J Sign Process Syst (2013) 72:209–228
DOI 10.1007/s11265-013-0775-x

SAFE: A Source Deduplication Framework for Efficient Cloud Backup Services

Yujuan Tan · Hong Jiang · Edwin Hsing-Mean Sha · Zhichao Yan · Dan Feng

Received: 6 November 2012 / Revised: 30 March 2013 / Accepted: 10 May 2013 / Published online: 21 June 2013

Abstract Due to the relatively low bandwidth of WAN that supports cloud backup services and the increasing amount of backed-up data stored at service providers, the deduplication scheme used in the cloud backup environment must remove the redundant data for backup operations to reduce backup times and storage costs and for restore operations to reduce restore times. In this paper, we propose SAFE, a source deduplication framework for efficient cloud backup and restore operations. SAFE consists of three salient features, (1) Hybrid Deduplication, combining the global file-level and local chunk-level deduplication to achieve an optimal tradeoff between the deduplication efficiency and overhead to achieve a short backup time; (2) Semantic-aware Elimination, exploiting file semantics to narrow the search space for the redundant data in hybrid deduplication process to reduce the deduplication overhead; and (3) Unmodified Data Removal, removing the files and data chunks that are kept intact from data transmission for some restore operations. Through extensive experiments driven by real-world datasets, the SAFE framework is shown to maintain a much higher deduplication efficiency/overhead ratio than existing solutions, shortening the backup time by an average of 38.7 %, and reduce the restore time by a ratio of up to 9.7 : 1.

Keywords Cloud backup services · Data backup · Data restore · Data deduplication

Y. Tan (✉) · E. H.-M. Sha
College of Computer Science,
Chongqing University, Chongqing, China
e-mail: tanyujuan@gmail.com

E. H.-M. Sha
e-mail: edwinsha@gmail.com

H. Jiang · Z. Yan
Department of Computer Science & Engineering,
University of Nebraska-Lincoln, Lincoln, NE, USA

H. Jiang
e-mail: jiang@cse.unl.edu

Z. Yan
e-mail: yanzhichao.hust@gmail.com

D. Feng
School of Computer Science & Technology,
Huazhong University of Science & Technology, Wuhan, China
e-mail: dfeng@hust.edu.cn

1 Introduction

Driven by the trend toward cloud computing, backup, which consumes significant IT resources and can be relatively independently integrated, is an attractive application to be outsourced to cloud, an effort referred to as *Cloud Backup Service*. Cloud backup service has been viewed favorably by telecommuting employees, Remote Offices/Branch Offices (ROBOs), and Small and Medium Businesses (SMBs) who lack sufficient remote backup strategies due to the limited IT staffs and constrained IT budgets.

However, cloud backup services face some serious challenges. The first one is the long backup/restore time that represents the time spent on sending/restoring specific dataset to/from backup destination, due to the low bandwidth WAN links between the user and service provider constraining the data transmission. For example, it would take more than 14 days to backup/restore 1TB data to/from Amazon S3 [5] with an assumed network bandwidth of 800 KB/s [46]. The ESG research indicates that 39 % of organizations that have

tried to run backups over WAN report both the backups and restores take too long and 31 % of them report that the cost of WAN bandwidth is too high [1]. Another challenge stems from the vast storage space and very high data management cost required for the rapidly increasing amount of backed-up data stored at service providers' site. Thus both challenges demand a novel space-efficient approach that can reduce both the bandwidth consumptions and storage costs for cloud backup services.

Two well-known source deduplication methods, *source local chunk-level deduplication* [39, 46] and *source global chunk-level deduplication* [11, 15, 29], have been proposed in the past for backup operations to reduce the backup times and save storage costs by removing the redundant data chunks from being transmitted to backup destination. The former removes the duplicate chunks locally within the same client while the latter removes the duplicate chunks globally across different clients. But as revealed by some recent studies [12, 34, 52], due to the out-of-memory fingerprint accesses to massive backed-up data, chunk-level deduplication has an inherent latency and throughput problem that significantly affects the backup performances. In source global chunk-level deduplication, this overhead of massive disk accesses will obviously throttle the deduplication process and hence lengthen the backup time. In source local chunk-level deduplication, the overhead is much alleviated since searching the duplicate chunks is restricted to the same client and the number of on-disk searched fingerprints is significantly reduced. This reduced overhead, however, comes at the cost of a severely limited data compression ratio, which lengthens the backup time and increases the storage cost due to the increased data transmission. Therefore it is desirable to *achieve an optimal tradeoff between the deduplication efficiency and deduplication overhead to maintain a much shorter backup time* than existing solutions for cloud backup operations.

Our preliminary studies obtained two important and insightful observations on the redundancy of typical backup data. First, it is observed that many file-semantic attributes(see Section 2.1), such as the file locality, file size, file type and file timestamps, have an indication of the existence or nonexistence of the redundant data. By exploiting these file semantics to narrow the search space for the redundant data, it is possible to fast find them in a very short time and thus to reduce the overall deduplication overhead. Second, it is observed that the redundant data across different clients is dominated largely by duplicate files(see Section 2.2). This observation, combined with the fact that the removal of duplicate files is much easier than that of duplicate chunks, suggests that a global file-level deduplication will likely achieve a higher efficiency/overhead ratio than a global chunk-level deduplication across different clients. Furthermore, by combining the global file-level deduplication and the local chunk-level deduplication, it is possible to achieve a higher compression ratio than the local chunk-level deduplication and a lower overhead than the global chunk-level deduplication. Thus the exploitation of file semantics integrated with a hybrid deduplication method has potential benefits in achieving a high deduplication efficiency/overhead ratio to maintain a short backup time for cloud backup operations.

Besides the two important observations described above, our preliminary studies have also observed some data redundancy for data restores (see Section 2.3). Each restore operation takes place after data corruptions and needs to restore the corrupted dataset to a previous backed-up version that has been stored in the backup destination. This corrupted dataset, as a general rule, is anyways evolved from its previous backed-up versions with relatively data modifications, insertions or deletions, and further in some restore scenarios (see Section 2.3.2), this evolution may have many files and data chunks unchanged in their entirety after backups. Thus by exploiting this data redundancy and removing those unchanged files and data chunks from data transmission, it is possible to significantly reduce the restore times for cloud restore operations.

The above observations and analysis motivate us to propose SAFE, a source deduplication framework for cloud backup services. SAFE achieves an optimal tradeoff between the deduplication efficiency and deduplication overhead for backup operations to achieve a short backup time, and also exploits the data redundancy for some restore operations to reduce the restore times. SAFE has three salient features that distinguish it from the existing solutions: (1)*Semantic-aware Elimination*, exploiting the file semantics such as the file locality, file types, file sizes, and file timestamps to narrow the search space for the redundant data to reduce the deduplication overhead; (2)*Hybrid Deduplication*, combining the Global File-level Deduplication (GFD) that removes the global duplicate files across different clients and Local Chunk-Level Deduplication (LCD) that removes the local duplicate chunks across similar files in the same client to achieve a high deduplication efficiency/overhead ratio for cloud backup operations; and (3)*Unmodified Data Removal*, removing the unmodified files and data chunks that are kept intact after backups from data transmission for cloud restore operations. Among these features, Hybrid Deduplication and Semantic-aware Elimination work in synch to remove the redundant data from data transmission to reduce backup times and storage costs, while Unmodified Data Removal aims to reduce the restore times in some restore scenarios. Through extensive experiments driven by real-world datasets, the SAFE framework is shown to maintain a much higher deduplication efficiency/overhead ratio than existing solutions, shortening the

backup time by an average of 38.7 %, and reduce the restore time by a ratio of up to 9.7 : 1.

It must be noted that our SAFE is mainly oriented toward the cloud backup of unstructured data. The unstructured data, as opposed to its structured counterpart typically found in databases, has no fixed data model and is created by individuals on a variety of systems using a variety of formats (documents, photos, videos, etc.) [45]. It comprises the vast majority of users information in modern-day organizations, estimated to be as high as 80 %, and this ratio is projected to further increase [20]. Therefore, the current form of SAFE is limited to file systems that consist of mostly unstructured data, to the exclusion of database systems.

The rest of the paper is organized as follows. In Section 2 we discuss the motivation for our research. We present the system architecture in Section 3 and the detailed SAFE framework in Section 4. Section 5 evaluates SAFE through experiments driven by real-world datasets. Section 6 presents related work and Section 7 concludes the paper.

2 Motivation of Our Work

Arguably, neither of the two well-known deduplication schemes mentioned in the previous section, namely, source global chunk-level deduplication and source local chunk-level deduplication, is effective or desirable for cloud backup services that demand short backup time, short restore time and high storage space-efficiency. In this section, we will discuss the data redundancy in cloud backup environment to motivate our SAFE research.

2.1 File Semantics Exploration

File semantics can be revealed at various levels, including definitional, associative, structural, behavioral, environmental level or through other relevant information of the files, where various hints can be obtained. In this subsection, we will focus on the file locality and some other file attributes to discuss the inherent data redundancy revealed by them.

File Locality There are many identical directories created by individual users in enterprise file systems [19], such as software packages, copies of repositories, directories of photos or music. This suggests that if two directories share one file, other files may also be shared, a common form of file duplicate locality. Thus by exploiting this file locality, it is possible to prefetch and cache the nearby duplicate files(i.e., the duplicate file hashes in SAFE) to memory to avoid the disk accesses for them when one is found, so as to reduce the overall deduplication overhead.

File Size It is a known fact that most files are small files in a typical file system [4]. During our experiments (detailed in Section 2.2), we observed that about 63.8 % of the identical files are smaller than 8 KB, accounting for only 0.53 % of the redundant data, which is consistent with previously published studies [4]. Thus small files can be ignored to improve the deduplication efficiency/overhead ratio during file-level deduplication process.

File Type Some specific files, such as compressed tarballs, multimedia files, that are semantically identical may share little redundant data in their binary presentations. For example, the same picture stored in the GIF format is completely different from that stored in the JPEG format. For these file types, very few similar files exist and thus exploiting their data redundancy at the file level instead of chunk level will likely be sufficient [26].

File Timestamps The timestamps of a file are changed whenever the file is modified. Thus, simply checking file timestamps(i.e., checking the file timestamps at user client site in SAFE) can directly identify and remove unchanged files from the backup process. There is no need to involve such unchanged files in either file-level deduplication or chunk-level deduplication.

The file semantics described above present the potential opportunity to narrow the search space for redundant data to reduce the deduplication overhead while still maintaining high deduplication efficiency, and thus they are exploited by SAFE to accelerate the deduplication process in backup operations. Furthermore, we believe that other file semantics exposed at various levels of the file system can also be helpful in optimizing deduplication performances, which is beyond the scope of this paper and will be investigated in our future work.

2.2 Hybrid Approaches for Data Backups

Both the intuitive scenarios statistical evidences reveal the benefits of exploring a hybrid deduplication method in cloud backup environment.

2.2.1 Intuitive Scenarios

Large amounts of redundant data from identical files.

- In each classical full backup, all the files, regardless of their change status, are sent to the backup destination. As a result, many duplicate files are repeatedly backed up among multiple full backups.
- Routine file operations tend to generate many identical files entering cloud backup systems through routine file backups, such as file replications (e.g., photos, reports,

etc.) for data sharing among friends or colleagues, downloading the same files from Internet by different users, installing the same operating-system files[–] or applications across different file systems, etc. More importantly, those files are rarely changed.
– Microsoft's study [27] reveals that 75 % of the redundant data are generated from the duplicate files after the survey of the file system content data collected from 857 desktop computers.

The above scenarios indicate that substantial identical files exist in backup systems. These files stand to be identified and removed at the file level to filter out data redundancy before resorting to chunk-level deduplication, thus significantly reducing the chunking and hence the deduplication overhead. It is beneficial to combine file-level and chunk-level deduplication strategies in backup systems.

2.2.2 Statistical Evidences

To verify the data redundancy observed from our intuitive scenarios, we have obtained statistical and quantitative evidences through a file-scanning backup program (FSBP) that we developed. FSBP simulates a file-system backup software that extracts file hashes and chunk fingerprints (chunk hashes) from the backed-up datasets but without sending them to the backup destination. We have distributed FSBP to five members under the same project in our research group and they do semi-regular full backups of their personal home directories(i.e, including course files, project documents, source trees, developing tools and other private files but without the operating system files) lasting for 6 days, totaling about 19 full backups. The total amount of backed-up data is about 256 GB consisting of 958, 641 files. Table 1 shows the data redundancy quantitatively. In what follows we will present three observations drawn from these statistics.

– Most of the redundant data can be attributed to identical files during filesystem backups, of which a vast majority is generated from multiple full backups of the same client, reinforcing the critical importance of Virtual Full Backup(see Section 4.1.1) in cloud backup systems such as SAFE.
– The redundant data across different clients is dominated largely by duplicate files, which is consistent with the latest research result from Microsoft [27]. This implies that the file-level removal of the duplicate files among different file systems alone is sufficient to eliminate the vast majority of such redundant data.
– Besides identical files, there are many similar files existing in the same client. The total amount of the redundant chunks shared by them is comparable to that of duplicate files across different clients, implying that the local chunk-level deduplication is important and essential to maintain a high compression ratio.

The above observations clearly suggest that the vast majority of redundant data stems from duplicate files, especially for the data redundancy across different clients. This, combined with the fact that removing duplicate files at file level is easier and more efficient than that at chunk level, motivates us to propose SAFE that exploits global file-level deduplication and local chunk-level deduplication together to achieve an optimal tradeoff between the deduplication efficiency and overhead for backup operations to maintain short backup time.

2.2.3 Theoretical Analysis

Besides the statistical evidences, we will further analyze the effectiveness of the hybrid approach proposed by SAFE that combines the global file-level and local chunk-level deduplication approaches, by comparing it to the existing source global chunk-level deduplication approach and source local chunk-level deduplication approach. This analysis is based on the assumption that there is N duplicate files and M duplicate chunks (not including the duplicate chunks from duplicate files) across different clients, and L duplicate chunks from the same client. Each of the duplicate file is composed of $K (K > 1)$ chunks in average. As a side note, the duplicate files generated from the same client by full backups are not considered here due to that the Virtual Full Backup method can remove these duplicate files which will not be involved to general deduplication processes. $Cost_{gf}$, $Cost_{gc}$, $Cost_{lc}$ are used to represent the deduplication overhead of each duplicate file in the global file-level deduplication approach, the deduplication overhead of each duplicate chunk in the global chunk-level deduplication approach, and the deduplication overhead of each duplicate chunk in the local chunk-level deduplication

Table 1 The data redundancy in preliminary study.

	Total	Duplicate files	Duplicate chunks
The total redundant data	211 GB (100 %)	199.2 GB (94.4 %)	11.8 GB (5.6 %)
Within the same client	199.6 GB (100 %)	188 GB (94.2 %)	11.6 GB (5.8 %)
Across different clients	11.4 GB (100 %)	11.2 GB (98.2 %)	0.2 GB (1.8 %)

approach respectively, with the assumption that $Cost_{gc} = \alpha \cdot Cost_{gf} (\alpha > 1)$ and $Cost_{gc} = \beta \cdot Cost_{lc} (\beta > 1)$.

Therefore, the overall deduplication overhead $Cost_{slc}$, $Cost_{sgc}$, $Cost_{hyb}$ and the number of removed chunks $Remove_{slc}$, $Remove_{sgc}$, $Remove_{hyb}$ in the source local chunk-level deduplication approach, source global chunk-level deduplication approach, and hybrid deduplication approach can be respectively computed as

$$\begin{cases} Remove_{slc} = L \\ Cost_{slc} = L \cdot Cost_{lc} \end{cases} \tag{1}$$

$$\begin{cases} Remove_{sgc} = N \cdot K + M + L \\ Cost_{sgc} = (N \cdot K + M + L) \cdot Cost_{gc} \end{cases} \tag{2}$$

$$\begin{cases} Remove_{hyb} = N \cdot K + L \\ Cost_{hyb} = N \cdot Cost_{gf} + L \cdot Cost_{lc} \end{cases} \tag{3}$$

and their ratios between the deduplication overhead and the number of removed chunk can be computed by

$$\begin{cases} \dfrac{Remove_{slc}}{Cost_{slc}} = \dfrac{L}{L \cdot Cost_{lc}} = \dfrac{1}{Cost_{lc}} \\ \dfrac{Remove_{sgc}}{Cost_{sgc}} = \dfrac{N \cdot K + M + L}{(N \cdot K + M + L) \cdot Cost_{gc}} = \dfrac{1}{Cost_{gc}} \\ \dfrac{Remove_{hyb}}{Cost_{hyb}} = \dfrac{N \cdot K + L}{N \cdot Cost_{gf} + L \cdot Cost_{lc}} \\ \qquad = \dfrac{N \cdot K + L}{\left(\frac{N}{\alpha} + \frac{L}{\beta}\right) \cdot Cost_{gc}} \\ \qquad = \dfrac{N \cdot K + L}{\left(N \cdot \frac{\beta}{\alpha} + L\right) \cdot Cost_{lc}} \end{cases} \tag{4}$$

As seen from the above formula, it is found that

$$\begin{cases} \dfrac{Remove_{hyb}}{Cost_{hyb}} = \dfrac{N \cdot K + L}{\left(\frac{N}{\alpha} + \frac{L}{\beta}\right) \cdot Cost_{gc}} > \dfrac{Remove_{sgc}}{Cost_{sgc}} \\ \qquad = \dfrac{1}{Cost_{gc}} \end{cases} \tag{5}$$

where $\alpha > 1, \beta > 1$. This reveals that the hybrid approach has better tradeoff between the deduplication efficiency and overhead than that for source global chunk-level deduplication approach. Moreover, the deduplication overhead of hybrid approach $Cost_{hyb} = N \cdot Cost_{gf} + L \cdot Cost_{lc} = \left(\frac{N}{\alpha} + \frac{L}{\beta}\right) \cdot Cost_{gc}$ is much less than that for source global chunk-level deduplication approach $Cost_{sgc} = (N \cdot K + M + L) \cdot Cost_{gc}$, while omitting only M duplicate chunks that have been verified very small according to our study and other literature.

While compared to the source local chunk-level deduplication approach,

$$\begin{cases} \dfrac{Remove_{hyb}}{Cost_{hyb}} = \dfrac{N \cdot K + L}{\left(N \cdot \frac{\beta}{\alpha} + L\right) \cdot Cost_{lc}} > \dfrac{Remove_{slc}}{Cost_{slc}} \\ \qquad = \dfrac{1}{Cost_{lc}} \end{cases} \tag{6}$$

when $K > \frac{\beta}{\alpha}$, meaning that the hybrid approach has better tradeoff between the deduplication efficiency and overhead than that for source local chunk-level deduplication approach. But when $K < \frac{\beta}{\alpha}$, $\frac{Remove_{hyb}}{Cost_{hyb}} < \frac{Remove_{slc}}{Cost_{slc}}$. However, this source local chunk-level deduplication approach doesn't remove the $N \cdot K + M$ duplicate chunks across clients, and these duplicate chunks, according to the experimental results from our study and other literatures, takes large percentage of the total redundant data that must be removed. In summary, the hybrid approach outperforms the existing source global chunk-level and local chunk-level deduplication approaches by combing both of the deduplication overhead and efficiency, which is more suitable for the redundancy elimination in cloud backup services.

2.3 Removing Unmodified Data for Restores

Most of the existing deduplication approaches mainly focus on removing the redundant data for backup operations, while paying little attention to the data restores in cloud backup environment. However, during each restore operation, the corrupted dataset that requires to restore in users' local computers is also evolved from its previous backed-up versions with data modifications(like that in backups), and thus many redundant data exists among multiple restore/backup operations and can be removed to improve restore performances. In this subsection, we will discuss the existence of the redundant data between the corrupted version of the dataset and its previous backed-up versions and present the potential opportunity for exploiting the data redundancy to reduce the restore time.

2.3.1 Data Redundancy

During some of the restore operations, there are many unchanged files and data chunks that are kept intact after their previous backups. In the following, we will present three types of files that intuitively describe the file relationships between the current corrupted version of the dataset and its previous backed-up versions to show the data redundancy.

Unchanged Files Most files are kept unchanged after their first backups. Policroniades et al. [31] noted that in real file

systems, most accesses to files are read-only, implying that most files are not modified since they are initially created or copied onto the file system. Microsoft's 5-years file metadata study [4] further shows that the percentage of these unmodified files has grown from 66 % to 76 % from 2000 to 2004. A good example of the data corruptions is the 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 Window XP operating systems. When one file system is attacked by this kind of viruses, there will be only a limited number of executable files infected and large amounts of non-executable files are kept intact.

Modified Files Generally, the individual files that require restorations all have been modified since their last backups. Nevertheless, a significant amount of unmodified data chunks are likely to exist between the current file version and its previous ones given that most data writes are concentrated on a small subset of data blocks in a short time in typical file systems [36].

Deleted Files Besides the unchanged and modified files, typically there are many files that have been deleted or entirely corrupted after their last backups. These deleted files, however, must be transmitted in their entirety from the remote backup destination to the users' local computers for restorations.

The above three types of files reveal that a large amount of unmodified data (i.e., unchanged files and unmodified data chunks) exists between the current corrupted version of the dataset and its previous backed-up versions after data corruptions during some restores. If such unmodified data in users' local computers is available (i.e., can be accessed), it is possible to remove them from transmission to significantly reduce the restore time.

2.3.2 Restore Scenarios

Many incidents can cause the data loss or disasters that triggers the data restorations. In some of the data restores, the unmodified data that has been described above exists with a high probability. The 2010 data loss survey [2] 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". Observed from these incidents presented in the report, we argue that the unmodified data is available when the 23 % of data loss is caused by viruses and technical incompetence and even sometime when the 11 % of the data losses are caused by software failures, since these incidents will not completely destroy the whole file system data and much chunk data will be kept intact in client computers. Therefore, under these restore scenarios, it is possible to prevent this unmodified data from being transmitted from remote backup destination to reduce the restore time, which motivates us to propose SAFE to exploit this data redundancy to improve restore performance. But in other restore scenarios when the unmodified data is unavailable, all the restore data must be entirely transmitted from the remote backup destination and thus the restore time cannot be reduced through the reduction of the transferred dataset.

3 System Architecture

To present SAFE clearly, we first describe the system architecture briefly. As shown in Fig. 1, the system is composed of three subsystems: File Agent, Master Server and Storage Server. File Agent is distributed and installed on user client machines that subscribe to backup services, while Master Server and Storage Server are located in remote datacenters to provide backup services.

3.1 File Agent

File Agent is a software program that provides a functional interface (file backups/restores) to users. It is responsible for gathering datasets and sending/restoring them to/from Storage Servers for backups/restores. To apply

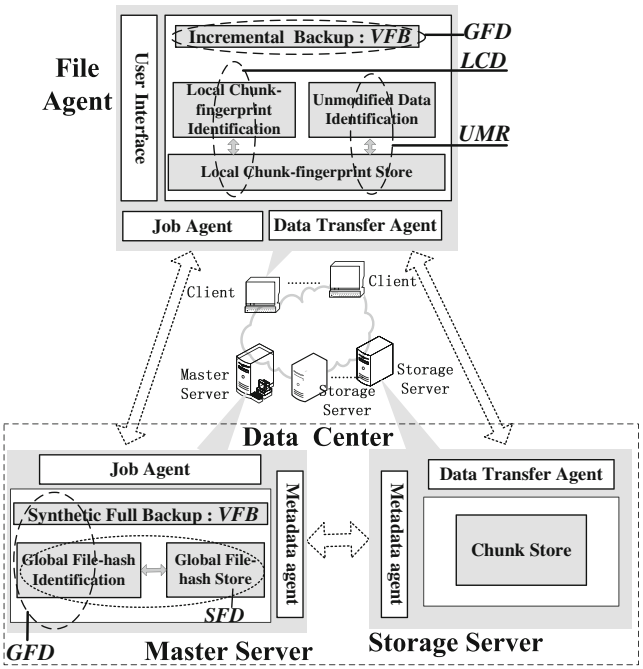

Figure 1 System architecture.

SAFE to the system, File Agent adds three key functional modules, Incremental Backup (Incremental Backup is the Phase-I of the two-phased Virtual Full Backup (VFB) module and Virtual Full Backup is the Phase-I of the two-phased Global File-level Deduplication (GFD) module), Local Chunk-level Deduplication (LCD) which consists of Local Chunk-fingerprint Identification and Local Chunk-fingerprint Store, and Unmodified Data Removal (UMR) which consists of Unmodified Data Identification and Local Chunk-fingerprint Store.

3.2 Master Server

Master Server is the administrative center in the whole system, which globally manages and schedules all backup and restore jobs. It uses a catalog database to keep track of which files are stored on which Storage Servers. To apply SAFE, two key functional modules are added to Master Server, Synthetic Full Backup (Phase-II of VFB) and Server-side File-level Deduplication (SFD, Phase-II of the GFD module) which consists of Global File-hash Identification and Global File-hash Store.

3.3 Storage Server

Storage Server is the depository for backed-up data. It performs actual data backup and restore jobs along with File Agent under the direction of Master Server. After each backup job, Storage Server sends the metadata information (including file metadata and job metadata) to Master Server that in turn stores it in catalog database for backed-up data indexing and retrieving. Chunk Store that consists of a large number of chunk containers is responsible for storing the backed-up data chunks from different user clients.

3.4 SAFE Process Flows

SAFE has two separate process flows, one for data backup and the other for data restore. In this subsection, we describe these two process flows to illustrate the functionalities of SAFE.

3.4.1 Backup Process flow

SAFE's data backups take the following three stages involving the semantic-aware elimination and the hybrid deduplication.

Stage 1 During each backup, SAFE first uses Virtual Full Backup (detailed in Section 4.1.1), the first phase of the Global File-level Deduplication (GFD), to filter out the completely unchanged files according to their file timestamps in File Agent.

Stage 2 After the processing of Virtual Full Backup, SAFE uses Server-side File-level Deduplication, the second phase of GFD, to remove the duplicate large files across different clients in Master Server. SAFE in this stage excludes small files to narrow the search space for duplicate files, and further exploits the file locality to reduce the deduplication overhead.

Stage 3 After the Global File-level Deduplication process, SAFE uses the Local Chunk-level Deduplication to remove the duplicate chunks across similar non-compressed files within the same Client. This process is carried out by File Agent locally instead of Master Server remotely so as to reduce the data transmission overhead and relieve the server workload.

After the above tri-staged deduplication process, the remained files and data chunks are considered new and directly sent to the corresponding Storage Servers under the direction of Master Server.

3.4.2 Restore Process Flow

In some restore scenarios when the local unmodified data is available, SAFE's data restores are accomplished with the help of the Unmodified Data Removal (UMR) module in File Agent, which assists in identifying and removing the unmodified data chunks from transmission by the following three stages. A more detailed description is showed in Section 4.2.

Stage 1 During each restore operation, UMR first finds the same files(the files with the same file names) that exist in both the corrupted version of the dataset and its previous backed-up version (the backed-up version that the corrupted dataset requires restored to).

Stage 2 After finding out all the same files, UMR then gets the corresponding chunk fingerprints of those files in both the corrupted version and backed-up version.

Stage 3 Having all the chunk fingerprints after stage 2, UMR finds the unmodified data chunks(i.e., by identifying those chunk fingerprints that exist in both the backed-up version and corrupted version) and removes them from transmission to reduce the restore time.

After the above three stages, the remained data chunks that have been modified are notified to Master Server that enables them sent from the corresponding Storage Servers. However, if there are no unmodified data chunks or the unmodified data chunks are unavailable, all the data chunks

must be sent from the corresponding Storage Servers in their entirety without the above three stages.

4 Design and Implementation

In this section, we will detail the designs and implementations of the three unique design features of SAFE, semantic-aware elimination, hybrid deduplication, and unmodified data removal.

4.1 Semantic-Aware and Hybrid Deduplication

In SAFE, the semantic-aware elimination and hybrid deduplication functions work in synch to remove the redundant data from transmission for backup operations. We detail their designs by describing the Global File-level Deduplication module and Local Chunk-level Deduplication module that embed their functionalities.

4.1.1 Global File-Level Deduplication

Global File-level Deduplication (GFD) is a two-staged deduplication approach, including Virtual Full Backup and Server-side File-level Deduplication, focusing on removing the duplicate file globally that have been backed up already.

Virtual Full Backup The Virtual Full Backup (VFB) module is the first stage of Global File-level Deduplication. It entails a two-phased backup operation, namely, a conventional incremental backup based on file timestamps in File Agent and a synthetic full backup following each incremental backup in Master Server, which work in synch to remove the vast majority of duplicate files without degrading the restore performance. The key to VFB's unique advantage lies in the fact that its first phase, incremental backup based on file timestamps that incurs nearly no deduplication overhead, blocks the bulk of completely unchanged files at the client site that would otherwise be repeatedly backed up in multiple full backups. And its second phase, synthetic full backup, helps to achieve a comparable restore performance to the conventional full backup. The concept of VFB was first introduced for storage optimization by ADSM [10], and it has been widely recognized in backup industries.

Server-Side File-Level Deduplication Server-side File-level Deduplication (SFD), the second stage of the Global File-level Deduplication approach, is implemented in Master Server. It compares the file hashes of incoming files (file hashes are generated by File Agent and sent to Master Server) with those already stored in the global file-hash store to identify and remove duplicate files (the files with the same file hashes) globally. However, SFD can become a potential performance bottleneck because the limited main memory capacity may force significant disk accesses for on-disk file hashes. SAFE in this stage exploits two important file semantics, namely, file size and file locality, to relieve this disk bottleneck as follows.

File Size SAFE excludes small files from SFD process to narrow the search space for duplicate files. Regardless of its size, each file has a single file hash that uniquely represents its file content. Any two files with the same file hash are regarded as duplicates and are stored only once. This implies that, when exploiting file hashes in file-level deduplication, files will be identified (except for data reading and file hash computing that are done in File Agent) at the same cost, independent of their sizes. This further suggests that, in terms of identification cost per MB, identifying large files will be far more cost-effective than small files at the file level. As shown in our statistical evidences, about 63.8 % of duplicate files are small files (smaller than 8 KB) accounting for only 0.53 % of redundant data (see Section 2.1). By ignoring these small duplicate files, we can reduce the number of file hashes by 63.8 % and arguably also reduce the number of disk accesses significantly, at the very low expense of leaving 0.53 % of redundant data unremoved. Simultaneously, the exclusion of small files from SFD can reduce the number of file hashes transferred from File Agent. If all the files are small, this stage can be skipped entirely, avoiding the round-trip between File Agent and Master Server altogether. Therefore, the exclusion of small files has the potential benefit of reducing disk accesses and data transmission overheads.

File Locality Given the existence of the file locality widely known and further confirmed in our study, SAFE uses three key techniques, namely, file vector, stream-informed file layout and locality-preserved cache [52], to identify and remove the duplicate file hashes, which significantly reduces the number of disk accesses. 1) File vector is implemented by Bloom Filter [8] that concisely represents the already-stored file hashes to support approximate membership queries with a small probability of false positive, whose use aims to avoid the disk accesses to nonexistent file hashes. 2) Stream-informed file layout is used to create file-hash locality in the global file-hash store. Upon each backup, File Agent reads files from a specified directory and sends their hashes in the same order as they are read to Master Server that in turn stores these file hashes to the global file-hash store in the same order as they are received, thus preserving the internal file locality. 3) Locality-preserved cache is used for caching the duplicate file hashes to avoid the disk accesses for them. As implied in Section 2.1, if one file is a duplicate, the nearby files are duplicate ones with a high probability. Thus SAFE uses locality-preserved cache

to prefetch and cache these nearby duplicate file hashes (these file hashes are stored in global file-hash store on disk) in memory to avoid the disk accesses for them when one is found. In summary, the combination of these three techniques can reduce the disk accesses for both duplicate file hashes and nonexistent file hashes, which effectively accelerates the SFD process.

4.1.2 Local Chunk-Level Deduplication

Local Chunk-level Deduplication (LCD) exploits the data redundancy within the same client. After global file-level deduplication, SAFE breaks the remained files into a series of variable sized chunks and determines whether they are duplicate ones by inspecting the local chunk-fingerprint store (As a side note, the local chunk-fingerprint store can be rebuilt from Master Server if File Agent fails) in File Agent. In the absence of sufficient main memory as that in SFD process, SAFE uses the following two approaches to reduce the disk accesses in LCD process.

Compressed-Files Exclusion SAFE excludes compressed files from LCD process. In the digital universe, there are many duplicate compressed files, especially digital images, audio and video files [20]. Two common features shared by these compressed files are their large size and high probability of remaining unmodified once generated. Even if such a compressed file is indeed modified by some special tools, there will be nearly no duplicate chunks between the original and the modified copy in their binary representations. Our experiments conducted to obtain statistical evidences show that no duplicate chunks are found among multiple highly semantically similar pictures stored in the JPEG file format, suggesting that there is no need to deduplicate such compressed files at the chunk level. Thus, SAFE excludes compressed files from LCD to narrow the search space for duplicate chunks to reduce the disk accesses while still maintaining high deduplication efficiency.

Two-Tiered Chunk Indexing and Small Files Optimization Inspired by the novel Extreme Binning [7] approach, SAFE uses the same two-tiered chunk indexing mechanism to reduce the disk accesses, where for each file a representative chunk fingerprint is selected as the primary index and the collection of chunk fingerprints of this file is binned as the secondary index. During LCD process, the primary index is fetched in its entirety to the memory and kept resident there since it is very small. If two files have the same primary index, they are considered highly similar and their secondary indices in disks are checked to find duplicate chunks. Thus, each file needs at most one disk access to identify its duplicate chunks. Distinct from Extreme Binning, SAFE further reduces the disk accesses for small files by piggybacking their secondary indices to primary indices when reading the latter to memory. Thus when identifying duplicate chunks among similar small files, no disk access is needed. In the current SAFE design, it is empirically determined that any file of no greater than two chunks is considered a small file, regardless of the chunk size.

4.2 Unmodified Data Removal

The Unmodified Data Removal (UMR) module focuses on identifying and removing the unmodified data from transmission for restore operations in File Agent. As described in Section 2.3, there are many unmodified files and data chunks that are kept intact in the corrupted dataset and require no data transmission in some restore scenarios. In this subsection, we describe how UMR identifies and removes such unmodified data from data transmission to reduce the restore time.

During each restore operation after selecting out the backed-up version that the corrupted dataset requires restored to, UMR runs the following three steps to filter out the unmodified data from restoration. In the first step UMR finds the same files (the files with the same file names) that exist in both the corrupted version of the dataset and its previously backed-up version (the backed-up version that the corrupted dataset requires restored to). It first consults the local chunk-fingerprint store (i.e., including the chunk fingerprints of backed-up files and the associated file metadata information such as file name, file backup time, and etc.) and finds all the files that exist in the backed-up version, and then it uses these obtained file names to find out the same files currently existing in the corrupted version in users' local computers.

After finding out all the same files, UMR in the second step retrieves their corresponding chunk fingerprints in both the corrupted version and its backed-up version. For the corrupted version, UMR gets their chunk fingerprint by chunking those files with the Rabin Fingerprints algorithm [33] and names each chunk with the SHA-1 hash function [30] in the same way as that in data backup operation. While for the backed-up version, UMR directly consults the local chunk-fingerprint store to retrieve the corresponding chunk fingerprints of those files. As a side note, the chunk-fingerprint store in File Agent can be corrupted as the corruptions of other normal files. Thus before each backup and restore operation, SAFE checks the correctness and completeness of the local chunk-fingerprint store by comparing it with that stored in Master Server, and when the File Agent fails or local chunk-fingerprint store fails, SAFE rebuilds it from Master Server. Obviously in this protocol, there will be some transmitting and computing overheads incurred for doing this comparison before each backup and restore operation. However, SAFE keeps

this overhead minor by only transmitting the signature of the local chunk-fingerprint store and comparing it with that stored in Master Server. The detailed comparison implemented in SAFE is omitted in this paper due to the space constraints.

After getting all the chunk fingerprints of the same files, UMR in the last step identifies the unmodified data chunks by finding the same chunk fingerprints that exist in both the corrupted version and backed-up version, and finally notifies Master Server that the remained data chunks that have been modified must be transferred from the corresponding Storage Servers for data restorations, thus removing the unmodified data chunks from transmission to reduce the restore time. As a side note, since UMR only finds the unmodified data among chronological file versions with the same file names, without considering the chronological file versions with different file names caused by the file rename or file copy operations, there will be some unmodified data chunks that are regard as modified data chunks with false positive and still require being transferred from the corresponding Storage Servers for data restorations.

5 Performance Evaluations

We have built both a trace-driven simulator and a prototype implementation of SAFE, fed real-world datasets to evaluate its performance. The SAFE simulator is used to evaluate the backup performance and the SAFE prototype system is used for evaluating the restore performance.

5.1 Experimental Setup

In SAFE simulator and prototype system, SAFE's Master Server and Storage Server, are both featured with two-socket dual-core 2.1 GHz CPUs, a total of 2 GB memory, 1 Gbps NIC cards, and a 500 GB hard drive. Multiple clients, equipped with different hardware but the same Windows XP operating system, are installed with File Agent and fed with the real-world datasets.

5.1.1 SAFE Simulator

SAFE simulator is composed of two parts: the FSBP program (described in Section 2.2) which is used to collect backup datasets and a prototyped SAFE functional module that simulates the functionalities of Master Server and File Agent (described in Sections 3 and 4). Due to privacy concerns of the collected datasets, Storage Server is not simulated for backup operations. However, this does not impact the correctness of the experimental results.

FSBP extracts the metadata information from the backup datasets at both the file level and chunk level.

File Level At the file level, FSBP collects the file attributes and computes the file hashes. The file attributes, including file ownership, file name, file type, file size and file timestamps, convey file semantic information exploited in SAFE's semantic-aware elimination.

Chunk Level At the chunk level, chunk fingerprints along with chunk sizes are collected. Each file is broken into a series of chunks with an average size of 8 KB by the Rabin fingerprint algorithm [33] and each chunk is named by the SHA-1 hash function [30].

We feed the collected datasets to SAFE in the order of their original sequence of backup operations and record the statistics such as data compression ratio, deduplication time, backup time, and so on for the backup performances analysis.

5.1.2 Relevant Systems for Comparison

To assess SAFE's benefits and limitations, we compare SAFE's performance with that of the source local chunk-level deduplication scheme and source global chunk-level deduplication scheme that are widely used in cloud backup environment.

Source Local Chunk-Level Deduplication Scheme (L-CDS) This scheme has been applied in commercial products such as Syncsort Backup Express [39] and in research prototypes such as Cumulus [46]. We have implemented the Cumulus prototype, the only published L-CDS prototype with sufficiently detailed description.

Source Global Chunk-Level Deduplication Scheme (G-CDS) This scheme has been applied in many commercial products [11, 15, 29]. We implement the prototype by using the well-known approaches in DDFS [52] for the redundancy identification at the server site.

5.2 The Evaluation of the Backup Performance

In SAFE, the hybrid deduplication function and semantic-aware elimination work in synch to remove the redundant data for backup operations. In this subsection, we compare the experimental results in deduplication efficiency, deduplication overhead and backup time that have been obtained from our SAFE simulator, L-CDS and G-CDS.

5.2.1 Trace Workload

We distributed FSBP to twenty-nine members in our research group, of whom twenty-five installed it on their desktop PCs and the other four on their laptops, and all installed the windows XP operating system. All of them did

semi-regular backups of their important personal directories (i.e., the user data including course files, project documents, source trees, developing tools and other private files but without the operating system files) from June 1st, 2009 to August 31st, 2009. Altogether, there are 1568 backups with a total of 1.07TB data, consisting of 4, 942, 186 files. Moreover, we have further used a subset of the trace reported by Xia et al. [48] that was collected from 15 graduate students totaling about 530 GB data. During our experimental evaluations, we first fed SAFE with the dataset collected by Xia and then followed with that collected by ourselves. As a side note, the dataset that was reported in Section 2.2 is excluded here, which was collected at different times for different purposes.

5.2.2 Deduplication Efficiency

Our experimental results present the cumulative deduplication efficiency for a total of 1568 backups. As a side note, SAFE does not rely on the specified chunking algorithm and chunk size, and thus we only focus on the deduplication efficiency that is impacted by the redundant data identification process but without the chunking algorithm. Nevertheless, both the chunking algorithm and the average chunk size have some influences on the deduplication efficiency, and thus to compare the deduplication efficiency on equal terms, we use the same chunking algorithm and the same average chunk size, and fed the same chunks to the three deduplication methods to compare their experimental results.

Figure 2a compares the cumulative deduplication efficiency of the three deduplication methods. We define deduplication efficiency as the ratio between the amount of the redundant data actually removed by each deduplication method and the total amount of the redundant data in each dataset. Obviously, the maximum deduplication efficiency is 1 (i.e., 100 %) and the minimum is 0 (i.e., 0 %). The results show that G-CDS removes all the redundant data at the chunk level while SAFE leaves 1.35 % of redundant data intact to reduce the deduplictation overhead to accelerate the whole deduplication process. As expected, L-CDS leaves about 40.69 % of redundant data intact because it ignores the redundant data across different clients, which is much more than SAFE.

5.2.3 Deduplication Overhead

We use the deduplication time required for each backup session as a metric to evaluate the deduplication overhead. Figure 2b compares the cumulative deduplication overhead (i.e., deduplication time) of the three methods, normalized to the overhead of G-CDS. The results show that SAFE introduces only 34.6 % of G-CDS' overhead on average while L-CDS incurs even lower overhead, at 26.2 %. During each backup session, SAFE's local deduplication process at the client site takes less time than L-CDS by singling out compressed files and small files that incur zero disk accesses, and the same time its global deduplication process at the server site spends much less time than G-CDS by identifying duplicate files at the file level instead of at the chunk level as G-CDS does and only processing large duplicate files by ignoring small duplicate files and all similar files. Thus SAFE incurs much less overhead than G-CDS and only slightly more than L-CDS.

To closely examine SAFE's tradeoff between the deduplication efficiency and overhead, we present the corresponding deduplication efficiency/overhead ratios, normalized to that of G-CDS, in Fig. 2c. Clearly, a higher ratio between efficiency and overhead represents a better tradeoff between the two measures, thus more desirable. During the first thirty backup sessions, L-CDS has higher efficiency/overhead ratio than the other two methods. But as the backup process continues to accumulate more than thirty sessions, SAFE's ratio exceeds that of L-CDS and becomes the highest among them. This is because that, when initially the backed-up data from different users is small, the local redundant data dominates the data redundancy, resulting that SAFE's global deduplication process has minor efficiency/overhead ratio and thus SAFE is less effective than L-CDS. But as the amount of backed-up data increases, the amount of global redundant data increases and becomes more dominant, thus making SAFE obtain much from its

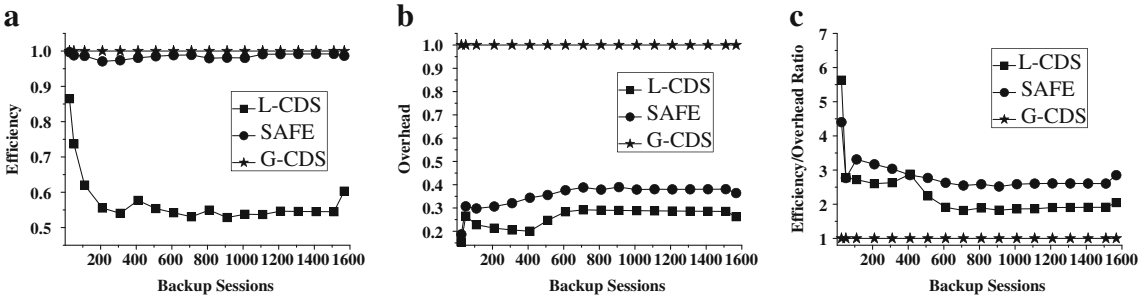

Figure 2 The tradeoff between deduplication efficiency and deduplication overhead.

global deduplication process and most cost-effective among the three methods.

Moreover, we have evaluated the deduplication efficiency/overhead ratio of a global file-level deduplication approach and compared its results with that of SAFE in Fig. 3. This global file-level deduplication approach, called G-FDS, only removes the duplicate files (without removing duplicate chunks) within the same client and across different clients globally. Compared to the chunk-level deduplication approaches, it obviously removes less redundant data and needs less deduplication time since it ignores the redundant chunk data across different files. Figure 3 compared the efficiency/overhead ratio between G-FDS and SAFE, and both are normalized to that of G-CDS. As seen from the results, it is found that G-FDS's efficiency/overhead ratio is much higher than that of SAFE, which seems that G-FDS is much more effective than SAFE. However, due to that the removal of the redundant data is the primary concern of deduplication approaches, we argue that SAFE is more effective in removing the redundant data since G-FDS dose not consider the data redundancy across different files and sacrifice much deduplication efficiency. Meanwhile, the chunk-level deduplication approach that can remove more redundant data and get higher deduplication efficiency is much more popular than file-level deduplication approach in most cases.

5.2.4 Backup Time

With regard to the three source deduplication methods in our study, the backup time consists of two parts: data deduplication time and data transfer time. For a fixed dataset, the deduplication time is generally fixed, whereas the data transfer time varies with network bandwidth. Figure 4 plots the backup time as a function of network bandwidth from an experiment where we select three backup sessions and simulate a network environment with different bandwidths:

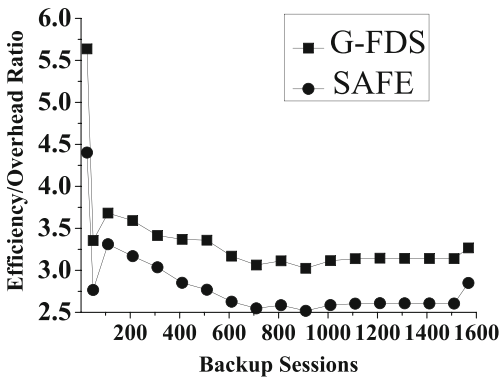

Figure 3 The comparison of SAFE and Global File-level Deduplication Approach(G-FDS) on the deduplication efficiency/overhead ratio.

100 KB/s, 800 KB/s, 1 MB/s, 2 MB/s, 4 MB/s, 8 MB/s. The data redundancy of these three backup sessions is summarized in Table 2. As a side note, the backup time is equal to the deduplication time when the network bandwidth is perfect. In Section 5.2.3, we have showed that SAFE takes much less deduplication time than G-CDS, but more than L-CDS for all of the collected backups, and thus in this subsection, we will not repeatedly show the backup time of these three selected backups under the perfect network bandwidth.

Figure 4a shows the backup time of Backup 1 that has about 69.51 MB duplicate files but only 2.02 MB duplicate chunks across different clients, which is very close to our observations reported in Section 2.2 that the duplicate files constitute a vast majority of data redundancy across clients. SAFE removes the global-redundant data at the file level with much less deduplication time than G-CDS at the chunk level while leaving only 2.02 MB (1.49 %) redundant data intact. L-CDS, which does not target global data redundancy, leaves 71.53 MB redundant data un-removed. Thus when the network bandwidth is lower than 4 MB/s, L-CDS has a very large backup time due to the heavy overhead in data transmission. SAFE, which incurs less deduplication time than G-CDS and less data transfer time than L-CDS, maintains a shorter backup time than both G-CDS and L-CDS by an average margin of 38.7 %.

Figure 4b shows the backup time of Backup 2 that has no duplicate files across different clients and no duplicate chunks that are already backed up by the same client, but only about 2.72 MB duplicate chunks shared with other clients. This is the worst case scenario that renders SAFE's global file-level and local chunk-level deduplications completely ineffective due to the complete lack of data redundancy SAFE is designed to remove. Thus compared to L-CDS, SAFE has a larger backup time for its longer deduplication time with an extra round-trip. While compared to G-CDS, SAFE's backup time is also larger for its lower compression ratio and thus the longer transfer time.

Figure 4c shows the backup time of Backup 3 that has about 482.21 MB duplicate chunks already backed up by the same client but no redundant data across different clients. Thus SAFE, L-CDS and G-CDS achieve the same compression ratio and require the same data transfer time regardless of the network bandwidth. But G-CDS, having to take more time to remove the redundant data at the server site, has a longer backup time than the other two.

As seen from the above three backups, Backup 1 is closest to our observations reported in Section 2.2, for which SAFE is designed and thus benefits the most. SAFE, designed to primarily remove duplicate files globally, incurs an extra round-trip that is more than compensated in Backup 1 but much less so in Backups 2 and Backup 3 where there are no duplicate files across clients. These backup samples

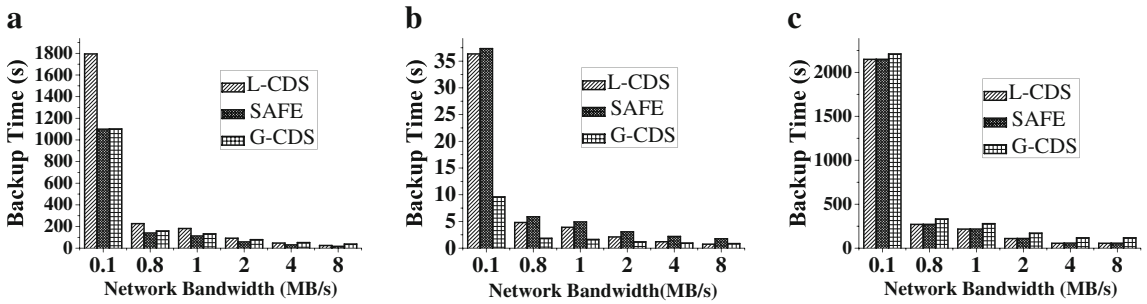

Figure 4 Backup time.

were selected to stress-test SAFE to show its overhead under extremely adversary conditions. In fact during our study, the data redundancy of most backup sessions is close to Bacukup 1 where SAFE benefits most.

Additionally, under low network bandwidth, the data transfer time dominates the backup time in most cases, stressing the importance of the deduplication efficiency. However, as the bandwidth increases, the deduplication time can hide the data transfer time and thus dominates the backup time. SAFE is the best deduplication method in trading off between the deduplication efficiency and overhead, generally achieving the shortest backup time among the three schemes under comparison.

5.2.5 The Benefits of Semantic-Aware Elimination

In this subsection, we focus on two key steps, SFD and LCD, to analyze the detailed benefits of semantic-aware elimination in SAFE.

1) *Server-side File-level Deduplication*. SFD is designed to remove duplicate files across different clients in Master Server. In this stage, we exploit two file semantics, *file size* and *file locality*, to reduce the disk accesses incurred by the out-of-memory file-hashes indexing.

File-size Threshold To reduce the disk accesses, SAFE excludes small files as they account for a large percentage of all files but a very small fraction of the actual data. The file-size threshold by which small files are excluded clearly affects the amount of redundant data to be removed and the deduplication time. In general, the smaller this threshold value is, the larger the number of files will be subject to deduplication but at the cost of more disk accesses. Therefore, the file-size threshold must be judiciously chosen to achieve a good balance between the deduplication overhead and deduplication efficiency. Figure 5a plots the cumulative deduplication time of SFD required for all the backups as a function of the file-size threshold. It shows that the larger the file-size threshold, the shorter the deduplication time will be. In particular, as the threshold increases from 0 KB to 8 KB, the deduplication time is significantly reduced. Figure 5b plots the corresponding redundant data that has been removed by SFD. The remained data after SAFE is just the amount of data to be transferred, which is proportional to the data transfer time. To better understand the tradeoff between the deduplication efficiency and overhead, we plot the corresponding backup time in Fig. 5c, which is the sum of the deduplication time and the data transfer time under the network bandwidth of 800 KB/s. It shows that the minimum backup time is achieved at the file-size threshold of 8 KB, as a result of the substantially reduced deduplication time during this period. On the other hand, the backup time increases with the file-size threshold beyond the 8 KB point, due to the increasing amount of data required to be transferred. We thus believe that the file-size threshold of 8 KB(i.e., the average chunk size in our dataset) is best suited for our collected data sets.

Disk I/O Reduction After filtering out small files and exploiting file locality, the disk I/Os are substantially

Table 2 Data redundancy of the three backups.

	Duplicate data in the same client	Duplicate data across different clients	
		Duplicate files	Duplicate chunks
Backup 1	66.84 MB	69.51 MB	2.02 MB
Backup 2	0 MB	0 MB	2.72 MB
Backup 3	482.21 MB	0 MB	0 MB

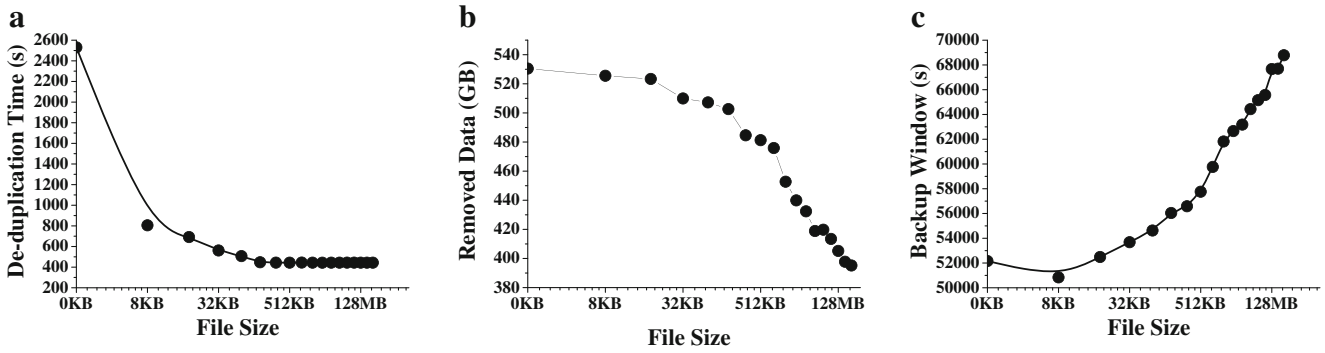

Figure 5 File-size threshold.

reduced in SFD. During our experiments, an I/O reduction of about 40 % can be reached through small-file filtering alone and 78 % can be reached by combining small-file filtering and file locality exploitation in the users'initial backups.

2) *Local Chunk-level Deduplication*. To reduce disk accesses in LCD, SAFE uses two-tiered indexing scheme and singles out two types of files, *small files* and *compressed files*, with zero disk accesses. As indicated in our experimental results, the combination of these methods can eliminate about 99.9 % of the disk I/Os, and the efficiency of the compressed-file-exclusion method varies from user to user depending on the backup datasets.

5.3 The Evaluation of The Restore Performance

In some restore scenarios when the local unmodified data is available, SAFE's data restore is assisted by the Unmodified Data Removal module, which helps to remove the unmodified files and data chunks from data transmission to reduce the restore time. In this subsection, we use SAFE prototype system and show the experimental results on both directory restore and individual file restore when the UMR module is not only effective but also ineffective.

5.3.1 Datasets

We use two datasets to evaluate SAFE's restore performances. One consists of 16 full backups of one author's home directory, totaling about 139.11 GB data, and the other consists of a tar file of Linux source tree with four backed-up versions. Moreover, because it is hard to trigger realistic data disasters, we simulate the disasters by injecting data corruptions that select some files to apply file rename operation, file modification(including data insertion, deletion and modification) and file deletion to the last version of the dataset, and then enable it to be restored to its previously backed-up versions. While we realize that these are contrived data disasters, we have not been able to find a better way to simulate/emulate disasters.

5.3.2 Directory Restore

We use the dataset of 16 full backups of one author's home directory (i.e., the user data including course files, project documents, source trees, developing tools and other private files but without the operating system files) to evaluate the directory restore performances. There is a total of 15 simulated restores by restoring this directory on the 16th day to its previous backed-up versions on the 1st day, 2nd day, 3rd day,..., and 15th day.

Figure 6 shows the amount of the redundant data that can be removed during each of the 15 restores if the local unmodified data chunks are available. As shown in this figure, the amount of the redundant data increases as the restored backup point approaches the 16th day, and when the restored backup point is on the 15th day, the amount of the data removed reaches the maximum value of about 8.26 GB, with a data reduction ratio of about 11.2 : 1 compared to the original size of the directory of about 9.07 GB. The reason behind such a high reduction ratio is that, during

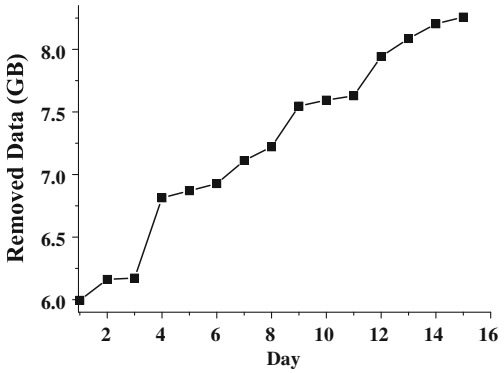

Figure 6 The amount of the redundant data removed by SAFE when local unmodified data is available.

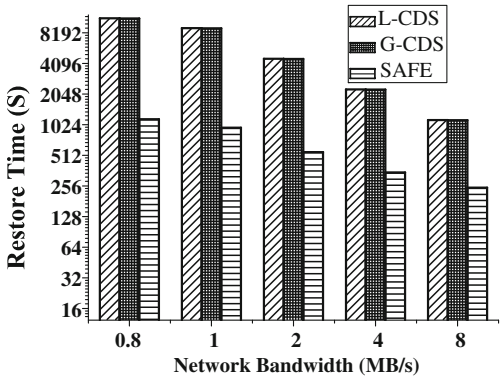

Figure 7 The comparison of the directory restore time when local unmodified data is available.

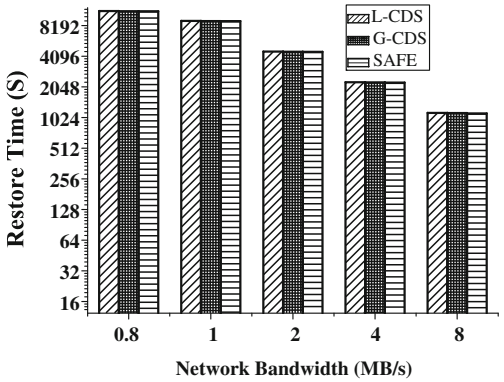

Figure 8 The comparison of the directory restore time when local unmodified data is unavailable.

directory restores, most of the files are kept intact after backups that can be removed from transmission, given that most data writes are centered on a small set of files in typical file systems [31]. However, if the data restoration is triggered by the site disasters or hardware failures when the local unmodified data is unavailable, there will be no redundant data that can be removed to optimize the restore performances. Thus below we will show the restore time of SAFE when the local unmodified data is available and unavailable.

To quantify the reductions on the directory restore time, we focus on the restore operation that restores the home directory on the 16th day to its backed-up version on the 15th day under a simulated network environment with different bandwidths: 800 KB/s, 1 MB/s, 2 MB/s, 4 MB/s, 8 MB/s. Figure 7 compares the restore time required for SAFE prototype system, L-CDS and G-CDS when the local unmodified data is available. As seen from this figure, SAFE significantly reduces the restore time after removing the unmodified data from transmission compared with G-CDS and L-CDS. Its reduction ratio is up to 9.7 : 1 when the network bandwidth is 800 KB/s. However, this reduction ratio decreases as the network bandwidth increases. Figure 8 shows the restore time required for the three methods when the local unmodified data is unavailable. In this case, SAFE, L-CDS and G-CDS all have to transfer all the restored data and costs nearly the same restore time regardless of the network bandwidth.

5.3.3 Individual File Restore

In addition to directory restores, the restore of individual files is another common restore operation in cloud backup environment. 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 backups. Thus during individual file restores, the amount of redundant data that can be removed is much less than that in directory restores. Another feature of the individual file restore is that it would never be triggered by the site disasters or disk failures that render the local unmodified data unavailable, and thus in our experiments, we only evaluate the individual file restore performances when the UMR module is effective by using a tar file of Linux source tree with four backed-up versions, without considering that the UMR module is ineffective and the unmodified data is unavailable.

We simulate three file restores that restore the 4th file version to its 1st, 2nd, and 3rd versions. Table 3 shows the file sizes of the four file versions and the amounts of redundant data that can be removed during the three simulated file restores. Similar to directory restores, the amount of the redundant data removed by SAFE increases as the file version approaches the 4th 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 of each specific dataset.

Figure 9 compares the restore time required for the SAFE prototype system, L-CDS and G-CDS under the network bandwidth of 800 KB/s. This figure shows that SAFE spends respectively 83.4 %, 80.04 %, and 73.73 % of the time required for G-CDS and L-CDS for the three simulated

Table 3 The file sizes of the four versions of tar file and the redundant data removed by three simulated file restores.

File version	File size	The redundant data removed during simulated restores
1st version	353.21 MB	63.28 MB
2nd version	365.83 MB	77.75 MB
3rd version	382.42 MB	105.4 MB
4th version	394.91 MB	–

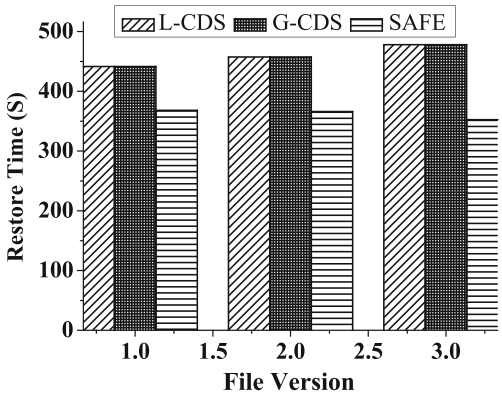

Figure 9 The comparison of the individual file restore time.

restores after removing the unmodified data from transmission, while G-CDS and L-CDS cost nearly the same restore time regardless of the restore sessions.

5.4 Discussions

Client Overhead SAFE uses more CPU power and storage space at the client site to compensate for the lack of sufficient network bandwidth for cloud backup and cloud restore operations. We argue that this is a good strategy with an acceptable overhead, given that the rapid advancement in processor and storage technologies is making processing capabilities and storage capacities increasingly affordable.

Flexibility It is relatively easy for SAFE to provide a user interface by which a user can choose between Global File-level Deduplication and Local Chunk-level Deduplication for data backup, and whether to use the Unmodified Data Removal for data restoration. Our rationale behind this possible choice is that a cloud backup user may be aware of the nature and characteristics of his/her datasets, if he/she is the one knowledgeable of the provenance of the datasets, and thus may choose a particular option accordingly to avoid the unnecessary data transmission cost and deduplication overhead.

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. For the single directory restore and individual file restore, SAFE can help find and remove the unmodified data to avoid the data transmission as long as the whole directory or file has not been deleted, which eases the searching burden of the users especially by facing the directories including tens of thousands of files or even more files. While for the full file-system restore, SAFE can also find the unmodified data if the file system is corrupted by software failures or viruses attacks and the unmodified data is available and accessible, significantly improving the restore performances.

6 Related Work

Recently, data deduplication has been emerged as an alternative lossless data compression [3, 9, 18] scheme that has been employed in various backup and archival systems [13, 14, 21, 25, 32, 49–52]. In cloud backup environment, it is desirable to have the redundant data removed at source client before reaching backup destination to reduce the network bandwidth consumption, which is different from the target deduplication schemes employed in such systems as DDFS [52], Sparse Index [24], Falconstor [17], Exgrid [16], and Sepaton DeltaStor [38]. Table 4 compares the source deduplication methods used in five well-known developed backup systems, along with our proposed SAFE scheme. As indicated in the table, SAFE differs greatly from all of them in a few ways. First, their deduplication methods used for backup operations are very different. The existing approaches either use the source global chunk-level deduplication which removes all the redundant data among different clients globally and thus incurring heavy system overhead, or the source local chunk-level deduplication which incurs very little system overhead by only removing redundant data locally within the individual client. SAFE, distinct from all of them, combines the source global file-level deduplication scheme and the source local chunk-level deduplication scheme to effectively trade off between the deduplication efficiency and deduplication overhead for cloud backup operations. Second, SAFE further exploits file semantics, such as file locality, file timestamps, file size and file type, to significantly reduce the deduplication overhead in its both global file-level deduplication stage and local chunk-level deduplication stage to accelerate the overall deduplication process for cloud backup Operations. The file semantics have been widely used in the design and optimization of file systems, such as perfecting and caching [47]. Extracting file semantics, motivated by our experimental observations, is proven very useful to improve SAFE's deduplication performances. Third, SAFE exploits the data redundancy for some restore scenarios to optimize the cloud restore performance, while most of the existing deduplication approaches only focus on removing the redundant data from transmission for backup operations, paying little attention to the fact that the restore operations over low-bandwidth WAN networks suffer serious performance degradation and must also be improved.

Our hybrid method used in SAFE is mainly motivated by our own study revealing that the redundant data across different clients is dominated largely by duplicate files, implying that the global file-level deduplication is sufficient

Table 4 The comparison of source deduplication schemes used in recently developed backup systems.

	Backup optimization			Restore optimization
	Global/Local	File/Chunk	Semantic-aware	
NetBackup PureDisk [29]	Global	Chunk	No	No
Commvault Simpana [11]	Global	Chunk	No	No
EMC Avamar [15]	Global	Chunk	No	No
Syncsort Backup Express and NetApp [39]	Local	Chunk	No	No
Cumulus [46]	Local	Chunk	No	No
CABdedupe [40]	Local	Chunk	No	Yes
SAFE	Global file + Local chunk		File semantic	Yes

to remove its vast majority of redundant data. Moreover, similar observations have been made and reported in the literature. J. C. Tang et al. [41] have found that about 54 % of files are duplicate ones among the members of an organization, accounting for 32 % of the total storage space, and a negligibly small fraction of files are similar ones among them. Microsoft's file metadata study results [4] show that users locally contribute to a decreasing fraction of their systems' content while file copies from others contribute to an increasing fraction. Besides SAFE, other system vendors [23] also exploit both the file level and chunk level deduplication schemes, such as TAPER [22] for reducing the data transmission for replica synchronization, Deep Store [51] for saving storage space in the archival system, Extreme Binning [7] for improving chunk-level deduplication throughput in backup systems, and etc. Although all of these systems have used the concept of the hybrid method, their detailed motivation and implementations are different from that of SAFE. As described before, SAFE's hybrid method is main motivated by our observation that the redundant data across different clients is dominated largely by duplicate files and the fact that the removal of duplicate files is much easier than that of duplicate chunks, and thus we only remove the duplicate files globally but not duplicate chunks globally to improve the deduplication efficiency/overhead ratio. Microsoft's study [27] also reveals that 75 % of the redundant data are generated from the duplicate files after the survey of the file system content data collected from 857 desktop computers, and the only removal of the duplicate files across different file systems can eliminate large percent of the redundant data. While in other systems, the use of file-level deduplication scheme is just helping to first filter out the data redundancy before resorting to the chunk-level deduplication, so as to reduce their chunk-level deduplication overhead.

Besides SAFE, a rich body of previous research has addressed the problem of data transmission over low-bandwidth network applications. Table 5 compares the methods used in some well-known network applications, along with our proposed SAFE scheme. Rsync [37] is an early study that uses the delta compression to remove the redundant data that is already stored in the server. Unfortunately, due to that the delta compression only removes the redundant data between two files, reference file version and current file version, it only focuses on the redundant data between two files with identical file names. Unlike Rsync, LBFS [28] exploits the data redundancy among all the files using the chunk-level deduplication scheme, which removes all the redundant data at the chunk level to improve the network filesystem performances. Distinct from SAFE, LBFS has not exploited the file semantics and even used the hybrid method that combined the file level and chunk level deduplication schemes to further reduce the overall deduplication overhead. TAPER [22] is a scalable data replication protocol that provides a four-phased redundancy elimination scheme to balance the tradeoff between the network

Table 5 The comparison of methods used to address the problem of data transmission over low-bandwidth networks in well-known network applications.

	Research goal	Redundant data elimination	Semantic-aware	Read or write optimization
Rsync [37]	Incremental file transfer	Delta compression	No	Write
LBFS [28]	Network file system	Chunk deduplication	No	Read and write
TAPER [22]	Replica synchronization	Directory+File+Chunk +Byte deduplication	No	Write
SAFE	Cloud backup service	File+Chunk deduplication	File semantic	Read and write

bandwidth savings and computation overheads. Although TAPER has exploited the file-level and chunk level deduplication scheme like SAFE does, it has not exploited any file semantics to further reduce the computation and matching overheads. Moreover, it cannot be directly integrated into cloud backup systems since it only focuses on the redundant data among the different versions of the same directory, without considering the redundant data across different directories and even across different clients. Besides these three methods, other approaches [6, 43, 53, 54] used in distributed file systems also have addressed this transmission problem, such as the recipe technique used in [44], the look aside caching technique used in [42], and etc. However, they are all not designed specially for backup systems and have limited effectiveness in cloud backup service environment.

In addition to the deduplication technology employed in cloud backup environment, the wide area data services (WDS), such as Riverbed [35], can also be leveraged to remove the redundant data from transmission over WAN to alleviate the network bottleneck. However, due to the fact that SAFE is specially designed for backup/restore workloads at the application level, we argue that SAFE is more efficient and cost-effective than WDS-based approaches.

7 Conclusion

Motivated by the observations from our preliminary studies, we propose an alternative source deduplication framework, SAFE, to improve the efficiency of both cloud backup and restore operations. SAFE not only employs a hybrid deduplication approach that combines the global file-level deduplication and local chunk-level deduplication and further exploits file semantics (including file locality, file timestamps, file type, file size) to narrow the search space for redundant data in backup operations to reduce backup times and save storage costs, but also exploits the data redundancy for some restore operations to remove the unmodified files and data chunks from transmission to reduce the restore times. Compared with the widely used L-CDS (source local chunk-level deduplication scheme) and G-CDS (source global chunk-level deduplication scheme) that work for removing the redundant data for backup operations, SAFE achieves a compression ratio approaching that of G-CDS with a small difference of 1.35 %, while incurring a deduplication overhead very close to that of the L-CDS method, thus achieving a much higher deduplication efficiency/overhead ratio than existing solutions and shortens the backup time by an average of 38.7 % during backup operations. During some restore operations when the local unmodified data is available, SAFE significantly reduces the restore time by a reduction ratio of up to 9.7 : 1 after removing up to 91.8 % of this unmodified data. As a direction of future work, we plan to investigate more file semantics to explore the data redundancy to optimize deduplication performances for both backup and restore operations.

Acknowledgments This work is supported by the Fundamental Research Funds for the Central Universities under Grant No.0903005203206 and No.CDJZR12180006, the National High Technology Research and Development (863 Program) of China under Grant No.2013AA013202 and No.2013AA013203, Chongqing High-Tech Research Programcsct2012ggC40005, National Basic Research 973 Program of China under Grant No. 2011CB302301, NSFC No.61025008, No.61232004 and No.61173014, the US NSF under grants IIS-0916859, CCF-0937993, CNS-1016609, CNS-1116606 and CNS-1015802.

References

1. Branch Office Optimization (2007). Enterprise strategy group.
2. Data loss survey: http://www.idgconnect.com (2010).
3. Adya, A., Bolosky, W.J., Castro, M., Cermak, G., Chaiken, R., Douceur, J.R., Howell, J., Lorch, J.R., Theimer, M., Wattenhofer, R.P. (2002). FARSITE: federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, *36*(SI), 1–14.
4. Agrawal, N., Bolosky, W.J., Douceur, J.R., Lorch, J.R. (2007). A five-year study of file-system metadata. In *FAST'07*.
5. Amazon Simple Storage Service: http://aws.amazon.com/s3.
6. Annapureddy, S., Freedman, M.J., Mazieres, D. (2005). Shark: Scaling file servers via cooperative caching. In *NSDI'05*.
7. Bhagwat, D., Eshghi, K., Long, D.D., Lillibridge, M. (2009). Extreme binning: scalable, parallel deduplication for chunk-based file backup. Technical Report, HPL-2009-10R2 HP Laboratories.
8. Bloom, B.H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, *13*(7), 422–426.
9. Bobbarjung, D.R., Jagannathan, S., Dubnicki, C. (2006). Improving duplicate elimination in storage systems. *ACM SIGOPS Transactions on Storage*, *2*(4), 424–448.
10. Cabrera, L., Rees, R., Steiner, S., Hineman, W., Pennere, M. (1995). ADSM: A multi-platform, scalable, backup and archive mass storage system. In *Compcon'95*.
11. Commvault Simpana: http://www.commvault.com.
12. Debnath, B., Sengupta, S., Li, J. (2010). ChunkStash: Speeding up inline storage deduplication using flash memory. In *USENIX'10*.
13. Dong, W., Douglis, F., Li, K., Patterson, H. (2011). Tradeoffs in scalable data routing for deduplication clusters. In *FAST'11*.
14. Dubnicki, C., Gryz, L., Heldt, L., Kaczmarczyk, M., Kilian, W., Strzelczak, P., Szczepkowski, J., Ungureanu, C., Welnicki, M. (2009). Hydrastor: A scalable secondary storage. In *FAST'09*.

15. EMC Avamar: http://www.emc.com.
16. Exgrid: http://www.exagrid.com.
17. Falconstor: http://www.falconstor.com.
18. Ferreira, R.A., Ramanathan, M.K., Grama, A., Jagannathan, S. (2007). Randomized protocols for duplicate elimination in peer-to-peer storage systems. *IEEE Transactions on Parallel and Distributed Systems*, *18*(5), 686–696.
19. Forman, G., Eshghi, K., Suermondt, J. (2009). Efficient detection of large-scale redundancy in enterprise file systems. *ACM SIGOPS Operating Systems Review*, *43*(1), 84–91.
20. Gantz, J.F., Chute, C., Manfrediz, A., Minton, S., Reinsel, D., Schlichting, W., Toncheva, A. (2008). The diverse and exploding digital universe: an updated forecast of worldwide information growth through 2011. IDC Report.
21. Guo, F., & Efstathopoulos, P. (2011). Building a high-performance deduplication system. In *USENIX ATC'11*.
22. Jain, N., Dahlin, M., Tewari, R. (2005). TAPER: Tiered approach for eliminating redundancy in replica synchronization. In *FAST'05*.
23. Kulkarni, P., Douglis, F., LaVoie, J., Tracey, J.M. (2004). Redundancy elimination within large collections of files. In *USENIX' 04*.
24. Lillibridge, M., Eshghi, K., Bhagwat, D., Deolalikar, V., Trezise, G., Campbell, P. (2009). Sparse indexing: Large scale, inline deduplication using sampling and locality. In *FAST'09*.
25. Liu, C., Gu, Y., Sun, L., Yan, B., Wang, D. (2010). R-ADMAD: High reliability provision for large-scale de-duplication archival storage systems. In *ICS'09*.
26. Meister, D., & Brinkmann, A. (2009). Multi-level comparison of data deduplication in a backup scenario. In *SYSTOR'09*.
27. Meyer, D.T., & Bolosky, W.J. (2011). A study of practical deduplication. In *FAST'11*.
28. Muthitacharoen, A., Chen, B., Mazières, D. (2001). A low-bandwidth network file system. In *SOSP'01*.
29. NetBackup PureDisk: http://www.symantec.com.
30. NIST: Secure hash standard (1993). In *FIPS PUB* (Vol. 180, p. 1).
31. Policroniades, C., & Pratt, I. (2004). Alternatives for detecting redundancy in storage systems data. In *USENIX'04*.
32. Quinlan, S., & Dorward, S. (2002). Venti: A new approach to archival storage. In *FAST'02*.
33. Rabin, M.O. (1981). Fingerprinting by random polynomials. Technical Report TR-15-81. Harvard University: Center for Research in Computing Technology.
34. Rhea, S., Cox, R., Pesterev, A. (2008). Fast, inexpensive content-addressed storage in foundation. In *USENIX'08*.
35. Riverbed: http://www.riverbed.com.
36. Roselli, D., Lorch, J.R., Anderson, T.E. (2000). A comparison of file system workloads. In *USENIX'00*.
37. Rsync: http://rsync.samba.org.
38. Sepaton DeltaStor: http://www.sepaton.com.
39. Syncsort Backup Express and NetApp: http://www.syncsort.com.
40. Tan, Y., Jiang, H., Feng, D., Tian, L., Yan, Z. (2011). CABdedupe: A causality-based deduplication performance booster for cloud backup services. In *IPDPS'11*.
41. Tang, J.C., Drews, C., Smith, M., Wu, F., Sue, A., Lau, T. (2007). Exploring patterns of social commonality among file directories at work. In *CHI'07*.
42. Tolia, N., Harkes, J., Kozuch, M., Satyanarayanan, M. (2004). Integrating portable and distributed storage. In *FAST'04*.
43. Tolia, N., Kaminsky, M., Andersen, D.G., Patil, S. (2006). An architecture for internet data transfer. In *NSDI'06*.
44. Tolia, N., Kozuch, M., Satyanarayanan, M., Karp, B. (2003). Opportunistic use of content addressable storage for distributed file systems. In *USENIX'03*.
45. Unstructured_data: http://en.wikipedia.org/wiki/Unstructureddata.
46. Vrable, M., Savage, S., Voelker, G.M. (2009). Cumulus: filesystem backup to the cloud. *ACM Transactions on Storage*, *5*(4), 1–28.
47. Xia, P., Feng, D., Jiang, H., Tian, L., Wang, F. (2008). FARMER: A novel approach to file access correlation mining and evaluation reference model for optimizing peta-scale file system performance. In *HPDC'08*.
48. Xia, W., Jiang, H., Feng, D., Hua, Y. (2012). SiLo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput. In *USENIX ATC'11*.
49. Yang, T., Feng, D., Niu, Z., Wan, Y. (2010). Scalable high performance de-duplication backup via hash join. *Journal of Zhejiang University Science*, *11*(5), 315–327.
50. Yang, T., Jiang, H., Feng, D., Niu, Z., Zhou, K., Wan, Y. (2010). *DEBAR: a scalable high-performance de-duplication storage system for backup and archiving*. IPDPS'10.
51. You, L.L., Pollack, K.T., Long, D.D.E. (2005). Deep store: An archival storage system architecture. In *ICDE'05*.
52. Zhu, B., Li, K., Patterson, H. (2008). Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST'08*.
53. Qiu, M., Sha, E.H.-M. (2009). Cost minimization while satisfying hard/soft timing constraints for heterogeneous Embedded Systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, *14*(2), 1–30.
54. Li, J., Qiu, M., Ming, Z., Quan, G., Qin, X., Gu, Z. (2012). Online optimization for scheduling preemptable tasks on IaaS cloud systems. *Journal of Parallel and Distributed Computing (JPDC)*, *72*(5), 666–677.

Yujuan Tan received the B.Sc. degree in Computer Science and Engineering in 2006 from Hunan Normal University, Changsha, China; and the PhD degree in Computer Science and Engineering in 2012 from Huazhong University of Science and Technology, Wuhan, China. Since July 2012 She has been at Chongqing University, Chongqing, China, where she serve as an Assistant Professor in the College of Computer Science. Her present research interests include computer architecture, computer storage systems, big data computing, cloud computing, and performance evaluation. She has over 15 publications in journals and international conferences, including TACO, IPDPS, ICPP, HiPEAC, and NAS. Dr. Tan is a member of ACM.

Hong Jiang received the B.Sc. degree in Computer Engineering in 1982 from Huazhong University of Science and Technology, Wuhan, China; the M.A.Sc. degree in Computer Engineering in 1987 from the University of Toronto, Toronto, Canada; and the PhD degree in Computer Science in 1991 from the Texas A&M University, College Station, Texas, USA. Since August 1991 he has been at the University of Nebraska-Lincoln, Lincoln, Nebraska, USA, where he is Willa Cather Professor of Computer Science and Engineering. At UNL, he has graduated 12 Ph.D. students who upon their graduations either landed academic tenure-track positions in Ph.D.-granting US institutions or were employed by major US IT corporations. His present research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, performance evaluation. He serves as an Associate Editor of the IEEE Transactions on Parallel and Distributed Systems. He has over 200 publications in major journals and international Conferences in these areas, including IEEE-TPDS, IEEE-TC, ACM-TACO, JPDC, ISCA, MICRO, USENIX ATC, FAST, LISA, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, ICPP, etc., and his research has been supported by NSF, DOD and the State of Nebraska. Dr. Jiang is a Senior Member of IEEE, and Member of ACM.

Edwin Hsing-Mean Sha received Ph.D. degree from the Department of Computer Science, Princeton University, USA in 1992. From August 1992 to August 2000, he was with the Department of Computer Science and Engineering at University of Notre Dame, USA. Since 2000, he has been a tenured full professor in the Department of Computer Science at the University of Texas at Dallas. Since 2012, he served as the Dean of College of Computer Science at Chongqing University, China. He has published more than 300 research papers in refereed conferences and journals. He has served as an editor for many journals, and as program committee and Chairs for numerous international conferences. He received Teaching Award, Microsoft Trustworthy Computing Curriculum Award, NSF CAREER Award, and NSFC Overseas Distinguished Young Scholar Award, Chang-Jiang Honorary Chair Professorship and China Thousand-Talent Program

Zhichao Yan received the B.Sc. degree in Computer Science and Engineering in 2006 from Huazhong Agricultural University, Wuhan, China; and the PhD degree in Computer Science and Engineering in 2013 from Huazhong University of Science and Technology, Wuhan , China. Her present research interests include computer architecture, parallel I/O, high-performance computing, big data computing, cloud computing, performance evaluation. He has over 15 publications in journals and international conferences, including TACO, IPDPS, ICPP, HiPEAC, and NAS.

Dan Feng received her B.E, M.E. and Ph.D. degrees in Computer Science and Technology from Huazhong University of Science and Technology (HUST), China, in 1991, 1994 and 1997 respectively. She is a professor and director of Data Storage System Division, Wuhan National Lab for Optoelectronics. She also is vice dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, parallel file systems, disk array and solid state disk. She has over 100 publications in journals and international conferences, including FAST, USENIX ATC, ICDCS, HPDC, SC, ICS and IPDPS. Dr. Feng is a member of IEEE and a member of ACM.