CABdedupe: A Causality-based Deduplication Performance Booster for Cloud Backup Services

Yujuan Tan[1], Hong Jiang[2], Dan Feng[1,*], Lei Tian[1,2], Zhichao Yan[1]
[1]School of Computer Science & Technology, Huazhong University of Science & Technology
[2]Department of Computer Science & Engineering, University of Nebraska-Lincoln
*Corresponding author: dfeng@hust.edu.cn
{tanyujuan, jarod2046}@gmail.com, {jiang}@cse.unl.edu, {dfeng,ltian}@hust.edu.cn

***Abstract*—Due to the relatively low bandwidth of WAN (Wide Area Network) that supports cloud backup services, both the backup time and restore time in the cloud backup environment are in desperate need for reduction to make cloud backup a practical and affordable service for small businesses and telecommuters alike. Existing solutions that employ the deduplication technology for cloud backup services only focus on removing redundant data from transmission during backup operations to reduce the backup time, while paying little attention to the restore time that we argue is an important aspect and affects the overall quality of service of the cloud backup services. In this paper, we propose a *CAusality-Based deduplication performance booster for both cloud backup and restore operations*, called CABdedupe, which captures the causal relationship among chronological versions of datasets that are processed in multiple backups/restores, to remove the unmodified data from transmission during not only backup operations but also restore operations, thus to improve both the backup and restore performances. CABdedupe is a middleware that is orthogonal to and can be integrated into any existing backup system. Our extensive experiments, where we integrate CABdedupe into two existing backup systems and feed real world datasets, show that both the backup time and restore time are significantly reduced, with a reduction ratio of up to $103 : 1$.**

I. Introduction

With the increasing popularity of the cloud platform, *cloud backup services* have been attracting a great deal of attention from both industry and academia. Compared with traditional backup methods, the new "pay-as-you-go" model in the cloud backup environment provides users with remote online backup/restore services at a reasonable performance/cost ratio. This has been received favorably by telecommuting employees, Remote Office/Branch Offices (ROBO), and Small and Medium Businesses (SMBs) who lack sufficient remote backup strategies due to the limited IT staffs and constrained budgets. Moreover, because most SMBs focus on their businesses, they do not have the time or desire to become backup experts. As a result, many SMBs prefer to outsource their backup/restore tasks, which is well leveraged by cloud backup services.

In both the traditional backup and cloud backup environments, there are two critical performance metrics, *backup window (BW)* and *recovery time objective (RTO)*, to evaluate the backup and recovery performances respectively. Backup window represents the time spent on sending specific datasets to the backup destination while recovery time objective denotes the maximum amount of downtime a business is willing to accept after data disasters. A recent ESG (i.e, Enterprise Strategy Group) research [1] has indicated that about 58% of professionals in SMBs can tolerate no more than four hours of downtime before experiencing significant adverse effect. This will be a much bigger challenge for cloud backup services due to the relatively low bandwidth of WAN (Wide-Area Network) [2] that underpins the cloud backup platform. For example, it is only able to transmit or restore about 11.52GB data with the measured network bandwidth of 800KB/s [3] during a four-hour period, which is far less than the target amount of restored dataset of about 100GB [4] in SMBs on average and thus fails to achieve the RTO of four hours since *the recovery is premised on restoring the required amount of data first*. Besides RTO, abundant data transmission overheads also lengthen the backup window in cloud backup environments. The ESG survey indicates that 64% of organizations are under pressure to reduce backup times and 48% of them need to reduce recovery times [5]. Therefore, it is important and critical to adopt network-efficient approaches to the cloud backup environment to improve *both the data backup and restore (i.e., a critical step for recovery) performances*, if cloud backup as a service is to become practical and cost-effective.

Recently, data deduplication has emerged as an attractive lossless compression technology that has been employed in various network efficient and storage optimization systems [6], [7], [8], [9]. In cloud backup environments particularly, many products, such as EMC Avamar [10], Cumulus [3], Asigra [11], Commvault Simpana [12], and etc., have adopted the source-side data deduplication technology to reduce the backup time by removing redundant data from transmission to backup destinations. However, despite the critical importance of the restore time in achieving a reasonable RTO as discussed earlier, much less attention has been paid to reducing the time spent on restoring data

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.76

from remote backup destinations to the user's local computer for cloud recovery. More seriously, Symantec's annual IT Disaster Recovery survey [13] in 2009 has observed that the average cost per hour of downtime is much higher than ever before, which further stresses the importance of the data restore time for data recovery operations. Thus *an efficient deduplication approach in that cloud backup environment must remove the redundant data in not only backup operations but also restore operations so as to optimize both the backup and restore performances*.

Our preliminary studies (see Section II) have found that a large amount of redundant data exists among multiple data backups and restores. Most datasets processed *in both data backups and data restores* are evolved from their previous backed-up versions with relatively minor modifications, insertions or deletions, resulting in most files and data chunks unchanged in their entirety after backups. By capturing and preserving this causal relationship among chronological versions of datasets, it is possible to fast identify which files have been changed and which data chunks differ among multiple file versions, thus helping remove the unmodified data from transmission to significantly reduce the total transferred dataset in both backup and restore operations. This reduction in data transmission in turn will help reduce both the backup time and restore time.

The above observations and analyses motivate us to propose a CAusality-Based deduplication performance booster for cloud backup services, called *CABdedupe*, which captures and preserves the *causal relationship* among chronological versions of datasets in the client site to remove the redundant data from data transmission for both backup and restore operations. CABdedupe is a middleware that is orthogonal to and can be integrated into any existing backup system and consists of three key functional components, File Monitor, File List, and File Recipe. File Monitor is a daemon process working at the file system level in the client site to monitor the file operations on the protected datasets, while File List and File Recipe are two data structures that record the above file operations observed by File Monitor. The combined functionality of these three components effectively captures the causal relationship information among multiple dataset versions, which helps CABdedupe quickly identify *which files and which data chunks have been changed or remained unchanged*, so as to *quickly remove the unmodified data from data transmissions for both backup and restore operations*. Our extensive experiments, with CABdedupe integrated into two existing backup systems and driven by real world datasets, show that both the backup time and restore time are significantly reduced, with a reduction ratio of up to $103 : 1$.

In summary, our proposed CABdedupe scheme for cloud backup systems provides the following advantages:

- *Network bottleneck mitigation.* With the help of the causal relationship among chronological versions of datasets, CABdedupe exploits the data redundancy among multiple backups/restores to remove redundant data from data transmission, thus alleviating the network bottleneck during each backup/restore operation.
- *Performance optimization.* By reducing the data transmission overheads, the two key performance measures of backup/restore operations, backup time and restore time, are both significantly improved.
- *Modular configurability.* CABdedupe as a middleware is a performance accelerator to existing backup systems. It can be implemented as an optional application-transparent module. The failure of CABdedupe, should it ever happen, will only cause unmodified data to be transmitted for backups/restores, but will not disturb backups/restores themselves or cause their failures.
- *Lightweight and flexibility.* CABdedupe removes the redundant data exiting in the same client instead of across different clients in the cloud, thus alleviating the load of service provider and supporting switches to different cloud backup providers.

The rest of this paper is organized as follows. In the next section we discuss the motivations for our research. In Section III, we present the system architecture and CABdedupe framework. The detailed exploration and exploitation of the causality information are described in Section IV. Section V presents our experimental results with real world datasets. Section VI presents the related work and Section VII concludes the paper.

II. Motivation

In cloud backup environments, the low bandwidth of WAN between the source client and backup destination lengthens both the backup time and restore time. The ESG research indicates that 39% of organizations that have tried to run backups over WAN report that both backups and restores take too long and 31% of them report that the cost of WAN bandwidth is too high [14]. An alternative way to solve this problem is to reduce the amount of data transmitted over WAN to accelerate the backup and restore processes, thus shortening the backup time and restore time. In this section, we discuss the causality-induced data redundancy in existing backup systems that provides a potential opportunity for us to remove redundant data from data transmission to alleviate the network bottleneck in the cloud backup environment.

A. Causality-induced Data Redundancy

In backup systems, there are many backed-up versions of the same dataset stored at the backup destination due to multiple full and incremental backups. Except for the initial full backups, the dataset backed up each time is evolved from its previous backed-up versions with data modifications. As a result, it is the modified data, rather than the unmodified data that is already stored at the backup destination by previous backed-up versions, that is required to be transmitted each time. The same is true during the data restores. Each restore operation takes place after data corruptions and needs to restore the corrupted dataset to a previous backed-up version stored at the backup destination. Just as the dataset backed up each time, the corrupted dataset requiring restore is evolved from its previous backed-up versions with data modifications done in users' local computers. Thus the data that has not been modified after backups has no need to be transmitted from the backup destination. Therefore during either data backup or restore operations, it is possible to reduce the amount of data transmitted over WAN by removing from transmission the unmodified data shared between the current dataset being processed (i.e., dataset to be backed up or restored each time) and its previous backed-up versions. The identification and removal of the unmodified data become the key to improving both the cloud backup and cloud restore performances .

In what follows, we present the causal connections among files in multiple versions of the same dataset to reveal the prevalent existence of unmodified data during each backup/restore operation.

- *Unchanged Files.* Either in the directory backups or directory restores, there are many files kept intact after their last backups. Policroniades et al. [15] noted that in real filesystems most accesses to files are read-only, implying that most files remain unchanged after their creations. Moreover, Microsoft's 5-years file metadata study results [16] also show that a large percentage of files have not been modified since they were initially copied onto file systems, and this percentage has grown from 66% to 76% from 2000 to 2004. A good example of data corruptions is virus attacks. Some viruses only attack the files with specific file types, for example, the "mmc.exe" virus only attacks all the executable files in the Windows XP operating system. When one directory is attacked by this kind of viruses, there will be only a limited number of files infected and most of the rest will remain unchanged.
- *Modified Files.* In most cases, the individual files that require data backups or restores have always been changed with data modifications, insertions, or deletions after their backups. Nevertheless, given that most data modifications are concentrated on a small subset of data blocks in a short time in typical file systems [17], a lot of unmodified data is likely to exist between the current file version and its previous one during each backup/restore operation.
- *Deleted Files.* Besides the unchanged and modified files, typically there are some files that have been deleted after their last backups. For example, some viruses such as the "Nimaya" virus always deletes the files with the suffix of ".gho" in their filenames; users sometimes delete files accidently or deliberately to reclaim space after their backups. Although these file deletions do not affect the backup operations since there is no need to back up deleted files, their deletions force all the deleted files to be transmitted entirely from the backup destination to the users' local computers for data restores.

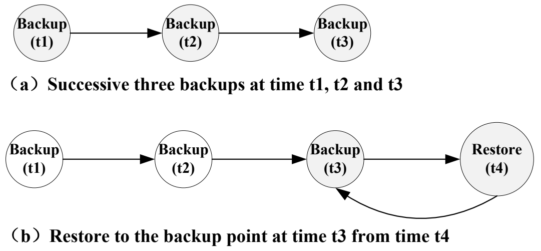

Backup (t1)
Backup (t2)
Backup (t3)
（a）Successive three backups at time t1, t2 and t3
Backup (t1)
Backup (t2)
Backup (t3)
Restore (t4)
（b）Restore to the backup point at time t3 from time t4

Figure 1: Successive Backups and Restores Illustrated in Directed Graphes.

The three cases described above intuitively describe the causal relationships among the different versions of the same dataset in backup/restore scenarios, suggesting that a large amount of unmodified data can be identified and removed from transmissions to reduce the total amount of data transmitted. By capturing these causal relationships among files in different dataset versions, it is possible to quickly identify which files and which data chunks have been changed or remained unmodified, thus removing the unmodified data from data transmission to improve both the backup and restore performances in the cloud backup environment.

B. Mining Data Redundancy

In this subsection, we introduce an alternative data redundancy mining model to analyze the removal of the unmodified data that exists among multiple backup and restore operations as mentioned before.

In the following, we first present five key terms that are used to describe this redundancy mining model.

- S_t. The size of the dataset that is processed in each backup or restore operation at time t.
- $S_{t1} \cap S_{t2}$. The total amount of the unmodified data that is shared between the two versions of the same dataset processed at time t_1 and at time t_2.
- $S_{t1} \cup S_{t2}$. The total amount of the data that is processed either at time t_1 or at time t_2.
- B_t. The amount of the data that needs to be transmitted for the backup operation at time t.
- R_t. The amount of the data that needs to be transmitted for the restore operation at time t.

To clearly present the data redundancy mining model, we use a Directed Graph (DG) to show several successive data backups and restores of the same dataset in Figure 1.

Figure 1(a) shows three successive backup operations that run at t_1, t_2 and t_3 ($t_1 < t_2 < t_3$) respectively, in which the backup at t_1 is the initial full backup of this dataset. After removing the unmodified data from these three backups, the amount of data requiring backup can be expressed as:

$$\begin{cases} B_{t1} = S_{t1} \\ B_{t2} = S_{t2} - S_{t2} \cap S_{t1} \\ B_{t3} = S_{t3} - (S_{t3} \cap S_{t1}) \cup (S_{t3} \cap S_{t2}) \end{cases} \tag{1}$$

Figure 1(b) shows a successive restore operation at time t_4 after three preceding backups, which tries to restore the corrupted dataset to the backup point t_3. In traditional restore methods, the dataset at the backup point t_3 should be entirely transmitted so that $R_{t4} = S_{t3}$ since no deduplication approach is employed . Mindful of the causal relationship among successive backups and restores, we observe that the data kept intact at t_4, after its backup at t_3, requires no restorations, and thus

$$R_{t4} = S_{t3} - S_{t3} \cap S_{t4} \tag{2}$$

under the assumption that the data in $S_{t3} \cap S_{t4}$ will not be modified during the restore operation. On the other hand, when the dataset at time t_4 cannot be accessed after data disasters, R_{t4} will be equal to S_{t3}, similar to that in the traditional restore methods.

This data redundancy mining model quantifies the amount of the data that is required to be transmitted for each backup/restore operation after removing its unmodified data from transmission. Motivated by this causality-induced data redundancy existing among multiple backups/ restores, and combined with the data redundancy mining model, we propose a causality-based deduplication scheme to improve both the cloud backup and restore performances, which will be detailed in the next sections.

III. CABdedupe Architecture

In this section, we present the system architecture of CABdedupe and describe how it can be applied to the existing cloud backup systems. As shown in Figure 2, the assumed general backup system consists of two software components: Client and Server. Client, installed on the user's local computer, is responsible for sending/retrieving the backup dataset to/from the backup destination, while Server, located in service provider's data center, is responsible for storing/returning the backed-up dataset from/to Client. To succinctly illustrate the role of CABdedupe in a backup system, we use Backup-Client and Backup-Server to represent the functionalities of the original client and server modules in existing backup systems. CABdedupe consists of CAB-Client and CAB-Server.

A. CAB-Client

CAB-Client is composed of two functional modules, the Causality-Capture module and Redundancy-Removal module. The former, consisting of File Monitor, File List and File Recipe Store, is responsible for capturing the causal relationships among different files, while the latter is responsible

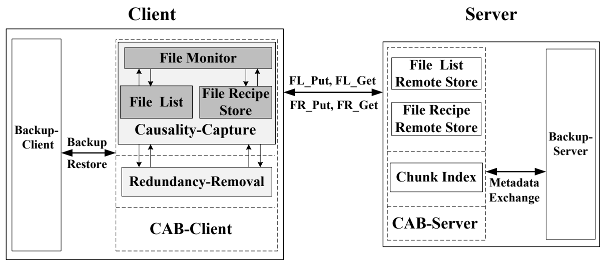


Figure 2: System Architecture.

for removing the unmodified data for each backup/restore operation with the help of the captured causality information by the former. Of the components of the Causality-Capture module, File Monitor is a daemon process that works at the file system level to keep track of some specific file operations, including file rename, file create, file delete, and file-content modification, File List is responsible for logging these file operations and File Recipe Store is responsible for saving the file recipes (i.e., the fingerprints of data chunks) of the backed-up files. These three components collectively capture the causal relationships among the different files in multiple backups and restores, as detailed in Section IV.

B. CAB-Server

CAB-Server, with File List Remote Store and File Recipe Remote Store as its components, stores the file lists and file recipes sent from CAB-Client, which ensures the availability of the causality information captured by CAB-Client in case of CAB-Client's corruptions. However, due to data transmission overheads, the file lists and file recipes stored in CAB-Server are not updated in the same timely fashion as that stored in CAB-Client.

Another component of CAB-Server, Chunk Index, is responsible for locating and retrieving the data chunks stored in CAB-Server for data restores. However, this Chunk Index component can be excluded from CAB-Server if the original backup system has exploited the chunk-level deduplication capability and it's intended functionality already exists in Backup-Server.

C. Interface

In this subsection, we describe several interfaces that are used to integrate CABdedupe into the existing backup systems as showed in Figure 2. These interfaces are classified into three categories as follows.

The first interface, with Backup and Restore, is used to connect CAB-Client with Backup-Client. During each backup/restore operation, Backup-Client communicates with CAB-Client through this interface to remove the unmodified data from transmission by the Redundancy-Removal module.

The second interface, with FL_Put, FL_Get, FR_Put and FR_Get, is used to store the file lists and file recipes in CAB-Server. FL_Put and FL_Get are used to exchange the file lists

, while FR_Put and FR_Get are used for the exchange of file recipes between CAB-Client and CAB-Server.

The third interface, called Metadata Exchange, is responsible for the communication of chunk metadata information between CAB-Server and Backup-Server. The metadata information, including chunk fingerprints, chunk addresses, chunk lengths, and etc., is used for building chunk index in CAB-Server during backups and locating the corresponding data chunks during restores. However, this interface will not be necessary if there is no Chunk Index in CAB-Server as described in Section III-B.

D. Redundancy Removal

In CABdedupe, the unmodified data is removed by the Redundancy-Removal module with the help of the causality information stored in File List and File Recipe Store, as follows.

1) Backup: During each backup operation, Backup-Client communicates with CAB-Client through the following four steps to remove its unmodified data.

- *Step 1.* Check File List to find which files have been modified after their last backup.
- *Step 2.* For each modified file, check whether the file metadata and file content have been changed.
- *Step 3.* If some files have only file metadata modifications, CAB-Client notifies Backup-Client to only back up their modified file metadata instead of the whole file content. Otherwise, the files are chunked by the Rabin Fingerprints algorithm [18] and each chunk is named by the SHA-1 hash function [19] to filter out the unmodified data chunks in the next step.
- *Step 4.* This step finds and removes the unmodified data chunks by checking the file recipes of the previously backed-up versions. After filtering out the unmodified data chunks, CAB-Client notifies Backup-Client to back up the remaining modified data chunks to Backup-Server, and the file recipes of the new backed-up files versions are saved in File Recipe Store for redundancy exploitation in future backup/restore operations.

2) Restore: During each restore operation, CABdedupe takes the following four steps to remove its unmodified data, similar to the backup operation. The only difference between the backup and restore operations is that, during each backup operation, it removes the unmodified data chunks that have already been stored in Server by previous backups, while during each restore operation, the redundant data chunks removed are those kept intact in Client after their last backups (i.e., one backup point to be restored).

- *Step 1.* Get the file recipes of all the files in the restored dataset (i.e., one backed-up version).
- *Step 2.* For each file in the restored dataset, check File List to find its current file version existing in Client to see whether it has been changed or not after its last backup (i.e., the restored backup point).
- *Step 3.* If the files are kept intact after their backups, CAB-Client notifies Backup-Client that these file are not required to be retrieved from Backup-Server. Otherwise, the files are chunked by the Rabin Fingerprints algorithm and each chunk is named by the SHA-1 hash function to find the unmodified data chunks in the next step.
- *Step 4.* This step is to find and remove the unmodified data chunks by checking the file recipes of the files in the restored dataset. After filtering out these unmodified data chunks, CAB-Client notifies Backup-Client to retrieve the remaining modified data chunks from Backup-Server.

IV. Exploring and Exploiting Causality Information

In CABdedupe, the causality information among different files plays a crucial role in removing the redundant data during the backup/restore operations. In this section, we present how CABdedupe uses its key components to capture and leverage the causality information.

A. File Monitor

File Monitor is a daemon process running at the file system level to monitor the file-system-level system calls to keep track of some file operations. It is triggered when the user's system is bootstrapped and terminated when the system is shut down. To reduce its overhead, CABdedupe only captures a small portion of file operations so as to keep File Monitor idle or lightly loaded during most of the time.

- *File set consideration.* CABdedupe only focuses on the directories and files that have been initially fully backed up, ignoring any other files existing in the same file system. Moreover, given that most files are small and holding a very small amount of data in typical file systems[16], CABdedupe excludes these small files to further reduce the size of the file set.
- *File operation consideration.* CABdedupe only captures the first file write operation after each backup, to track whether the files (i.e., including file metadata and file content) have been modified, ignoring all the other file write and read operations. At the same time, two other special file operations, file rename (i.e., file content is not changed) and file delete, are also captured by CABdedupe to track the data redundancy presented in Section II. For the file deletions, CABdedupe renames the deleted files and makes them only visible to CAB-Client, thus to prevent them from actual deletions in case of later file restores. However, restricted by the available storage space in Client, not all the files can be prevented from file deletions forever. When the space usage reaches a pre-set limit, CABdedupe uses a FIFO (First In First Out) replacement algorithm to reclaim the used space.

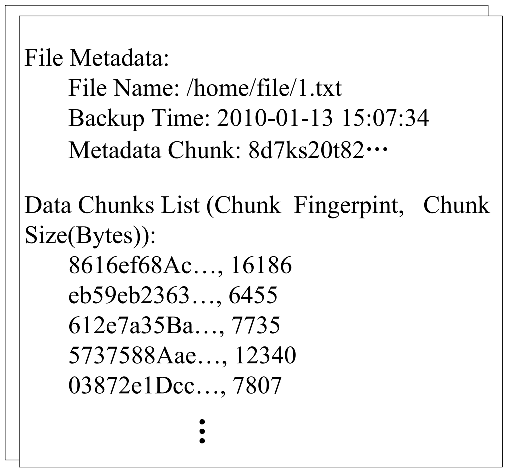
B. File List

File List (FL) is a table that is used to record the file operations captured by File Monitor. Table I depicts its four entries with an example.

- *Last Backup Time*. The time when the file was the most recently backed up.
- *Modification Flag*. This flag indicates the files' modification status after their last backups, including file create, file rename, file delete, file-metadata modification and file-content modification. Note that the status of some successive modifications may be overlapped. For example, a file create operation creates both its metadata and content, thus resulting in both file-metadata modification and file-content modification.
- *Original File Name and New File Name*. These two entries are used to keep track of the file rename operations. The Original File Name represents the original name by which the file is backed up and the New File Name denotes the new file name after the rename operations.

The above four entries describe the causality information that is stored in File List. During each backup/restore operation, CABdedupe checks File List to find the files that have been modified since the most recent backups. In order to limit the size of File List so as to accelerate this search process, CABdedupe excludes the small files along with their file operations as discussed in Section IV-A. In our experiments, we observed that 86.7% of files are smaller than 8KB during directory backups/restores, but these small files only occupy 2.6% of the total storage space. By excluding those files smaller than 8KB from the CABdedupe process, it can remove 86.7% of the files that the Causality-Capture module must otherwise consider while only failing to eliminate 0.54% of the redundant data. Moreover, CABdedupe can use the FIFO or LRU replacement algorithms, or delete the outdated entries in File List to further restrict the growth of its size.

Table I shows one example of File List that contains all of the file operations, including file create, file rename, file delete, file-metadata modification and file-content modification, which CABdedupe mainly focuses on. In this table, the files "/home/file/1.txt" and "/home/file/2.txt" has been modified, file "/home/file/3.txt" has been renamed to file "/home/data/3.txt", file "/home/file/4.txt" has been deleted and file "/home/file/5.txt" has been created after the backup of directory "/home" at 15:07:34 on January 13, 2010. During the next backup, the modified file metadata and file content of the files "/home/file/1.txt" and "/home/file/5.txt", must be backed up, while for the files "/home/file/2.txt" and "/home/file/3.txt", only their modified file metadata is needed to be backed up. During the next restore operation (i.e., restored to the backup point at 15:07:34 on January 13, 2010), for the files "/home/file/1.txt" and "/home/file/2.txt",



File Metadata:
    File Name: /home/file/1.txt
    Backup Time: 2010-01-13 15:07:34
    Metadata Chunk: 8d7ks20t82···

Data Chunks List (Chunk Fingerpint, Chunk Size(Bytes)):
    8616ef68Ac..., 16186
    eb59eb2363..., 6455
    612e7a35Ba..., 7735
    5737588Aae..., 12340
    03872e1Dcc..., 7807
    ⋮

Figure 3: An example of file recipe.

it is required to retrieve their file metadata and file content from the backup destination, while for files "/home/file/3.txt" and "/home/file/4.txt", it is only required to rename them. So according to the causality information stored in File List, CABdedupe can easily find out which files have been modified after their most recent backups.

C. File Recipe Store

File Recipe Store is a container used to store the file recipes. As showed in Figure 3, each file recipe consists of two parts, the file metadata and data chunk list. The former contains file name, file backup time and metadata chunk fingerprint, while the latter includes the chunk fingerprints and chunk sizes of all the data chunks that constitute a specific file. With the help of these file recipes, CABdedupe can easily locate the unmodified data chunks to be removed from transmission for each backup/restore operation (see Section III-D).

D. Causality Information Consistency

During each backup/restore operation, CABdedupe relies heavily on the causality information that is stored in File List and File Recipe Store to identify and remove unmodified data chunks. Therefore, CABdedupe must ensure that the causality information stored in File List and File Recipe Store is consistent with that actually existing among the datasets from multiple backups/restores. The inconsistency of this information can lead to false positives in that unmodified data is regarded as modified data or false negatives in that modified data is regarded as unmodified data. Although the false positives can not affect the routine backups/restores but only degrade the effectiveness of CABdedupe, the false negatives can mislead the routine backups/restores to skip the backups or restores of some modified data chunks.

To avoid this problem, CABdedupe only stores the consistent causality information to File List and File Recipe Store in CAB-Client. After each backup/restore operation, CABdedupe buffers and accumulates the newly captured

Table I: An example of file list.

Original File Name	Revised File Name	Modification Flag	Last Backup Time
...	...	...	...
/home/file/1.txt	——	MetaModify|ContentModify	2010-01-13 15:07:34
/home/file/2.txt	——	MetaModify	2010-01-13 15:07:34
/home/file/3.txt	/home/data/3.txt	Rename|MetaModify	2010-01-13 15:07:34
/home/file/4.txt	/home/delete/home-file-4.tmp	Delete	2010-01-13 15:07:34
——	/home/file/5.txt	Create|MetaModify|ContentModify	2010-01-13 15:07:34
...	...	...	...

causality information until the next backup/restore operation, thus ensuring that the captured information among these two backups/restores is accurate and consistent. Moreover, CABdedupe stores this consistent causality information in both CAB-Server and CAB-Client. When the causality information stored in CAB-Client is corrupt, CABdedupe retrieves the same information from CAB-Server and rebuilds File List and File Recipe Store to improve the next backup/restore performance. However, due to the lack of the new causal relationship between the dataset processed in the next backup/restore and its previous backed-up versions in the newly built File List and File Recipe Store, CABdedupe must scan the file system to find the unmodified data chunks for the optimization of the next backup/restore operation. This detailed process, and the detailed maintenance of the causality information consistency implemented by CABdedupe, are both omitted in this paper due to space constraints.

V. Performance Evaluations

We have built our prototype systems and fed real-world data sets to evaluate CABdedupe's performance. The goal is to answer the following questions.

- How much redundant data can CABdedupe remove for each backup/restore operation?
- How effective is CABdedupe in reducing the backup time and restore time?
- How much overhead does CABdedupe introduce?

A. Experimental Setup

1) Prototype Systems: We have integrated CABdedupe into two existing backup systems.

- ***Cumulus***. Cumulus is a cloud backup system [3] that exploits the source-side chunk-level deduplication approach to remove the redundant data from transmission for backup operations while ignoring the restore operations. We select it as a baseline system to assess *how effective CABdedupe is in optimizing the backup/restore performances of an existing cloud backup system that already exploits the data deduplication technology*. We denote the prototype system of Cumulus integrated with our CABdedupe as ***Cumulus+CAB***. Cumulus+CAB has inherited all the intrinsic characteristics of Cumulus, such as supporting switches to different storage providers by storing CABdedupe's file lists and file recipes in normal files in providers' sites.

Table II: Key Statistics of One Author's Home Directory.

Duration		31 days
The Status of The Dataset on The 31st Day	Entries	31647
	Files	28837
	Average(File Size)	314.5KB
	Median(File Size)	43.1KB
	Maximum(File Size)	253.6MB
	Total(File Size)	9.07GB
Average Update Rates	New data/day	12.9MB
	Changed data/day	39.7MB
	Total data/day	52.6MB

- ***MBacula***. As its name implies, MBacula is developed based on an open-source backup software called "Bacula" [20] with some modifications to Bacula (version 2.0.3), such as the data layout optimization, improving the flow control of backups/restores, and etc. However, MBacula does not exploit the deduplication technology to optimize both the backup and restore performances. We select it as another baseline system to assess *how effective CABdedupe is in optimizing the backup/restore performances of an existing backup systems that does not exploit the data deduplication technology*. We call the prototype system of MBacula integrated with our CABdedupe as ***MBacula+CAB***. In MBacula+CAB, CABdedupe's components, such as File List, File Recipe Store, Chunk Index, and etc., are implemented in a database form for fast queries and retrievals of the causality information.

In these prototype systems, each client or server machine is featured with two-socket dual-core 2.1GHz CPUs, a total of 2GB memory, 1 Gbps NIC cards, and a 500GB hard drive.

2) Datasets: We report the experimental results based on three datasets with different characteristics. The first dataset, used to evaluate the directory backups/restores, is the full backups of one author's home directory lasting for one month, totaling about 275.17GB data. Its key statistics is showed in Table II. The other two datasets, a database file generated in MBacula and a tar file of Linux source tree, both have five backed-up versions as described in Table III, which are used to evaluate the restore performances of individual files.

B. Restore Performance

In this subsection, we show both the directory restore performance and individual file restore performance separately to analyze the impact of the restore optimizations by CABdedupe. However, because it is hard to trigger realistic data disasters, we inject data corruptions by developing a program that randomly selects some files to apply the file rename operations, file modifications (including data insertion, deletion and modification) and file deletions to the last version of the dataset to simulate data disasters, so that the restore operations are invoked to restore the datasets to their previous backed-up versions to show the restore performances. While we realize that these are contrived data disasters, we have not been able to find a better way to simulate/emulate real-world disasters.

1) Single Directory Restore Performance: We use the datasets of 31 full backups of one author's home directory(i.e., not the whole file system) to evaluate the directory restore performance. There are a total of 30 simulated data restore operations by restoring this directory on the 31st day to its previous backed-up versions on the 1st day, 2nd day, 3rd day, ..., 30th day.

Figure 4 shows the amount of the redundant data removed and the corresponding transfer cost reduced by CABdedupe during each of these 30 data restores. We adopt the transfer prices set for Amazon S3 in April 2010 as the pricing criteria, at about 0.15 dollars per GB [21]. As shown in this figure, the amount of the removed data during each restore is gradually increasing as the restored backup point approaches the 31st day, implying that more data transfer cost can be reduced when restoring the directory to the more recent backup points.

To more accurately quantify the optimizations of the directory restore performance , we focus on the restore operation that restores the home directory on the 31st day to its backed-up version on the 30th day. During this restore operation, the redundant data that can be removed by CABdedupe is about 8.93GB, and the transfer costs that can be reduced by Cumulus+CAB and MBacual+CAB are about \$1.34 and \$1.32 respectively. From these results, we find that the reduced transfer cost by MBacual+CAB is proportional to the amount of the removed redundant data by CABdedupe. But this is not the case for Cumulus+CAB. This is because Cumulus+CAB groups many data chunks into one segment as a file stored in service provider's site as Cumulus does, and, as a result, retrieving a single data chunk requires the retrieval of the entire segment, followed by the extraction of the required data chunk from the segment. This results in many irrelevant data chunks being transmitted by Cumulus+CAB. We set the segment size to 16MB in both Cumulus+CAB and Cumulus.

Figure 5 compares the restore times of MBacula and MBacula+CAB, Cumulus and Cmulus+CAB, where we simulate a network environment with different network bandwidths: 800KB/s, 1MB/s, 2MB/s, 4MB/s, 8MB/s. As the figure shows, both Cmulus+CAB and MBacula+CAB significantly reduce the restore times of their respective original (CAB-less) backup systems, in particular with a reduction ratio of 61.9 : 1 by MBacula+CAB under the network bandwidth of 800KB/s. This is because, during directory restores, most of the files are kept intact after their backups that require no data transmission, given that most data writes are centered on a small subset of files in typical file systems [15]. On the other hand, due to the transmission of many irrelevant data chunks by Cumulus and Cumulus+CAB, the restore times of both Cumulus and Cumulus+CAB are longer than that of MBacula and MBacula+CAB.

Table III: The file sizes of the five backed-up versions of the database file and tar file.

File Version	**File Size(MB)**	
	Database File	**Tar File**
1st version	6.06	294.06
2nd version	7.62	353.21
3rd version	8.38	365.83
4th version	9.32	382.42
5th version	10.07	394.91

2) Individual File Restore Performance: Besides directory restores, the restore of individual files is another common restore operation in cloud backup environments. In most cases, the individual file restore happens when the file in the client site is changed or deleted, which is different from the directory restore where many files are kept intact after their backups. Thus during individual file restore operations, the amount of redundant data that can be removed is much less than that in directory restores. In our experiments, we select a database file generated in MBacula and a tar file of the Linux source tree as our datasets to show the individual file restore performances. Both the database and tar files have five backed-up versions as descried in Table III. We simulate four restore operations by restoring the 5th file version to the 1st, 2nd, ..., and 4th file version for each of the two files.

Table IV shows the amount of the redundant data removed during each restore operation. Similar to directory restores, the amount of the redundant data removed by CABdedupe increases as the file version approaches the 5th version. However, this common trend observed in both the directory restores and the individual file restores should not be regarded as a rule since the amount of this redundant data is heavily dependent on the amount of data modifications to each specific dataset.

Table V compares the amount of the data required to be transmitted by MBacula+CAB and Cumulus+CAB during those restores presented in Table IV. From these results, we find that Cumulus+CAB transmits much more data than MBacula+CAB, especially in the restores of the database file. This is because in Cumulus+CAB, each backed-up version of the database file is smaller than 16MB, and each time we have grouped all the data chunks in one file stored in the service provider's site, thus resulting in no data reduction since any restore must retrieve the entire database file.

Figure 6 compares the restore times of MBacula and

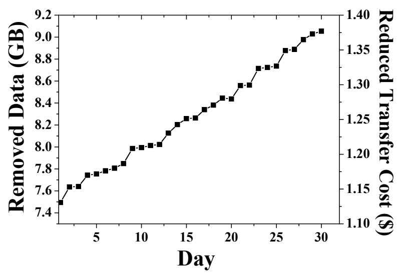
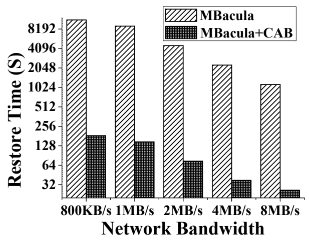
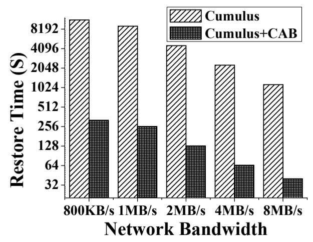


Figure 4: The amount of the redundant data removed and the corresponding transfer cost reduced by CABdedupe during each of the 30 restores.

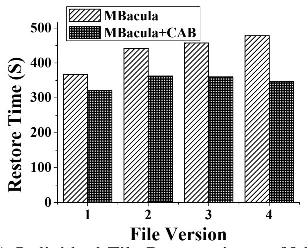


(a) Directory Restore times of MBacula and MBacula+CAB

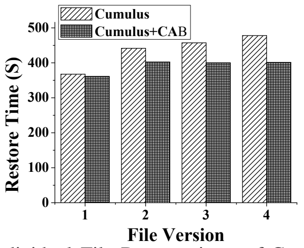


(b) Directory Restore times of Cumulus and Cumulus+CAB

Figure 5: The comparison of the directory restore times of MBacula and MBacula+CAB, Cumulus and Cmulus+CAB for restoring one author's home directory on the 31st day to its backed-up version on the 30th day.



(a) Individual File Restore times of MBacula and MBacula+CAB



(b) Individual File Restore times of Cumulus and Cumulus+CAB

Figure 6: The comparison of the individual file restore times of MBacula and MBacula+CAB, Cumulus and Cmulus+CAB for restoring the 5th version of the tar file to its 1st, 2nd , ..., and 4th version.

Table IV: The amount of the redundant data removed from transmission by CABdedupe during the individual file restore operations for restoring the 5th versions of database and tar files to their 1st, 2nd , ..., 4th versions.

File Version	Database File	Tar File
1st version	6.05MB	37.05MB
2nd version	7.6MB	63.28MB
3rd version	8.37MB	77.75MB
4th version	9.3MB	105.4MB

Table V: The amount of the data transmitted by MBacula+CAB and Cumulus+CAB during the individual file restores for restoring the 5th versions of database and tar files to their 1st, 2nd , ..., 4th versions.

	Database File (MB)		Tar File (MB)	
	MBacula+CAB	Cumulus+CAB	MBacula+CAB	Cumulus+CAB
1st version	0.01	6.06	257.1	289.1
2nd version	0.02	7.62	290.08	322
3rd version	0.01	8.38	288.16	320.16
4th version	0.02	9.32	277.11	321.12

MBacula+CAB, Cumulus and Cumulus+CAB when restoring the 5th version of the tar file to its 1st, 2nd, ..., and 4th version respectively under the network bandwidth of 800KB/s. This figure reveals that MBacula+CAB incurs 27% less restore time than MBacula, whereas the restore time of Cumulus+CAB is only 12% less than that of Cumulus due to the backup format stored in the service provider's site as described in above sections.

C. Backup Performance

Figure 7 shows the amount of the cumulative redundant data removed by CABdedupe during these 31 backups. As shown in this figure, the cumulative data required to be backed up is about 275.17GB. After the CABdedupe processing, the cumulative data actually backed up is only 10.26GB, achieving a compression ratio of 26.82. The cumulative reduced storage cost by CABdedupe is about 39.74 dollars, calculated based on the storage service prices set for Amazon S3 [21] in April 2010, at about $0.15 per GB per month.

To quantify the optimizations of backup performances by CABdedupe, we focus on the last backup of the home directory on the 31st day and report the backup times of MBacula and MBacula+CAB, Cumulus and Cumulus+CAB in Figure 8. We simulate a network environment with different network bandwidths: 800KB/s, 1MB/s, 2MB/s, 4MB/s, 8MB/s. From these results, we find that MBacula+CAB significantly reduces the backup times of MBacula, with a reduction ratio of up to 1031. However, the backup times of Cumulus+CAB are only about 1.4% less than those of Cumulus. This is because, before integrating CABdedupe, Cumulus has already exploited the source-side deduplication technology to remove the redundant data among different file versions to improve the backup performance, resulting in very limited room for Cumulus+CAB to improve by additionally considering the file rename operations and file copy operations.

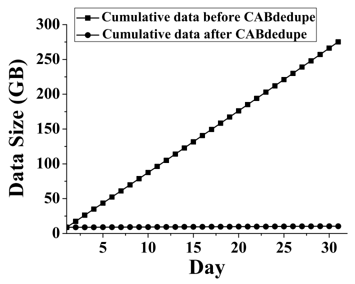
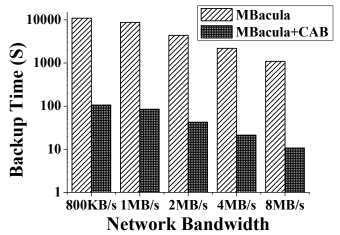
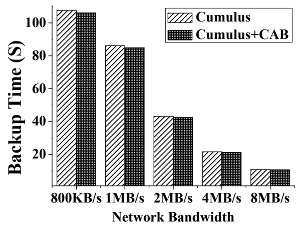


Figure 7: The comparison of the cumulative data before and after CABdedupe during the 31 backups of one author's home directory.



(a) Backup times of MBacula and MBacula+CAB



(b) Backup times of Cumulus and Cumulus+CAB

Figure 8: The comparison of the backup times of MBacula and MBacula+CAB, Cumulus and Cmulus+CAB during the last backup of one author's home directory on the 31st day.

D. Discussions

File Monitor's Overheads. File Monitor starts when the system is bootstrapped and terminates when the system is shut down. In CABdedupe, by restricting the number of the processed file operations and keeping File Monitor idle during most of the time (see Section IV-A), we find that CABdedupe imposes only 1.45% of computation and memory overheads on average.

Storage Overheads. CABdedupe needs to store the causality information in File List and File Recipe Store in the client's machine for fast identifying and removing the unmodified data during each backup/restore operation. In our experiments where we protect 9.07GB data by 31 backups, File List and File Recipe Store occupy only 1.76MB and 47.01MB storage space respectively, amounting to about 0.52% of the 9.07GB protected data.

Restore Cases. The restoration cases can be classified into three categories, the full file-system restore, the single directory restore, and the individual file restore. To the first restore case when the file system is destroyed by site disasters or hardware failures, the whole file system must be retrieved in their entirety from the remote backup destination. But if the file system is corrupted by software failures or viruses attacks, there may be only a small portion of files or data chunks that need to be transmitted with the help of CABdedupe. While for the secondary restore case as long as the directory has not been entirely deleted, CABdedupe can fast find the unmodified files or data chunks to avoid their data transmission, which eases the searching burden of the users especially by facing the directories including tens of thousands of files or even more files. In the last restore case of the individual file restoration, the gain obtained by CABdedupe is limited by the size of the unmodified data, unless there is a very big file with only a few data chunks modified. In summary, CABdedupe has the advantages of fast identifying the unmodified data for restorations to reduce the restore times, which avoids the human operations and further eases the users' burden especially on the restoration of full file-systems and directories when the unmodified data is available and accessible.

Data Loss. There are many incidents causing data loss and triggering data restorations. The 2010 data loss survey [22] carried out by Cibecs Company classifies these incidents into seven categories, including theft, negligence, hardware failure, software failure, technical incompetence, viruses and others. Its survey report pointed out that "39% of data losses are ascribed to hardware and software failures, 34% of losses are attributed to negligence and theft, 23% of data losses are caused by viruses and technical incompetence" [22]. Observed from these incidents indicated by the report, we argue that the unmodified data can be accessible when the 23% of data loss is caused by viruses and technical incompetence [22] and even sometime when the 11% of data loss is caused by software failures [22]. Under such restore scenarios, it is possible for CABdedupe to remove the unmodified data from transmission to reduce the restore time. For other restore scenarios when all of the data in users' local computers is destroyed, CABdedupe has little benefit and all the restore data must be entirely transmitted. Nevertheless, CABdedupe is a very simple scheme running in the users' local computers and outperforms other deduplication approaches by combing the cloud backup and cloud restore optimizations together by sharing the same architecture. Furthermore, CABdedupe is not designed to replace the existing deduplication approaches in backup systems; it is only a middleware that is orthogonal to and can be integrated into any existing backup system (i.e., regardless whether exploiting the deduplication approach) to relieve the data transmission overheads . Its failure will not affect the normal backup and restore operations.

VI. Related work

One serious challenge cloud backup services face is the low bandwidth of WAN that connects end users and service providers, which lengthens both the backup time and restore time when a large amount of data is involved. Existing cloud backup systems such as Commvault [12], EMC Avamar [10], Cumulus [3], Asigra [11], and etc., have all adopted the source-side deduplication technology to remove the redundant data from transmission to improve backup performances. However, the problem encountered by data restore operations over the low bandwidth network has by and large been overlooked by these existing cloud backup systems. The existing cloud recovery solutions either build complete servers in the client site to transmit and restore the data locally over high bandwidth network [10], build virtual servers in service providers'site to shift the operations and businesses to provider's servers after disasters [11], or ship the data from service providers to users by vehicles avoiding Internet transmission [11]. All of these approaches cost much more financially than directly transmitting the data over Internet especially in some restore cases the restore time can be significantly reduced by removing the unmodified data like CABdedupe does.

A rich body of previous research has addressed the problem of data transmission over low-bandwidth networks in other applications. Rsync [23] is an early study that attempts to reduce the transmission of the redundant data that is already stored in the server. Unfortunately, it only focuses on the redundant data among files with identical file names, without concerning the redundant data across different files. Unlike Rsync, LBFS [6] exploits the data redundancy among all the files using a content defined chunking (CDC) algorithm, which removes all the redundant data at the chunk level to improve the network file system performances. TAPER [24] is a scalable data replication protocol that provides a four-phased redundancy elimination scheme to balance the tradeoff between network bandwidth savings and computation overheads. Besides these three methods, other approaches [25], [26] used in distributed file systems also address this transmission problem, such as the recipe technique used in [27], the lookaside caching technique used in [28], and etc.. They all try to fetch data from nearby providers or mobile devices locally, instead of retrieving from remote severs. However, they are all not designed specially for backup systems and have limited effectiveness in cloud backup service environment.

Most of the existing deduplication approaches employed in backup and archival systems heavily depend on a large chunk index stored in servers to find and locate the redundant data chunks [7], [9], [29]. Many researchers have reported that it incurs heavy overheads in terms of increased latency and reduced throughput and thus addressed it in many different ways, such as DDFS [8] and Sparse Index [30] exploiting the chunk locality, Extreme Binning [31] adopting the file similarity, and Cumulus [3] narrowing the search space of redundant data chunks. However, all of the deduplication researchers have not tried their best to address the network efficiency problem for data restore operations. To our best knowledge, our CABdedupe is the first deduplication scheme trying to address this problem from both angles of data backup and data restore over WAN in cloud backup environments. CABdedupe uses the causality information among dataset versions in the source client, instead of a large chunk index, to identify and remove the redundant data from transmission for not only backup but also restore operations. The causality information is widely used by file system designers for a variety of different purposes [32], [33]. For example, Taser [34] uses it to identify files tainted by intrusions, BackTracker [35] captures it to analyze intrusions, and [36] uses it to enhance personal search capability.

In addition to the deduplication technology employed in the cloud backup environment, the wide area data services (WDS), such as Riverbed [37], can also be leveraged to remove the redundant data from transmission over WAN to alleviate the network bottleneck. However, due to the fact that CABdedupe is specially designed for backup/restore workloads at the application level, we argue that CABdedupe is more effective than WDS-based approaches. Moreover, CABdedupe, as a middleware that is orthogonal to and can be integrated into any existing backup system, is also capable of accelerating the performance of any existing cloud backup system that is based on the WDS approaches.

VII. Conclusion

Based on the observations that both the backup time and restore time in the cloud backup environment are too high for small businesses and telecommuters alike due to the relatively low WAN bandwidth, and the existing solutions that use the deduplication technology only focus on the reduction of backup time while ignoring the restore time, we propose CABdedupe, a causality-based deduplication performance booster for both cloud backup and cloud restore operations. CABdedupe first captures and preserves the causal relationship among chronological versions of datasets using three key components, File Monitor, File List, and File Recipe, and then exploits this captured causality information to fast identify which files and data chunks have been changed or remained unchanged, thus enabling the removal of the unmodified data from transmission for each backup/restore operation to improve backup/restore performance. CABdedupe is a middleware that is orthogonal to and can be integrated into any existing backup system. Our extensive experiments with CABdedupe integrated into two existing backup systems and driven by real world datasets show that both the backup time and restore time are significantly reduced, with a reduction ratio of up to $103 : 1$.

Acknowledgements

This work was supported by the National Basic Research 973 Program of China under Grant No. 2011CB302300, the National High Technology Research and Development Program ("863" Program) of China under Grant No.2009AA01A401 and 2009AA01A402, Changjiang innovative group of Education of China No. IRT0725, National Natural Science Foundation of China (NSFC) under Grant 60703046, 60873028 and 60933002, and the US NSF under

Grants NSF-CNS-1016609, NSF-IIS-0916859, NSF-CCF-0937993 and NSF-CCF-0621526.

References

[1] J. Gahm and J. Mcknight, "Medium-size business server & storage priorities," *Enterprise Strategy Group*, Jun. 2008.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and I. Stoica, "Above the Clouds: A Berkeley View of Cloud Computing," UC Berkeley, Tech. Rep. 2009-28, Feb. 2009.

[3] M. Vrable, S. Savage, and G. M. Voelker, " Cumulus: Filesystem Backup to the Cloud," in *FAST'09*, Feb. 2009.

[4] "The case for managed backup services. Fuel revenue and growth." [Online]. Available: \url{http://asigra.com/pdf/thinkingofmanagedservices.pdf}

[5] "Data protection survey," *Enterprise Strategy Group*, Jan. 2008.

[6] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *SOSP'01*, Oct. 2001.

[7] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *FAST'02*, Jan. 2002.

[8] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the Data Domain deduplication file system," in *FAST'08*, Feb. 2008.

[9] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki, "Hydrastor: A scalable secondary storage." in *FAST'09*, Feb. 2009.

[10] EMC Avamar, "http://www.emc.com/avamar."

[11] Asigra Hybrid Cloud Backup and Recovery Software, "http://www.asigra.com."

[12] Commvault Simpana, "http://www.commvault.com."

[13] "Symantec's fifth annual IT Disaster Recovery survey," *Symantec*, June. 2009.

[14] "Branch Office Optimization," *Enterprise Strategy Group*, Jan. 2007.

[15] C. Policroniades and I. Pratt, "Alternatives for detecting redundancy in storage systems data," in *USENIX'04*, Jun. 2004.

[16] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "A Five-Year Study of File-System Metadata," in *FAST'07*, Feb. 2007.

[17] D. Roselli, J. R. Lorch, and T. E. Anderson, "A Comparison of File System Workloads," in *USENIX'00*, Jun. 2000.

[18] M. O. Rabin, "Fingerprinting by random polynomials," Center for Research in Computing Technology, Harvard University, Tech. Rep. TR-15-81, 1981.

[19] NIST, "Secure Hash Standard," in *FIPS PUB 180-1*, May 1993.

[20] Bacula, "http://www.bacula.org."

[21] Amazon Simple Storage Service, "http://aws.amazon.com/s3."

[22] 2010 Data Loss Survey, "http://www.idgconnect.com/idgconnect/view_abstract/4605/connect/it-systems-management/security/enterprise-data-protection/2010-data-loss-survey."

[23] Rsync, "http://samba.anu.edu.au/rsync/."

[24] N. Jain, M. Dahlin, and R. Tewari, "TAPER: Tiered approach for eliminating redundancy in replica synchronization," in *FAST'05*, Dec. 2005.

[25] S. Annapureddy, M. J. Freedman, and D. Mazieres, "Shark: Scaling File Servers via Cooperative Caching," in *NSDI'05*, May. 2005.

[26] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil, "An Architecture for Internet Data Transfer," in *NSDI'06*, May. 2006.

[27] N. Tolia, M. Kozuch, M. Satyanarayanan, and B. Karp, " Opportunistic Use of Content Addressable Storage for Distributed File Systems," in *USENIX'03*, Jun. 2003.

[28] N. Tolia, J. Harkes, M. Kozuch, and M. Satyanarayanan, "Integrating Portable and Distributed Storage," in *FAST'04*, Mar. 2004.

[29] S. Rhea, R. Cox, and A. Pesterev, "Fast, inexpensive content-addressed storage in Foundation," in *USENIX'08*, Jun. 2008.

[30] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Campbell, "Sparse Indexing: Large scale, inline deduplication using sampling and locality," in *FAST'09*, Feb. 2009.

[31] D. Bhagwat, K. Eshghi, D. D. Long, and M. Lillibridge, "Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup," HP Laboratories, Tech. Rep. HPL-2009-10R2, Sep. 2009.

[32] K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *USENIX'06*, May. 2006.

[33] K. Muniswamy-Reddy and D. A. Holland, "Causality-Based Versioning," in *FAST'09*, Feb. 2009.

[34] A. Goel, K. Po, K. Farhadi, Z. Li, and E. de Lara, "The Taser intrusion recovery system," in *SOSP'05*, Oct. 2005.

[35] S. T. King and P. M. Chen, "Backtracking Intrusions," in *SOSP'03*, Oct. 2003.

[36] S. Shah, C. A. N. Soules, G. R. Ganger, and B. D. Noble, "Using provenance to aid in personal file search," in *USENIX'07*, Jun. 2007.

[37] Riverbed, "http://www.riverbed.com."