Deduplicating Compressed Contents in Cloud Storage Environment

Zhichao Yan, Hong Jiang
University of Texas Arlington
zhichao.yan@mavs.uta.edu
hong.jiang@uta.edu

Yujuan Tan*
Chongqing University
tanyujuan@gmail.com
Corresponding Author

Hao Luo
University of Nebraska Lincoln
hluo@cse.unl.edu

Abstract

Data compression and deduplication are two common approaches to increasing storage efficiency in the cloud environment. Both users and cloud service providers have economic incentives to compress their data before storing it in the cloud. However, our analysis indicates that compressed packages of different data and differently compressed packages of the same data are usually fundamentally different from one another even when they share a large amount of redundant data. Existing data deduplication systems cannot detect redundant data among them. We propose the X-Ray Dedup approach to extract from these packages the unique metadata, such as the "checksum" and "file length" information, and use it as the compressed file's content signature to help detect and remove file level data redundancy. X-Ray Dedup is shown by our evaluations to be capable of breaking in the boundaries of compressed packages and significantly reducing compressed packages' size requirements, thus further optimizing storage space in the cloud.

1 Introduction

Due to the information explosion [1, 3], data reduction technologies such as compression and deduplication have been developed to improve the space efficiency in storage systems, including the cloud storage environment. Lossless data compression tries to find repeated strings ***within the specific range of the individual files*** and replaces them with a more compact coding scheme. It reduces bits by identifying and eliminating statistical redundancy. Data deduplication divides data into fixed-size or variable-size chunks, identifies and removes the redundant chunks ***across all the files*** by unique fingerprints of these chunks, and reassembles the chunks to serve the subsequent access operations on the data.

However, we have observed several problems in existing data deduplication systems with compressed contents. First, they cannot identify redundant data between the compressed and uncompressed versions of the exactly same contents, because the compressed contents, being encoded by the compression algorithm, will have very different string patterns from their uncompressed counterparts. Second, different compression algorithms will generate different compressed data of the same contents that render fingerprint-based redundancy identification difficult. Third, very similar but different digital contents (e.g., files or data streams), which would otherwise present excellent deduplication opportunities, will become fundamentally distinct compressed packages after applying even the same compression algorithm.

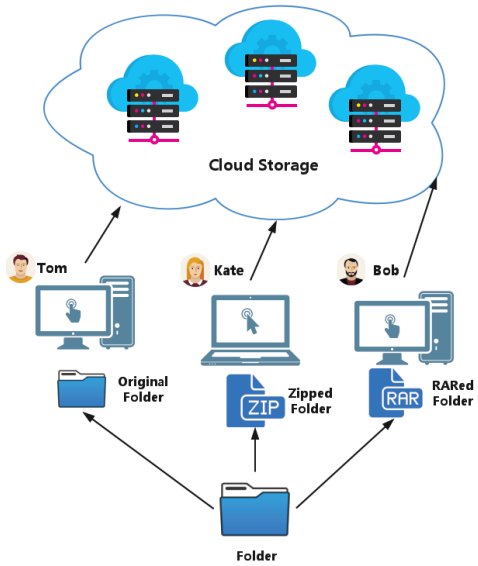

Figure 1: A user scenario on cloud storage environment

To understand the potential negative impact of the above observed problems, we describe a simple but arguably common use case in a cloud storage environment as shown in Figure 1. Three users, Tom, Kate and Bob, use the same cloud storage service to store their data; and they share substantial contents among their individual local folders (e.g., taking the same courses, working on the same projects, having similar tastes in movies, music, etc.). Both users and cloud providers have economic incentives to compress their data before storing it in the cloud. To help make the point, we assume the extreme case where the three local folders are identical. Tom directly stores the original folder to the cloud servers, while Kate and Bob choose different compression software (i.e., zip and rar) to compress their local folders before storing them to the cloud servers. The cloud storage servers must store all these three different data streams. Obviously, they are actually of different formats of representation for the same data contents (folder), and it is

enough to store only one copy from the perspective of information preservation. Unfortunately, the existing data deduplication methods will not be able to detect and remove this kind of data redundancy because the zip and rar algorithms will encode the data stream in different ways, making the two output streams different from each other. This is also the reason why traditional data deduplication systems try to skip deduplicating compressed archives because they may share little redundant data in their binary representations with the original uncompressed files, even though they may have the same digital contents [7]. As cloud storage becomes a digital content aggregating point in the digital universe, different users will likely share the same digital contents with others, often with different compression methods due to their own habits and preferences. As a result, it is arguable that this type of hidden data redundancy already exists widely in deduplication-based storage systems and will likely increase with time. Thus, it is necessary to detect and remove this kind of redundancy for a more efficient cloud storage system.

Our observation of prevailing compression algorithms, such as zip, 7z, rar, pea, tar.gz, tar.bz or tar.xz, indicate that they contain some "integrity check" mechanisms to protect their compressed contents. More specifically, some, such as tar.gz, tar.bz or tar.xz, will only protect the metadata (they compress a single archive file generated from tar, and tar will perform integrity check on its archived files' header); while others, such as zip, 7z, rar or pea, will protect all compressed files, meaning that they will perform integrity check on all their compressed files' contents. We discover that this kind of mechanism, along with some other metadata (file length), within the compressed package can be used as a file signature to detect data redundancy. As a result, it can be used to detect the potential redundancy between compressed files and uncompressed files, and among differently compressed files.

In summary, this paper makes the following contributions: (1) it studies the potential data redundancy between compressed files and uncompressed files, and among differently compressed files; (2) it proposes the X-Ray Dedup approach to identify the redundant files across the (differently) compressed files and uncompressed files in the cloud storage environment; and (3) it evaluates X-Ray Dedup to demonstrate its substantial improvements over the traditional data deduplication method.

2 X-Ray Dedup Approach

In this section, we will introduce some basic information on typical compressed packages, then elaborate on our X-Ray Dedup approach.

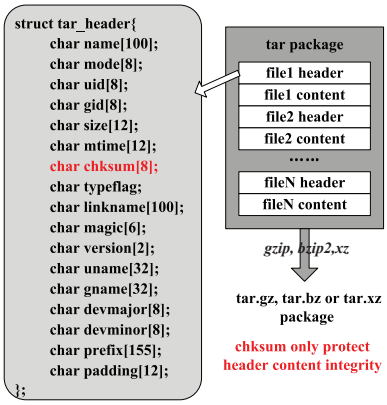

Figure 2: Compression flow for gzip, bzip2 or xz

2.1 Format of Compressed Package

Linux/Unix users usually archive their files into a folder before using one of the various compression tools (e.g., gzip, bzip2,xz, etc.) to compress it. Consequently, a great number of compressed packages have been generated with a suffix of "tar.gz", "tar.bz" or "tar.xz". Some systems will run a script to automatically translate the directory into a specific compressed package. As shown in Figure 2, the basic work flow is to archive the folder, treat it as a single file and compress it. Tar will stack each file's header and content together. However, the checksum field only protects the file header's data integrity. Compression tools will compress the tar package, which is comprised of both metadata and data.

Windows users usually use a compression program, e.g., 7zip, peazip, rar, etc., to directly compress the folder, which results in compressed packages suffixed with "7z", "pea" or "rar". These tools are also widely used, for example, the "7z" and "peazip" programs work across the Linux and Windows platforms, and the commercial software WinRAR has more than 500 millions users around the world. These compression algorithms evolved from the "zip" algorithm, and they usually calculate a "CRC32" checksum per file for data integrity. Besides CRC32, some tools can choose more complex checksum schemes such as MD5, SHA1, RIPEMD-160, SHA256, SHA512, BLAKE2, etc., which can be leveraged as the file's content signature for data deduplication.

2.2 System Overview and Operations

Based on our observation, we propose an approach, called X-Ray Dedup, to leverage the metadata information (checksum and file length) as the file's content signature within the compressed package to identify the redundant data between compressed files and uncompressed files, and among differently compressed files. Specifically, we calculate the files' checksums information in the "tar.gz", "tar.bz" and "tar.xz" packages and directly use checksums existing in the "7z", "pea" or "rar" packages.

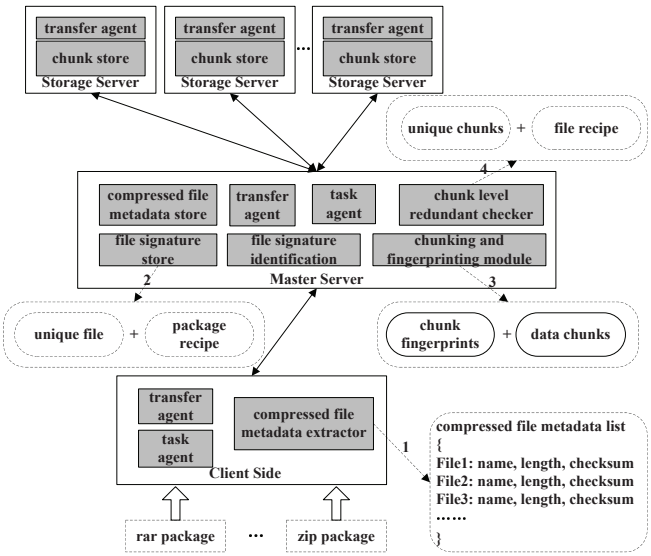

Figure 3: System overview

Figure 3 shows a typical cloud storage system integrated with our X-Ray Dedup approach. The transfer agent is responsible for data transfer between clients and servers. The system usually contains at least one master server that manages the storage tasks from different clients. Once a client sends a file to the master server, it will perform file level data deduplication before performing chunk level data deduplication. In this step, the system will first remove redundant files. Then, it will identify redundant data chunks, generate file recipe metadata (i.e., chunk mapping information) that helps reconstruct the original file and remove the redundant chunks. In the last step, it will store the unique data chunks into chunk stores within the storage servers.

The X-Ray Dedup's workflow is shown in Figure 3. It consists of 4 main steps, along with the type of data generated by each step. In Step 1, the `compressed file metadata extractor` module on the client side extracts the metadata of the compressed package by parsing through the compressed package and collecting the corresponding metadata information (i.e., name, uncompressed length and checksum). In Step 2, a file signatures store is used to help the `file signature identification` module identify and remove file-level data redundancy by its recorded files' metadata entries. In Step 3, the unique (non-redundant) files are chunked to generate data chunks and their individual fingerprints. In Step 4, the conventional chunk-level deduplication will be executed to generate file recipes and unique chunks. Finally, the previously generated package recipes, file recipes and unique chunks are stored to the storage servers. In other words, X-Ray Dedup adds an additional level of deduplication exclusively for compressed files, on top of the traditional deduplication systems.

In order to detect all redundant files in a cloud storage environment, one possible solution is to calculate all files' checksums and construct a file metadata key-value store. The scope of the key-value store can be defined by the users to limit the overhead on building and maintaining it. It is very similar to existing hash store for file-level data deduplication, except that it works with a different hash format that represents each individual files content to filter out files. More specifically, if we choose the same file-level hash signature as the one used in conventional data deduplication systems, such as"SHA1" (by configuring or extending checksum methods of compression tools on the cloud side), we can combine this structure with the existing data deduplication hash store.

In this work, we leverage the most popular checksum (CRC32) within the compressed packages. It is then combined with the "original file length" to constitute a file's signature and help detect data redundancy at the file level as follows. First, two files are considered redundant only if they have identical checksums and equal original file lengths. That is, file-level redundancy can be detected by configuring the system with an extra comparison step to compare and verify whether files have identical CRC32 values and original file lengths. Second, before removing the redundant files, a package recipe is constructed for the storage server to be able to later recover the compressed package. Each package recipe contains all the records (usually in the form of other compressed packages' pointer information) of the compressed files of corresponding package. After the redundant files within a compressed package are identified, a new compressed package without them is generated. The new package, combined with the package recipe, enables the restoration of the original compressed package.

3 Evaluation

In this section, we will introduce the evaluation environment, present and analyze some preliminary results.

3.1 Experimental Environment

We have collected two freely available code packages (coreutils and Linux kernel) as our data set. There are 20 versions of coreutils and 11 versions of Linux kernel, all in compressed packages. We run the experiments under $Ubuntu-14.04$ with the ext4 file system, and compressed packages are generated from both the *Ubuntu* and *Windows*7 platforms. We use Destor [2] as the chunk level deduplication engine. It is designed for backup applications with chunk level data deduplication. We have added the file-level data duplication feature for compressed files to it to integrate our X-Ray Dedup approach. In Table 1, we list the compression tools that we have used in this work.

Table 1: Compression tools

	tar	gz	xz	7z	rar
ubuntu	1.27.1	1.6	5.1.0α	9.20	4.20
windows	1.28-1	1.6	5.2.2	15.09β	5.31

Table 2: Sizes (KiB) of different compression formats under the Ubuntu / Windows platforms

	coreutils-8.25	linux-4.5-rc5
tar	49990 / 49990	642550 / 642550
xz	5591 / 5591	86287 / 86287
gz	12784 / 12784	132608 / 132609
7z	6169 / 5723	93561 / 89437
rar	12402 / 12401	156310 / 155135

Table 3: Comparison of redundancy ratio (in percentage) between different compressed packages between the same content, whose row is Ubuntu and column is Windows

		coreutils				linux			
		xz	gz	7z	rar	xz	gz	7z	rar
coreutils	xz	100	0	0	0	0	0	0	0.05
	gz	0	7.6	0	0	0	0	0	0.05
	7z	0	0	0	0	0	0	0	0.05
	rar	0	0	0	1.0	0	0	0	0.05
linux	xz	0	0	0	0	100	0	0	0.05
	gz	0	0	0	0	0	5.6	0	0.05
	7z	0	0	0	0	0	0	0	0.05
	rar	0	0	0	0	0	0	0	0.24

3.2 Results and Analysis

Compressed Content Study: evaluating data redundancy among different compression formats. Table 2 lists the various sizes of different formats for a specific version of selected datasets under both the Ubuntu and Windows platforms. We use the default parameters to convert the data package downloaded from the Internet into different package formats. We find that compression tools can significantly reduce the original digital contents' sizes. It can significantly reduce the costs for both users and cloud service providers. We use the chunk-level data deduplcation engine to study data redundancy among the compressed packages. As shown in Table 3, we find that nearly all pairs have 0% data redundancy, a few pairs have 0.05%-7.63% data redundancy. Except for "tar.xz", which has generated 100% identical compressed packages from both the Ubuntu and Windows platforms. We further verify that they share 0% redundancy with the compressed packages generated by $xz-5.0.8$. Although most packages have the same sizes across the two platforms, our study shows that: (1) except for "tar.xz", the compressed packages are fundamentally different from each other even under the same compression algorithm; (2) for the same digital contents, different compressed algorithms will generate fundamentally different data streams; (3) a compressed package itself has very low data redundancy (0-0.05%) at the chunk level. **All these results indicate that traditional data deduplication methods cannot detect data redundancy in the compressed packages**.

Decompressed Content Study: evaluating the data redundancy among different packages. We decompress various versions of packages and apply the chunk level duduplication engine on them. As shown in Figure 4, we plot both local and global data redundancies, where the former represents the redundancy within the current version and the latter represents the redundancy among all versions. We find that most packages have very low local data redundancy within themselves (5.29%-6.48% in coreutils and 0.57%-2.25% in Linux). Although both the local and global data redundancy rates vary over different versions, the global data redundancies are very high across different versions, indicating that high data redundancy exists among these packages (32.95%-81.45% in coreutils and 14.98%-83.84% in Linux). **All these results indicate that there are a lot of duplicate files within these compressed packages, which can be detected and removed by X-Ray Dedup**.

X-Ray Dedup Study: evaluating how much data can be deduplicated by X-Ray Dedup. In Figure 5, we show three kinds of data redundancy rate across all versions of the compressed packages, namely, chunk level redundancy, file level redundancy and compressed redundancy. Compressed redundancy is defined as the ratio of the total size of compressed intact files' size divides and the total size of compressed package. Chunk level redundancy indicates the maximal redundancy across these packages. In this study, we collect the information about the sizes before and after compression under the compression tool "rar", so the compressed redundancy is specific to "rar". File level redundancy vary from 1.39%-26.46% in coreutils and 9.04%-55.55% in Linux. X-Ray Dedup can help find this kind of redundant data and eliminate it. As a result, it can reduce the size of compressed packages by 1.61%-35.78% in coreutils and 11.05%-65.59% in Linux with its file-level data deduplication. It is worth noting that,for one particular version of the Linux dataset, the compressed redundancy is a bit higher than its chunk level redundancy. This is because the compression algorithm encodes data by its statistical redundancy. Meanwhile, we find that some coreutils versions have made major modifications on most files, leading to a very low file level redundancy. As a result, X-Ray Dedup can scan compressed packages and opt out performing file level deduplication. However, extracting metadata information from these low-redundancy compressed packages is still necessary and in fact important because there would be high file level redundancy in the compressed packages of the subsequent versions. We define the *deduplication factor for X-Ray Dedup* as the ratio of the compressed redundancy to the chunk level redundancy. **We find that X-Ray Dedup can reduce a significant amount of redundant data in compressed packages in traditional data deduplication system (the deduplication factor is** 22.20% **in coreutils and** 65.60% **in linux kernel on average)**.

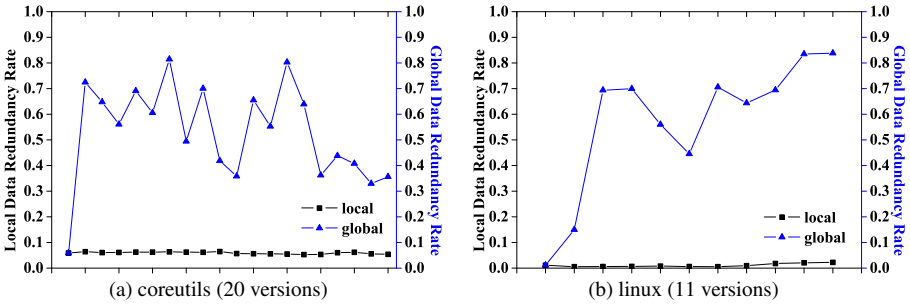

Figure 4: Real data redundancy throughout different versions of decompressed packages

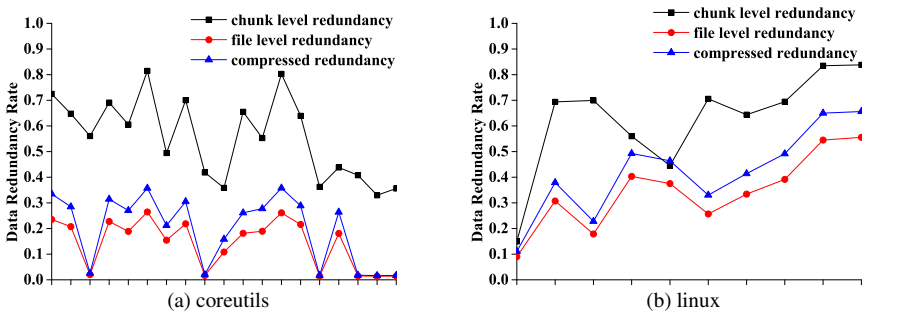

Figure 5: Compressed redundancy information of the X-Ray Dedup approach throughout all compressed packages

4 Related Work

Data deduplication was proposed to remove redundant data in backup application [9]. As more and more data migrates to the cloud, it has been integrated within the cloud platform [8]. Different from the backup storage systems where chunk-level deduplication dominates, there exists strong evidence indicating that file-level deduplication can achieve comparable compression ratio to chunk-level deduplication in the cloud environment [6]. However, it remains a challenge to find redundant data within a compressed package or among different compressed packages because conventional methods for detecting data redundancy usually scan the compressed stream itself without touching, let alone leveraging its internal information. Migratory compression [5] and M-tar [4] try to reorganize data to improve space efficiency. X-Ray Dedup can scan and extract metadata information to further help identify and remove redundant files across the compressed files and uncompressed files.

5 Acknowledgment

The authors wish to thank the reviewers for their constructive comments. We are also grateful to our shepherd, Professor Xiaosong Ma, for her very helpful feedback on this paper's revision. This work was supported in part by National Natural Science Foundation of China (NSFC) under Grant No.61402061, No.61309004, Chongqing Basic and Frontier Research Project of China under Grant No.cstc2013jcyjA40016, No.cstc2013jcyjA40025, Research Fund for the Doctoral Program of Higher Education of China under Grant No.20130191120031, and NSF CNS-1116606.

References

[1] EMC. Managing storage: Trends, challenges, and options(2013-2014). `https://education.emc.com/content/_common/docs/articles/Managing_Storage_Trends_Challenges_and_Options_2013_2014.pdf`, 2013.

[2] Fu, M., Feng, D., Hua, Y., He, X., Chen, Z., Xia, W., Zhang, Y., and Tan, Y. Design tradeoffs for data deduplication performance in backup workloads. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies* (2015), pp. 331–344.

[3] Gantz, J., and Reinsel, D. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. `https://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf`, 2012.

[4] Lin, X., Douglis, F., Li, J., Li, X., Ricci, R., Smaldone, S., and Wallace, G. Metadata considered harmful ... to deduplication. In *Proceedings of the 7th USENIX Conference on Hot Topics in Storage and File Systems* (2015), pp. 11–11.

[5] Lin, X., Lu, G., Douglis, F., Shilane, P., and Wallace, G. Migratory compression: Coarse-grained data reordering to improve compressibility. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies* (2014), pp. 257–271.

[6] Meyer, D. T., and Bolosky, W. J. A study of practical deduplication. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies* (2011), pp. 1–1.

[7] Tan, Y., Jiang, H., Feng, D., Tian, L., Yan, Z., and Zhou, G. Sam: A semantic-aware multi-tiered source de-duplication framework for cloud backup. In *Parallel Processing (ICPP), 2010 39th International Conference on* (Sept 2010), pp. 614–623.

[8] Vrable, M., Savage, S., and Voelker, G. M. Cumulus: Filesystem backup to the cloud. In *Proccedings of the 7th Conference on File and Storage Technologies* (2009), pp. 225–238.

[9] Zhu, B., Li, K., and Patterson, H. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies* (2008), pp. 18:1–18:14.