory service on top of a flat file service. The flat file service maps the data with unique file identifiers and stores the data on the storage device. The organization of the data can have several forms, either unstructured or structured as a sequence or hierarchy of records. The directory service maps the metadata to the files' unique identifiers. Typically this mapping is stored on the storage device using the flat-file service itself. There is a number of distributed file systems, the most famous being NFS (Network File System) that is used on many UNIX networks.

File sharing [1] emerged relatively recently as an Internet application and greatly participated to the definition of a new type of distributed system: peer-to-peer systems. The goal of a file sharing system is to enable multiple users to access files. Classical and well-known file-sharing systems are Napster, eDonkey, Gnutella, Kazaa, MojoNation, BitTorrent, etc.

The role of a backup system is to tolerate faults affecting some storage device, be it local or distant, centralized or shared. The type of faults considered here can be permanent failures of the storage devices (e.g., crashed disks), or even localized catastrophes like a fire incident in an office when the backup media are taken off-site.

Given these definitions, one can see that there are quite some differences in the specification of these services even if there are also some strong similarities (primary goal is storage, the concept of file, etc.). If one wants more specific differences, one can consider the following properties: multiplicity of the data readers/writers and mutability of the content. The following table presents these characteristics.

[1] We differentiate here between peer-to-peer file sharing systems and distributed file systems that can also be seen as a way to share files.

Among the afore-mentioned file-sharing systems, we can identify some features that are centralized, distributed or cooperative. Similarly to file systems, file-sharing systems have two main functions: first they have to manage the actual distribution of the shared files, and second they have to organize the lookup, i.e. manage a global directory for users to search for given files. The lookup service of Napster [2] is centralized, the one of eDonkey [3] is distributed among a set of servers and finally, the one of Kademlia [4] is cooperatively realized between the participating nodes.

It is clear that file-sharing systems are different from backup systems. These systems do not guarantee long-term survivability of files, especially those files that few users are interested in storing or accessing. Thus, they could hardly be used for the purpose of backup. One could argue that regular file systems could easily be used for such a purpose since long-term survivability and fault-tolerance are very important concerns for this type of service. For instance, a simple solution to back-up on top of file systems would be to use Unix-like facilities, e.g., tar, CVS, etc. However, the specification and the semantics of file systems being so much broader than those of backup systems (multi-writers vs. single-writers; read-write vs. write-once/read-many), it would be unfair to compare these two types of system.

In this paper we will survey only backup services that use a cooperative approach. We will be concerned both with cooperation between resources pooled directly over a fixed infrastructure such as Internet and with mobile resources that are pooled opportunistically according to locality. This latter class of co-operative backup service is motivated by the observation that users of mobile devices (laptops, personal digital assistants, mobile phones, etc.) often perform a backup of their data only when they have access to their main desktop machine, by synchronizing the two machines. Typically, the first generation of personal digital assistants (PDAs) had only a short distance communication means, generally a serial or infrared device. This meant that the user had to be physically close to the machine on which she performed the synchronization. Nowadays, portable devices usually have several communication interfaces (for instance WiFi, Bluetooth, etc.). When a network infrastructure is available in their vicinity, for instance WiFi access points, those devices could connect to their main desktop machine in order to back-up their data. However, in practice, this is rarely the case, for several reasons :

- the desktop machine must be running, connected to the Internet and available;
- access to a network infrastructure using wireless communications is still rare and expensive, and it can take a while before a device is able to connect to the Internet;
- finally, to our knowledge, the software able to perform such a backup on a remote desktop are still rare.

Another solution to mobile device backup is the use of a trusted third party that guarantees its backup servers' availability. Several commercial offerings enable their customers to back-up their data on a limited storage capacity for a yearly fee.

However, the growth rate of this kind of wireless communicating devices is such that a cooperative approach to back-up is becoming feasible, based on peer-to-peer interactions. These wireless interactions are frequent but ephemeral. Nevertheless, they could be leveraged: whenever two devices meet, a backup service can automatically initiate a request for a partial data backup. As a counterpart, it has to offer the same service to the community, i.e. to form a *cooperative* backup system. In Section 2 of this survey, we discuss the features that characterize cooperative backup systems. Several existing systems are then described and compared briefly in Section 3. In Section 4, a more in-depth analysis is given with respect to storage management issues. Then Section 5 focuses on the dependability techniques used in these systems. Finally, Section 6 concludes the survey by a summary and sketches some directions for future work.

2 Characterization of Cooperative Backup Systems

In [5], Chervenak *et al.* described a number of features for the characterization of backup systems: full vs. incremental, file-based vs. device-based, online or not, snapshots and copy-on-write, concurrent backups, compression, file restoration, tape management, and finally disaster recovery. On the one hand, most of these features remain of interest in our context. However, on the other hand, some characteristics concerning dependability and the cooperative nature of the considered systems were not addressed: privacy, denial-of-service resilience, trustworthiness management, etc. In their survey, Chervenak et al. characterized backup systems using a set of properties. It must be noted that their focus was on centralized or server-based, system-wide, backup systems, i.e., the target was large multi-user client systems. The type of backup services we are interested in targets personal computers and thus, some of the properties defined by Chervenak et al. are not relevant. We thus propose another characterization of backup systems based on a set of functionalities and of dependability issues, as described in the following sections.

4

2.1 Functionalities of Backup Systems

2.1.1 Full vs. Incremental Backups

The simplest solution to back-up a file-system is to copy its entire content to a backup device. The resulting archive is called a full backup or a snapshot of the source data. Both a full file-system and a single lost file can be easily restored from such a full backup. However, this solution has usually two major drawbacks: since it concerns the entire content of the file-system, it is slow and requires a large amount of backup storage space. We will come back onto this issue of resource usage in the next subsection.

As a solution to this, incremental backup schemes can be used. They copy only the data that have been created (added) or modified since a previous backup. To restore the latest revision of a given file, the first full backup along with all the subsequent incremental backups must be read, which can be extremely slow. For this reason, typical incremental schemes perform occasional full backups supplemented with frequent incremental backups. Several variations of incremental backup techniques exist: incremental-only, full+incremental or even continuous incremental where newly created or modified data is backed-up within a few minutes, as it is created or modified, instead of once a day, typically, with traditional incremental backups.

2.1.2 Resource Usage

To reduce both storage requirements and network bandwidth usage, backup systems can use classic compression techniques. This can be done at the client-side or at the server-side. Recently, other techniques emerged to reduce the storage space required to back-up several file systems. An example is single-instance storage [6] which aims to store once every block of data even if it is present on the file systems of several users, or if there are multiple instances in a single file-system.

2.1.3 Performance

Backup system performance is measured in terms of the backup time as well as the restoration time. The performance of the backup process is impacted by factors such as incremental backups, compression, etc. Several parameters and features have a dramatic effect on the actual efficiency of the restoration operations. For instance, restoration will be slower in an incremental backup system, which must begin with the last full backup and apply changes from subsequent incremental backups. An additional concern when restoring an entire file system is that files deleted since the previous backup will reappear

in the restored file system. More generally, the unbacked-up changes on the metadata, the structure and the hierarchy of the file system cannot be restored.

It is important to note that scalability is a very important issue when dealing with cooperative systems. The number of nodes participating in the cooperative system can be potentially very large and this raises a number of issues and problems to be dealt with. An important metric for cooperative backup system is thus the number of nodes that the system can accomodate.

2.1.4   On-line Backups

While many backup systems require that the file system (or the files) remain quiescent during a backup operation, on-line or active backup systems allow users to continue accessing files during backup. On-line backup systems offer higher availability at the price of introducing consistency problems.

The most serious problems occur when directories are moved during a backup operation, changing the file system hierarchy. Other problems include file transformations, deletions and modifications during backup. In essence, any type of write operation on the files or on the file-system hierarchy during a backup is a potential source of problems. There are several possible strategies to overcome these problems:

(1) *Locking* limits the availability of the system by forbidding write accesses while backing-up.
(2) *Modification detection* is used to reschedule a backup of the modified structures/files.
(3) *Snapshots*, i.e., frozen, read-only copies of the current state of the file-system offer another alternative for online backup. The contents of a snapshot may be backed-up without danger of the file system being modified from subsequent accesses. The system can maintain a number of snapshots, providing access to earlier versions of files and directories.
(4) A *copy-on-write* scheme is often used along with snapshots. Once a snapshot has been created, any subsequent modifications to files or directories are applied to newly created copies of the original data. Blocks or file segments are copied only if they are modified, which conserves disk space.

2.2   Dependability and Other Orthogonal Issues of Cooperative Backup Systems

In the previous section we presented several functional aspects of backup systems. We now look at the orthogonal issues raised by a cooperative approach to backup: integrity and consistency, confidentiality and privacy, availability,

synergy and trust.

2.2.1 Integrity and Consistency

A backup service has to guarantee the integrity and consistency of restored data.

Any corruption of the backed up data, be it intentional or not (for instance due to a software or hardware fault on the system actually providing the storage), must be detected by its owner during restoration. Network protocols as well as storage devices commonly use error-detecting or correcting codes to tolerate software and hardware faults. However, to be resilient to intentional corruption, the data owner must be assured that the data restored is the same that which was backed up.

Consistency is an issue when multiple items of data must ensure some common semantics. In such cases, special care must be taken to manage dependencies.

2.2.2 Confidentiality and Privacy

The entities participating in a cooperative backup service store some of their data on the resources of other participants with whom they have no a priori trust relationship. The data backed up may be private and thus should not be readable by any participating entity other than its owner, i.e., the service has to ensure the confidentiality of the data. Furthermore, a cooperative backup service must protect the privacy of its users. For instance it should not deliver any information concerning the past or present location of its users.

2.2.3 Availability

In a backup system, availability has several dimensions. First, the primary goal of a backup system is to guarantee the long-term availability of the data being backed up. In some sense it is the functional objective of the system. Second, to be useful, the backup system itself must be available, i.e., it must be resilient to failures (hardware, software, interaction, etc.). In the context of a cooperative approach to backup, additional concerns arise, especially with respect to malicious or selfish denial-of-service attacks.

2.2.4 Synergy

Synergy is the desired positive effect of cooperation, i.e., that the accrued benefits are greater than the sum of the benefits that could be achieved with-

out cooperation. However synergy can only be achieved if nodes do indeed cooperate rather than pursuing some individual short-term strategy, i.e. being rationale.

Hardin introduced the tragedy of the commons concept in 1968 [7] to formalize the fact that a shared resource (a common) is prone to exhaustion if the resource consumers use short-term strategy to maximize their benefit out of the resource. Consider the simple example of a grass field shared by 25 farmers. The field can normally accommodate 50 cattle. However, each rational farmer is tempted to maximize his outcome by having more than 2 cattle feeding from the shared field. This short-sighted strategy eventually leads the field to exhaustion through over-consumption. A generalized form of the problem is when a resource market has externalities, i.e., when the cost of using a resource is shared among its consumers.

The tragedy of the commons has recently been extended to the digital world, or "Infosphere", leading to the tragedy of the digital commons [8]. It is relatively intuitive, for instance, to regard the Internet as a shared resource. Each user uses his connection without paying much attention to the presence of other users and to the fact that they share a common bandwidth. Each user thus uses his available bandwidth up to its maximum, only being reminded that other users also consume this resource when there is a network congestion.

One way to ensure synergy in a cooperative backup system is to enforce the "fair exchange" property: if one contributes up to 5 MiB to the system, one wants to get serviced up to 5 MiB too. Reciprocally, it is desirable that a device getting serviced for such an amount of resources offers an equivalent amount to the cooperative service.

2.2.5   Trust Management

An important aspect of many cooperative systems is that each node has to interact with unknown nodes with which it does not have a pre-existing trust relationships. The implementation of a cooperative backup service between nodes with no prior trust relationship is far from trivial since new threats must be considered:

(1) selfish devices may refuse to cooperate;
(2) backup repository devices may themselves fail or attack the confidentiality or integrity of the backup data;
(3) rogue devices may seek to deny service to peer devices by flooding them with fake backup requests; etc.

There is thus a need for trust management mechanisms to support cooperative services between mutually suspicious devices.

3 Existing Cooperative Backup Systems

In this section, we first give a preliminary description and analysis of various systems devoted to cooperative backup. Cooperative backup are inspired by both cooperative file systems and file sharing systems. Most are concerned with the problem of cooperative backup for fixed nodes with a permanent Internet connection. To our knowledge, there are only two projects looking at backup for portable devices with only intermittent access to the Internet: *FlashBack* [9] and *MoSAIC* [10].

3.1 Peer-to-peer Backup Systems for WANs/LANs

The earliest work describing a backup system between peers is the one of Elnikety *et al.* [11], which we will henceforth refer to as *CBS*. Regarding the functions of a backup system (resource localization, data redundancy, data restoration), this system is quite simple. First, a centralized server is used to find partners. Second, incremental backup, resource preservation, performance optimization were not addressed. However, various types of attacks against the system are described. We will come back to this later.

The *Pastiche* [12] system and its follow-up *Samsara* [13], are more complete. The resource discovery, storage, data localization mechanisms that are proposed are totally decentralized. Each newcomer chooses a set of partners based on various criteria, such as communication latency, and then deals directly with them. There are mechanisms to minimize the amount of data exchange during subsequent backups. Samsara also tries to deal with the fair exchange problem and to be resilient to denial-of-service attacks.

Other projects try to solve some limitations of the *Pastiche/Samsara* systems, or to propose some simpler alternatives. This is the case for *Venti-DHash* [14] for instance, based on the *Venti* archival system [6] of the *Plan 9* operating system. Whereas *Pastiche* selects at startup a limited set of partners, *Venti-DHash* uses a completely distributed storage among all the participants, as in a peer-to-peer file sharing system.

PeerStore [15] uses a hybrid approach to data localization and storage where each participant deals in priority with a selection of partners (like *Pastiche*). Additionally, it is able to perform incremental backup for only new or recently modified data. Finally, *pStore* [16] and ABS [17], which are inspired by versioning systems, propose a better resource usage.

Based on the observations that worms, viruses and the like can only attack machines running a given set of programs, the *Phoenix* system [18] focuses on

techniques favoring diversity among software installations when backing up a machine (e.g., trying to not backup a machine that runs a given vulnerable web server on a machine that runs the same web server). The main added value is here in the partnership selection.

In [19], the authors focus on the specific issue of resource allocation in a cooperative backup system through an auction mechanism called bid trading. A local site wishing to make a backup announces how much remote space is needed, and accepts bids for how much of its own space the local site must "pay" to acquire that remote space.

In [20], the authors implement a distributed backup system, called *DIBS*, for local area networks where nodes are assumed to be trusted: the system ensures only privacy of the backed up data but does not consider malicious attacks against the service. Since *DIBS* targets LANs, all the participating nodes are known *a priori*, partnerships do not evolve, and no trust management is needed.

3.2 Cooperative File Systems

As mentioned earlier, a backup system (static data files, single writer) can be implemented on top of any file system (mutable data files, multi-writer). There exist a number of peer-to-peer general file systems such as Ivy [21], OceanStore [22], InterMemory [23], Us [24], etc. We briefly present here two of them for the sake of the comparison although they are outside the scope of this survey.

Us [24] provides a virtual hard drive: using a peer-to-peer architecture, it offers a read-only data block storage interface. On top of *Us*, *UsFs* builds a virtual file system interface able to provide a cooperative backup service. However, as a full-blown filesystem, *UsFs* provides more facilities than a simple backup service. In particular, it must manage concurrent write access, which is much more difficult to implement in an efficient way.

OceanStore [22] is a large project where data is stored on a set of untrusted cooperative servers which are supposed to have a long survival time and high speed connection. In this sense we consider it as a distributed file system using a super-peers approach rather than a purely cooperative system. The notion of super-peers relates to the fact that peers are specifically configured as file servers (with large amount of storage) that can cooperate to provide a resilient service to non-peer clients.

The *FlashBack* [9] cooperative backup system targets the backup of mobile devices in a Personal Area Network (PAN). The nature of a PAN simplifies several issues. First, the partnerships can be defined statically as the membership in the network changes rarely: the devices taking part in the network are those that the users wear or carry. Second, all the devices participating in the cooperative backup know each other. They can be initialized altogether at configuration time so there is no problem of handling dynamic trust between them. For instance, they may share a cryptographic key.

MoSAIC [10] is a cooperative backup system for communicating mobile devices. Mobility introduces a number of challenges to the cooperative backup problem. In the context of mobile devices interacting spontaneously, connections are by definition short-lived, unpredictable, and very variable in bandwidth and reliability. Worse than that, a pair of peers may spontaneously encounter and start exchanging data at one point in time and then never meet again. Unlike *FlashBack*, the service has to be functional even in the presence of mutually suspicious device users.

4   Storage Management

In this section, we present two aspects that are specific to peer-to-peer data storage systems: mechanisms for storage allocation, and techniques for efficient usage of resources.

4.1   Storage Allocation

Among the systems studied, one can identify three distinct approaches to the dissemination of the data blocks to be stored:

- the storage can be allocated to specific sets of participants or partners;
- the storage can be allocated across all participants using a distributed hash table (DHT), which has the property of ensuring an homogeneous block distribution;
- the storage can be allocated opportunistically among neighbors met when storage of a block is needed.

In the first case, the relationships between the partners are relatively simple: each participant chooses a set of partners at start-up. Then, for each backup, it directly sends the blocks to be saved to its partners. In Pastiche and in

CBS, each participant chooses a set of partners that will remain almost static. Finally, the FlashBack devices, in a PAN, choose their set of partners according to the amount of time spent in each other's vicinity.

The second approach is based on a technique that is fundamental to peer-to-peer file sharing systems, *virtual networks* or *overlay networks* [25], which use the notion of distributed hash tables (DHT) for allocating data blocks. Each node of the network is responsible for the storage of the blocks whose identifier is close (numerically) to its own identifier. The advantage of using a DHT is that the blocks are homogeneously distributed over the network if their identifiers are numerically homogeneously distributed. Both Venti-DHash and pStore use DHTs to store backup data blocks. However, there are two disadvantages to this approach:

- The cost of migration of the data blocks when a node enters or leaves the system can be high (bandwidth-wise) [15]. Because of the mathematical mapping between data blocks and node identifiers, no exception is acceptable: when a node enters the virtual network, it must obtain and store all the blocks for which it is mathematically responsible; respectively, when a node leaves the network, the various blocks it was responsible for must be re-distributed using the DHT mechanism.
- A DHT automatically distributes the data blocks homogeneously among the participants, independently of how much storage space each node consumes. Consequently, using a DHT makes it impossible for a system to ensure fair exchange.

For these reasons, PeerStore proposes a hybrid approach where the data blocks are directly exchanged between partners and where the blocks' meta-information (the mapping between a block ID and the node that stores it) are stored using a DHT. For optimization, the set of partners is sometimes extended at runtime to nodes that were not originally in the partnership: when a node needs to store a block, it looks into the DHT to see if the block is already stored (single-instance storage). When that is the case, the block is not stored twice. Instead, the node that already stores it becomes a new partner for the node owning it.

The third approach is very different. The MoSAIC system targets mobile devices, so partnerships cannot be established a priori [2], but have to be defined during the backup itself, opportunistically. MoSAIC is an active backup system - whenever some critical data is modified, the modified blocks need to be backed-up. This is done towards the devices that the user will meet along its way. In this case, the partnership is determined at runtime and is a function

---

[2] There may be exceptions to this in some application scenarios where mobility patterns are known in advance. For instance, when two mobile device users take the same train every single morning while commuting.

of the mobility patterns of the participating nodes.

4.2   Storage Optimization

The amount of storage necessary to store backed-up data can be optimized by applying compression techniques. Compression is worthwhile even if data is ultimately backed-up in redundant copies (to ensure backup availability). Indeed, the redundancy that is eliminated using compression techniques can be seen as "accidental", e.g., due to overly prolix data formats. Thus, compression can be thought of as a way to *normalize data entropy* before adding new redundancy. In other words, going through the compression step before adding redundancy is a means to achieve *controlled redundancy*. In particular, controlled redundancy means that the backup software is able to control the distribution of redundant data.

Backup systems often rely on "traditional" stream compression techniques, such as *gzip* and similar tools. Additionally, most backup systems have focused on techniques allowing for storage and bandwidth savings when only part of the data of interest has been modified, i.e., incremental backup techniques. Of course, similar techniques are used by revision control systems [26] or network file systems [27].

Incremental backup has the inherent property of reducing storage (and bandwidth) usage because only changes need to be sent to cooperating peers and stored. However, snapshot-based systems can be implemented such that they provide storage and bandwidth efficiency comparable to that of incremental backup systems, while still allowing for constant-time restoration. Namely, *single-instance storage* is a technique that has been used to provide these benefits to a number of backup [12,15,28], archival [6,29] and revision control systems [26], as well as distributed file systems [27,30].

Single-instance storage consists in storing only once any given data unit. Thus, it can be thought of as a form of compression among several data units. In a file system, where the "data unit" is the file, this means that any given content, even when available under several file names, will be stored only once. The single-instance property may also be provided at a finer-grain level, thus allowing for improved compression.

The authors of Pastiche and PeerStore argue that single-instance storage can even be beneficial at the scale of the aggregated store made of each contributor store. In essence, they assume that a lot of data is common to several participants, and thus argue that enforcing single-instance of this data at a global scale can significantly improve storage efficiency.

While common data may easily be found among participants in the context of Pastiche and PeerStore, where each participant is expected to back up their whole disk (i.e., including application binaries and data), this is certainly not the general case. For example, the mobile users of MoSAIC are expected to explicitly pay attention to their personal, critical data which are unlikely to be shared among several participants. Consequently, single-instance storage may be beneficial to mobile users only when used at a local scale, i.e., on each data owner's local store.

5 Dependability Techniques

We study, in this section, the various techniques found in the literature to address the dependability issues presented in section 2.2.

5.1 Integrity and Consistency

Integrity and consistency are two properties that are usually obtained using some kind of data encoding. Apart from CBS, every system studied here systematically fragments the backup files. This is necessary for load-balancing: with small sized fragments, it is easier to adapt the placement to fairly balance the load imposed on the participants.

pStore uses simple data structures to encode the backup files. The files are fragmented in varying size blocks. Along with the blocks themselves, each node also stores a list of blocks that contains, for each version of the considered file, the list of the identifiers of its constituent blocks. Each block list is indexed with a structure containing a cryptographic hash of the file path and the key of its owner. There is thus one namespace per user. In practice, for the restoration of a given file, one needs to know the file path and the key of the owner. Without this metainformation, one cannot access the file's block list and consequently its blocks. The same technique is used in PeerStore, and a similar technique in Pastiche.

In ABS, each fragment is stored along with a block of meta-information about the file from which the block originates, as well as the position of the fragment within the file. These data (fragment and metainformation) are indexed using a cryptographic hash of the fragment to implement single instance storage of each fragment. The metainformation is encrypted using the owner's public key and the set (fragment and metainformation) with the owner's private key. These signatures are used to certify ownership and for ensuring integrity of the blocks.

In a similar manner, Venti-DHash encoding is based on Venti. As with a classical file system, the files are represented as trees whose leaves are the file fragments. Here, all the blocks are indexed using their digest. They are fixed-sized and the underlying storage middleware is not aware of their semantics (leaf nodes, intermediary nodes, data, metadata, etc.). To be able to restore a file, only the knowledge of the identifier of the root node is necessary.

All these techniques provide some guarantee of data block integrity since block addressing is realized using an identifier that depends on the content of the block (using a digest). When a block is restored, one can then check whether or not it is the requested block and if it is correct. If the metainformation concerning a file is stored using the same technique, integrity is thus also guaranteed file-wise. However, from the user point of view, several files can have common semantics and thus should form a consistency unit. Only Pastiche guarantees inter-file consistency. Since it is implemented as a file system, Pastiche can create shadow copies of the blocks being backed-up, so that they can be modified during the backup process without compromising their consistency.

5.2   Confidentiality and Privacy

Most of the systems studied here, like many file sharing systems, use *convergent encryption* [30] to provide some confidentiality despite untrustworthy partner nodes. The objective is to have an encryption mechanism that does not depend on the node performing the encryption, i.e., that is compatible with single-instance-storage. Convergent encryption is a symmetric encryption technique whose key is a digest of the block to be ciphered. The ciphered block can then be stored on the untrustworthy partner nodes. A digest of the ciphered block is commonly used as an identifier of that block. The tuple of digests (ciphered/unciphered block digests) is called digest-key or CHK for "content hash key". The data owner needs the CHK to be able to locate and uncipher a block. The CHKs are themselves backed up and ultimately the data owner only has to "remember" one CHK. Generally this ultimate CHK is stored using a secret that the data owner cannot forget, like his ID for instance. It is important to note that this technique can lead to some loss in privacy for the data owner. Indeed, when several nodes own the same block, since the ciphering scheme depends on the content and not on the nodes, they produce the same ciphered blocks, and the same CHKs. Thus they are each able to know that they share a file.

It is important to note that when single-instance-storage is not considered, it is much simpler to use classic encryption techniques, e.g., based on asymmetric ciphering.

5.3   Data Availability

In this section, we explore the techniques described in the literature for improving data availability despite failures while optimizing the use of the system resources: data replication and garbage collection.

5.3.1   Data Replication

For the systems that distribute the data among a specific set of participants (Pastiche, PeerStore, CBS, Flashback), the replication mechanism is quite simple. In Pastiche, each participant entering the system looks for 5 other participants having a lot of data in common with itself. These 5 participants then become its backup partners. It can thus tolerate 4 node failures. With PeerStore, the choice of partners is done in a different manner. However, the authors say that there are ideally as many partners as there are data replicas, which is similar to Pastiche. For the systems based on DHTs, thanks to (or because of) the properties of DHTs, the global set of data stored is homogeneously distributed among the nodes. Consequently, to tolerate the departure or the failure of a participating node, the data *has to* be replicated. In practice, the data blocks are generally replicated by the node that is responsible for it (with the closest ID) on a small number of its neighbors in the identifier space. Additionally, the block can also be kept in cache on the nodes that are on the path between the owner and the node responsible for it. ABS, among the systems based on DHTs, proposes an alternative. The data owner can choose the key under which a block will be stored. When a new block is inserted in the DHT, an attempt is made to insert it with a digest of the data as the key. A digest of the key itself (this is called rehashing) can also be used to store the block on some other participant in order to move the data or to tolerate a departure or crash.

Coding techniques are also used to finely control the level of data redundancy. Many different error-correcting coding techniques can be used: erasure codes [31] like Tornado [32], Fountain codes (also called rateless erasure codes) [33] like Raptor [34], etc. The idea of erasure codes is basically that each data block is fragmented into k fragments. From these k fragments, r other redundancy fragments are computed. From these $k + r$ fragments, any k fragments are sufficient to rebuild the original data block.

Blocks are thus used to produce fragments with a controlled level of redundancy. Venti-DHash uses this technique and stores the fragments on the successors of the node responsible for the original data block. MoSAIC also uses erasure codes for the production of redundant fragments but distributes them opportunistically to the nodes encountered.

5.3.2   Garbage Collection

Pastiche, pStore and ABS offer the possibility to delete the backed up data. Only the data owner can request this operation - requests must be signed with the owner's private key. Additionally, when single instance storage is used, as a block can be stored for several owners, an owner list is associated to each data block to permit its deletion only when every owner has requested it. In PeerStore, however, such an owner list does not exist (or is incomplete), i.e., other nodes can rely on a block for their own backup without having notified the node actually storing this block. This is due to the way PeerStore implements inter-node single-instance-storage. For this reason PeerStore does not allow delete operations. FlashBack uses the notion of a "lease" whereby a data block is stored for a given duration. This duration is determined a priori and exceeds the expected duration of unavailability of the data owner. Leases can be renegotiated when they are half-expired.

5.4   Service Availability

Failures of a cooperative backup system can lead to the loss of some of the stored data, as discussed in the previous section, but can also lead to the unavailability of the entire backup system, which we address in this section. Resilience to malicious denial-of-service attacks is a wide and active research field. The approaches used to mitigate the lack of trust between the participating nodes and to tolerate these DoS attacks can be based either on the notion of reputation (a level of the trust of the partners that can be acquired either locally or transitively) or on the use of micro-economy (exchange of checks, tokens, etc.) [35–37].

We concentrate here on the attacks that are specific to cooperative backup in general and more specifically the ones we found in the cooperative backup system we studied: selfishness and retention of backup data.

5.4.1   Selfishness

Selfishness is a problem for every resource sharing system, as we saw in Section 2.2.4. Some mechanism is required to enforce fairness amongst peers - that they contribute in proportion to what they consume. Many different solutions have been proposed, most of them being based on the notion of micro-economy. We look here only at the solutions adapted to storage systems.

It is worth noting that it is not possible for a system based on DHTs to guarantee that the participants fairly contribute to the system with respect to the amount of resources they consume (see Section 4.1). Consequently, Venti-

DHash and pStore are not resilient to this type of attack. The ABS rehashing technique (see Section 5.3.1) can be used to balance the loads on the DHT but it does not take the effective usage of each node into account.

PeerStore proposes a simple solution based on pair-wise symmetrical exchanges, i.e., each one of the two partners offers (approximately) the same storage capacity that it uses. To find partners, newcomers broadcast an offer for a given storage capacity and listen to other participant replies that offer some capacity in exchange that may be different. It is then up to newcomers to decide whether or not to accept an offer. CBS also imposes symmetrical exchange relationships, restricting data placement.

Pastiche does not deal with this problem but Samsara does: it extends the notion of symmetrical exchanges with the use of *claims*. The data owner issues a claim for the node that accepts to store its data, this exchange constitutes a contract. The value of the claim represents the storage capacity of the stored data. These claims can be forwarded to another contributor when the contributor needs to store some of its own data. Finally, each node periodically checks its co-contractors to ensure that they are adhering to the contract, i.e., to verify that its claims are satisfied, by challenging its contributors. If a node breaches a contract, its partner is free to drop its data. The use of challenges can be seen as a way to compute locally a level of reputation for a contributor.

Another simple solution is proposed in CFS [38]: each contributor limits any individual peer to a fixed fraction of its space. These quotas are independent of the peer's space contribution. CFS uses IP addresses to identify peers, and requires peers to respond to a nonce message to confirm a request for storage, preventing simple forgery. This does not defend against malicious parties who have the authority to assign multiple IP addresses within a block, and may fail in the presence of network renumbering.

Several of these solutions were proposed to be extended with trusted third parties, either centralized or distributed among trusted hardware devices. For instance, PAST [39] provides quota enforcement that relies on a smartcard at each peer. The smartcard is issued by a trusted third party, and contains a certified copy of the node's public key along with a storage quota. The quota could be determined based on the amount of storage the peer contributes, as verified by the certification authority. The smartcard is involved in each storage and reclamation event, and so can track the peer's usage over time. Fileteller [40] proposes the use of micro-payments to account for storage contributed and consumed. Such micro-payments can provide the proper incentives for good behavior, but must be provisioned and cleared by a third party and require secure identities.

It is worth noting that, as a side effect, solutions based on symmetrical ex-

changes have the advantage of being resilient to flooding attacks, whereby a node tries to obtain many storage resources by flooding the network with requests. On the contrary, DHT based systems are not resilient to this type of attack due to the very nature of DHTs.

5.4.2 Retention of Backup Data

Data retention is the situation in which a contributor does not release backed up data when an owner issues a restoration request. This can be non intentional, e.g., the contributor has crashed, or is disconnected, or intentional/malicious, e.g., the contributor did not actually store the data or tries to blackmail the data owner. Generally speaking, unintentional retention should be tolerated whereas malicious retention should be prevented, or even punished.

In CBS, there is a two-fold solution to these problems: first there are periodic challenges to verify that the partners really do store the data for which they are responsible for, and second, there are rules to tolerate temporary node failures. The periodic challenges are actually read requests for randomly chosen data blocks sent to the contributors by data owners. Tolerance of temporary faults is based on a *grace period* during which a participant can be legitimately unavailable. After expiration of the grace period, the data stored for the disconnected node can be erased (the data owner locally decides to associate a bad reputation to the contributor). However, the grace period can be used to gain resources dishonestly without contributing to the system. A countermeasure is to define a *trial period*, that is longer than the grace period, during which backup and challenges are permitted but restoration is not.

This challenge technique is also used by the other studied systems, in an optimized form: a challenge concerns several blocks at a time and the response is a signature of the set of blocks [13] [15].

Samsara and PeerStore also have a slightly different way of punishing *unavailable* nodes: their blocks are progressively deleted. The probability of deletion of a block is chosen such that, given the number of block replicas, the probability of all the replicas being deleted becomes significant only after a large number of unsatisfied challenges.

6   Conclusion

Peer-to-peer/cooperative systems constitute a new emerging approach for the design of heavily distributed systems. They have very good properties regarding scalability and are thus particularly well-adapted to ubiquitous computing

scenarios. The application of peer-to-peer coopearation to backup has been rendered feasible by recent dramatic increases in storage capacity and network bandwidth. In this paper, we have surveyed the technical solutions to this problem.

We first observed that the field of cooperative backup for wide-area networks or local-area networks is very active. This research field has been recently boosted by the peer-to-peer trend and reused many of the P2P techniques: distributed hash tables, single-instance-storage, convergent encryption, etc. However, very little work has targeted mobile devices, even if cooperative backup seems to be quite appropriate for them (new data is frequently produced on many types of devices, even disconnected from the fixed infrastructure: digital cameras, phones, PDAs, laptops. However, mobile devices have their specificities (ephemeral connections, reduced energy, etc.), so many of the techniques developed for WANs and LANs cannot be applied. Much effort is still needed to alleviate the specific problems raised by frequent disconnections, ephemeral connections, limited battery power, inability to access trusted third parties, etc.

From this situation, trails that can be followed to make some progress in this field include: adequate disconnected cooperation incitatives, proper erasure codes with varying parameters, realistic mobility models, and stochastic models of the dependability of mobile devices implementing cooperative services.

References

[1] E. Growchowski, Emerging trends in data storage on magnetic hard disk drives, in: Datatech, ICG Publishing, 1998, pp. 11–16.

[2] Napster, Site internet napster : http://www.napster.com.
URL `http://www.napster.com`

[3] O. Heckmann, A. Bock, The eDonkey 2000 Protocol, Tech. Rep. KOM-TR-08-2002, Multimedia Communications Lab, Darmstadt University of Technology (Dec. 2002).
URL
`http://www.kom.e-technik.tu-darmstadt.de/publications/abstracts/HB02-1.html`

[4] P. Maymounkov, D. Mazières, Kademlia: A peer-to-peer information system based on the XOR metric, Lecture Notes in Computer Science 2429 (2002) 53–??
URL
`http://link.springer.de/link/service/series/0558/papers/2429/24290053.pdf`

[5] A. Chervenak, V. Vellanki, Z. Kurmas, Protecting file systems: A survey of backup techniques, in: Proceedings Joint NASA and IEEE Mass Storage

Conference, 1998.
URL `citeseer.ist.psu.edu/chervenak98protecting.html`

[6] S. Quinlan, S. Dorward, Venti: a new approach to archival storage, in: Proceedings of the First USENIX Conference on File and Storage Technologies, Monterey,CA, 2002, pp. 89–101.
URL `http://www.cs.bell-labs.com/sys/doc/index.html`

[7] G. Hardin, The tragedy of the commons, Science 162 (1968) 1243–1248.

[8] G. M. Greco, L. Floridi, The tragedy of the digital commons, Ethics and Information Technology 6 (2) (2003) 73.
URL `http://dx.doi.org/10.1007/s10676-004-2895-2`

[9] B. T. Loo, A. LaMarca, G. Borriello, Peer-to-peer backup for personal area networks, Tech. Rep. IRS-TR-02-015, UC Berkeley; Intel Seattle Research (USA) (May 2003).
URL `http://www.intel-research.net/seattle/publications.asp`

[10] M.-O. Killijian, D. Powell, M. Banâtre, P. Couderc, Y. Roudier, Collaborative backup for dependable mobile applications, in: Proceedings of 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (Middleware 2004), ACM, Toronto, Ontario, Canada, 2004, pp. 146–149.
URL `http://www.laas.fr/mosaic/`

[11] S. Elnikety, M. Lillibridge, M. Burrows, Peer-to-peer cooperative backup system, in: The USENIX Conference on File and Storage Technologies, Monterey, California, USA, 2002.
URL `http://www.cs.rice.edu/ sameh/`

[12] L. P. Cox, B. D. Noble, Pastiche: making backup cheap and easy, in: Fifth USENIX Symposium on Operating Systems Design and Implementation, Boston, MA, USA, 2002, pp. 285–298.
URL `http://mobility.eecs.umich.edu`

[13] L. P. Cox, B. D. Noble, Samsara: honor among thieves in peer-to-peer storage, in: Proceedings 19th ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 2003, pp. 120–132.
URL `http://mobility.eecs.umich.edu/`

[14] E. Sit, J. Cates, R. Cox, A DHT-based backup system, Tech. rep., MIT Laboratory for Computer Science (August 2003).
URL `http://project-iris.net/isw-2003/papers/sit.pdf`

[15] M. Landers, H. Zhang, K.-L. Tan, PeerStore: better performance by relaxing in peer-to-peer backup, in: Proceedings of the Fourth International Conference on Peer-to-Peer Computing, Zurich, Switzerland, 2004, pp. 72–79.
URL `http://femto.org/p2p2004/`

[16] C. Batten, K. Barr, A. Saraf, S. Treptin, pStore: a secure peer-to-peer backup system, Tech. Rep. MIT-LCS-TM-632, MIT Laboratory for Computer Science (December 2001).
URL `http://citeseer.ist.psu.edu/batten01pstore.html`

[17] J. Cooley, C. Taylor, A. Peacock, ABS: the apportioned backup system, Tech. rep., MIT Laboratory for Computer Science (2004).
URL http://www.pdos.lcs.mit.edu/6.824/reports/

[18] F. Junqueira, R. Bhagwan, K. Marzullo, S. Savage, G. M. Voelker, The Phoenix recovery system: rebuilding from the ashes of an internet catastrophe, in: Ninth Workshop on Hot Topics in Operating Systems (HotOS IX), 2003.
URL http://www.usenix.org/events/hotos03/tech/junqueira.html

[19] B. F. Cooper, H. Garcia-Molina, Bidding for storage space in a peer-to-peer data preservation system., in: ICDCS, 2002, pp. 372–.

[20] E. Hsu, J. Mellen, P. Naresh, DIBS: distributed backup for local area networks, Tech. rep., Parallel & Distributed Operating Systems Group, MIT, USA (2004).
URL
http://pdos.csail.mit.edu/6.824-2004/presentation-schedule.html

[21] A. Muthitacharoen, R. Morris, T. M. Gil, B. Chen, Ivy: a read/write peer-to-peer file system, SIGOPS Oper. Syst. Rev. 36 (SI) (2002) 31–44.

[22] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, OceanStore an architecture for global-scale persistent storage, in: Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), 2000, pp. 190–201.
URL http://oceanstore.cs.berkeley.edu/

[23] A. V. Goldberg, P. N. Yianilos, Towards an archival intermemory, in: Proceedings IEEE International Forum on Research and Technology Advances in Digital Libraries (ADL'98), IEEE Society, 1998, pp. 147–156.
URL http://www.pnylab.com/pny/papers/intermemory/main.html

[24] C. Randriamaro, O. Soyez, G. Utard, F. Wlazinski, Data distribution in a peer to peer storage system, Journal of Grid Computing (JoGC), Special issue on Global and Peer-to-Peer Computing, Springer,Lecture Notes in Computer Science.

[25] K. Lua, J. Crowcroft, M. Pias, R. Sharma, S. Lim, A survey and comparison of peer-to-peer overlay network schemes, Communications Surveys & Tutorials, IEEE 72–93.

[26] T. Lord, The GNU arch distributed revision control system (2005).
URL http://www.gnu.org/software/gnu-arch/

[27] A. Muthitacharoen, B. Chen, D. Mazières, A low-bandwidth network file system, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles, 2001, pp. 174–187.
URL http://www.scs.cs.nyu.edu/ dm/

[28] M. Rubel, Rsnapshot: a remote filesystem snapshot utility based on rsync (2005).
URL http://rsnapshot.org/

[29] L. L. You, K. T. Pollack, , D. D. E. Long, Deep Store: an archival storage system architecture, in: Proceedings of the 21st International Conference on Data Engineering (ICDE 2005), IEEE, Tokyo, Japan, 2005, pp. 804–815.
URL `http://ssrc.cse.ucsc.edu/publications.shtml`

[30] W. J. Bolosky, J. R. Douceur, D. Ely, M. Theimer, Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs, in: Proceedings of the International Conference on Measurement and Modeling of Computer Systems, 2000, pp. 34–43.
URL `http://research.microsoft.com/sn/Farsite/`

[31] H. Weatherspoon, J. Kubiatowicz, Erasure coding vs. replication: A quantitative comparison, in: Proceedings of the First International Workshop on Peer-to-Peer Systems, 2002.

[32] J. W. Byers, M. Luby, M. Mitzenmacher, Accessing multiple mirror sites in parallel: Using Tornado codes to speed up downloads, in: INFOCOM (1), 1999, pp. 275–283.
URL `citeseer.nj.nec.com/article/byers99accessing.html`

[33] J. W. Byers, M. Luby, M. Mitzenmacher, A. Rege, A digital fountain approach to reliable distribution of bulk data, in: SIGCOMM, 1998, pp. 56–67.
URL `citeseer.ist.psu.edu/article/byers98digital.html`

[34] A. Shokrollahi, Raptor codes (2003).
URL `citeseer.ist.psu.edu/shokrollahi03raptor.html`

[35] C. Grothoff, An excess-based economic model for resource allocation in peer-to-peer networks, Wirtschaftsinformatik.
URL `http://www.gnu.org/software/gnunet/`

[36] K. Lai, M. Feldman, J. Chuang, I. Stoica, Incentives for cooperation in peer-to-peer networks, in: Workshop on Economics of Peer-to-Peer Systems, 2003.
URL `http://www.sims.berkeley.edu/ mfeldman/oath/`

[37] N. Oualha, Y. Roudier, Cooperation incentive schemes and validation techniques, Tech. rep., Institut Eurecom (June 2006).

[38] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, Wide-area cooperative storage with CFS, in: Proceedings 18th ACM Symposium on Operating Systems Principles, 2001, pp. 202–215.
URL `http://www.pdos.lcs.mit.edu/chord/`

[39] A. Rowstron, P. Druschel, Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility.
URL `http://freepastry.rice.edu`

[40] J. Ioannidis, S. Ioannidis, A. D. Keromytis, V. Prevelakis, Fileteller: Paying and getting paid for file storage, in: Sixth Annual Conference on Financial Cryptography, Bermuda, 2002, p. 282299.