BUILDING A VERSATILE DEDUPLICATION SYSTEM

by
ZHICHAO YAN

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2018

To my family, for their endless trust, support, encourage and love.

ACKNOWLEDGEMENTS

There were many people who helped me during my PhD studying period, and I would like to take this opportunity to thank them for their kindness and selfless help.

First of all, I would like to thank my supervising professor Dr. Hong Jiang for constantly motivating and encouraging me, and also for his invaluable support during the course of my doctoral studies. Dr.Jiang has set an example of excellence as a researcher, mentor, instructor, and role model.

I wish to thank my thesis committee members Dr. Hao Che, Dr. Song Jiang, and Dr. Jia Rao for their interest in my research and for their valuable discussion, ideas, suggestions and feedback regarding my early proposal and this thesis. It is a privilege for me to have each of them serve in my dissertation committee.

I want to thank all my colleagues from the system research group at the computer science and engineering department. It is my pleasure to meet such a concentration of creative and nice people here. In particular, our system group weekly seminar not only brings a lot of good foods but also me provides a lot of opportunities to learn the latest system research advancements, which have inspired me to conduct my own research. I am grateful to all with whom I spent my time as a graduate student at UTA.

I also want to thanks the awesome graduate advisors and staffs of the computer science and engineering department, and the advisors of the office of international education, who have provided the kindness support and countless help during my study time.

Finally, my special thanks go to my family. I would like to express my earnest gratitude to my parents for their love and countless sacrifices to give me the best possible education. Without their patience and unreserved support, it would not have been possible to reach this stage in my career. In particular, I will give special thanks to my wife Yujuan Tan who has loved and supported me throughout this thesis and throughout my life. She has spent most of her time to take care of our children and raise them up. Without her sacrifices, I definitely cannot finish this study. Finally, thanks to my children Kathryn and Mark, for the energy and the happiness them bring to me.

November 16, 2018

ABSTRACT

BUILDING A VERSATILE DEDUPLICATION SYSTEM

Zhichao Yan, Ph.D.
The University of Texas at Arlington, 2018

Supervising Professor: Hong Jiang

With the development of the Internet and information technology, a large amount of unstructured data is generated and stored in various storage systems. In particular, data reduction techniques such as compression and deduplication have become an effective way to address the combined challenges of explosive growth in data volume but lagging network bandwidth growth to increase the space and bandwidth efficiency of various storage systems. However, we have found that existing deduplication systems cannot effectively process compressed data and image data because existing deduplication systems only analyze the hash value of the bitstream to detect redundant data. At the same time, we found that it is hard to integrate deduplication in resource-constrained solid-state drives (SSDs) due to their internal structure although it is worthwhile because deduplication not only can expand their logical capacity but also extend their lifetime by reducing the program and erase (P/E) operations. Inspired by these problems, this thesis will focus on building a versatile deduplication system to addressing these issues.

With respect to the problem of deduplicating compressed data, we propose Z-dedup approach, which leverages the existing invariable metadata such as original file's length and checksum within the compressed packages to help detect and eliminate the potential duplicated files across all compressed packages. Moreover, for the complicated solid compression mode, Z-dedup injects such metadata into the solid compressed packages to make their internal contents to be analyzed by our versatile deduplication system.

With respect to the problem of deduplicating image data, we propose WM-dedup approach, which injects an invariable chunking and content description information in the form of a steganographic watermark to help identify and remove the perceptible redundant image data. This is a lossy deduplication scheme that may tolerate some information losses while the super-resolution and impaint techniques can help recover the perceptible equivalent image data to enable deduplicating image data in our versatile deduplication system.

With respect to the problem of efficiently integrating deduplicating in SSDs, we propose SES-dedup, which bypasses the data scrambler module to enable the low-cost ECC-based data deduplication. Specifically, we propose two design solutions, one on the host side and the other on the device side, to enable ECC-based deduplication. Based on our approach, we can effectively exploit SSD's built-in ECC module to calculate the hash values of stored data for data deduplication in our versatile deduplication system.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

In the big data era, the world's most valuable resource is no longer oil, but data [20]. For example, look at the companies with the highest market capitalization in twenty years ago, all of which are energy companies like General Electric and ExxonMobil; in 2018, they are big data companies like Apple, Amazon, Microsoft, Alphabet, and Facebook. In particular, big data are associated with five key concepts: volume, variety, velocity, veracity and value, whose challenges include but not limit to capturing data, data storage, data analysis, and data security. Nowadays, the biggest challenge is how to manage exponential growth data. Moreover, the quantity of newly generating data will continue to outpace the ability to manufacture all kinds of storage devices to store all of the data since 2007 [21] [48]. Fortunately, there exist a lot of redundant contents in the digital universe, which makes the data reduction technology become a much-have feature in those big data storage environment to store more unique data. In this thesis, we work on improving an important data reduction technology, deduplication, and demonstrate how to build a versatile deduplication system that can efficiently store those data previously cannot be processed by the traditional deduplication approaches.

1.1 Problem Description

Data deduplication is a specialized lossless data compression technique that eliminates duplicate copies of repeating data at the file or subfile level. It incurs less computation and dictionary overheads than traditional sliding window compression algorithms, thus making it adopt to detect and eliminate the duplicate chunks across the whole storage system. Nowadays, besides being used to improve storage efficiency, it can also be applied to improve the low-bandwidth network transmission efficiency by reducing the number of bytes that must be sent to the remote receiver.

As shown in Figure 1.1, a deduplication system generally divides the incoming file/stream into a serial of fixed-size or variable-size chunks, which is called as chunking, calculating each chunk's cryptographically secure fingerprint (i.e., collision-resistant hash), which is called hashing, identifies duplicate content by querying its hash signature in the existing hash table, which is called querying, and eliminates redundant data at the

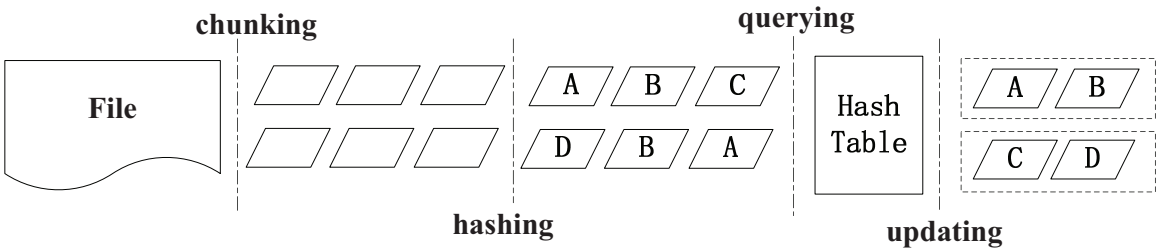

Figure 1.1: Deduplication workflow.

chunk granularity by using the reference pointer to the unique chunk, which is called updating. Finally, each file will maintain the corresponding mapping to locate all of its chunks and each unique chunk will be stored in different containers to optimize the data layout for the future accesses.

The chunking phase calculates the chunk's boundaries. There are two major classes of chunking algorithms, fixed-size chunking and content-defined chunking (i.e., variable-size chunking), while the former obtains a serial of chunks with the uniform length (i.e., 4KB or 8KB) at invisible computation cost but having the potential content-shifting problem, and the latter determines each chunk's boundaries based on its contents to meet the predefined anchor points at non-trivial computation costs. Because the content-defined chunking algorithm can find the most proper content-based anchor points to detect most duplicate contents, so it is often used in latency-insensitive scenarios such as backup and archive storage systems to maximize the deduplication ratio. How to further exploit more anchor points and reduce the computation costs is the main optimization direction for the chunking phase.

The hashing phase calculates each chunk's cryptographically secure fingerprint (i.e., SHA-1 or SHA-256), which is used as the unique identification within the deduplication system. If two chunks have the same fingerprint, we assert they have the same contents without the costly byte-to-byte comparison and can eliminate the duplicated chunk by the salient collision-resistant feature of cryptographic hash, which is the core idea of deduplication. However, there is an inevitable hash collision in the real world, whose probability p can be estimated by formula 1.1, where n is the total number of chunks and b is the length of the fingerprint. In Venti [47], it estimated that an exabyte (10^{18} bytes) data stored as 8 Kbyte chunks (n = 10^{14}) and using the SHA-1 hash function (b = 160), the probability of a collision p is less than 10^{-20}, which is far lower than the hardware error rate. That is why compare-by-hash has been accepted by deduplication systems. NowadaysïijŇ SHA-1 is no longer considered secure enough because it will suffer from practical collision attacks [52],

and SHA-256 with even lower collision probability is widely used in the deduplication systems. In order to guarantee the reliability of deduplication, it is time to think about how to efficiently detect a hash collision and how to upgrade the fingerprint algorithms when the existing algorithms are no longer secure.

$$p \leq \frac{n(n-1)}{2} \times \frac{1}{2^b} \tag{1.1}$$

The querying phase looks up the hash table to check the existence of corresponding fingerprint in the fingerprint index. For each incoming chunk: (1) if its fingerprint can be found, this is a duplicate chunk and can be eliminated; (2) if its fingerprint cannot be found, this is a new chunk and its fingerprint will be added to the fingerprint index. All fingerprints are organized in a key-value store, whose querying latency will directly add to the deduplication system's critical path. How to organize the fingerprint index to exploit the locality in the limited CPU, DRAM and SSD resources is the main research topic of deduplication system.

The updating phase replaces the duplicate chunk with the reference pointer to the corresponding unique chunk and generates all necessary metadata information. Moreover, it organizes the chunks in different containers and implements different policies to optimizing the actual data layout to make the proper trade-off among deduplication ratio, write performance and read performance.

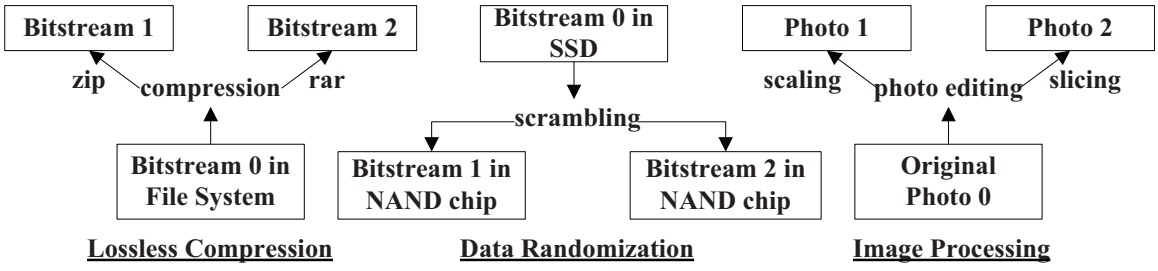

Figure 1.2: Decoupling examples of contents and bitstreams.

Driven by the data backup application, existing deduplication systems have made a lot of optimizations for the above four aspects. There is an implicit assumption, which is a unique content is closely coupled with its bitstream's fingerprint, in all existing deduplication systems. However, as shown in Figure 1.2, there are some common scenarios that contents and bitstreams are decoupled, like lossless compression, data randomization and image processing. For example, when the *bistream 0* is compressed by zip and rar tool

to generate *bistream 1* and *bistream 2*, all these three different bitstreams will share the same content but existing deduplication system cannot identify the duplicate contents. The same thing happens in data randomization and image processing, which will lead to a lot of duplicate contents cannot be processed by existing deduplication systems, thus wasting not only the storage capacity but also the corresponding operational costs associated with the unnecessary redundant data. In order to detect and eliminate these duplicate contents, we propose several approaches that can be used to build a versatile deduplication system.

1.2 Our Technologies

Inspired by the problem that unique content may decouple from its bitstream in some specific applications, which means that there will be many different bitstreams sharing with the same content. In general, a bitstream is the carrier of the content. For most users, the content is valuable data, and the form of the specific carrier is often not as important as the actual content. As a result, an efficient deduplication system should have the capability to detect and eliminate such duplicate contents, which motivate the technologies in this thesis.

First, we propose Z-dedup, an approach that exploits the invariant metadata existing in the compressed packages to help detect the duplicate files among these different compressed packages. It is inspired by our observation that in most compression algorithms, each compressed file will maintain its checksum and original length information that will be used to verify the data integrity on decompressing the file. As a result, this information will be invariant across different compressed packages when they share the same checksum algorithm. Based on this observation, we propose how to leverage this metadata information to help detect and eliminate the compressed files across different compressed packages. Moreover, we not only focus on the simple non-solid compression mode but also propose an extended method to process the duplicate files in the complicate solid compressed packages. In a non-solid compression, each file is compressed and maintains its metadata independently, it is easy to extract such information by parsing the metadata of the compressed packages to detect and remove the potential duplicate files. On the other hand, in a solid compression package, all files are concatenated together and compressed as a whole file. It will only contain the checksum and original length information for this concatenated file. In order to process with this case, Z-dedup suggests injecting the per-file checksum and original length information into the solid compressed packages to help detect the potentially similar contents to determine whether to perform deduplication operation or not. More details about Z-dedup are described in Chapter 2.

Secondly, we propose WM-dedup, an approach that combines injecting and extracting the chunking and content description information in the form of a steganographic wa-

termark to help detect and remove the perceptible redundant image contents. It is inspired by our observation that unlike computers, which can evolve in a short time to acquire more computation power, there are some obvious limitations to human's capability that cannot improve in a short time to perceive information contained in images. In general, users will not find the subtle differences between an image with the steganographic watermark and its original copy. Meanwhile, we also notice that there will be a lot of similar images derived from the same original image by applying some different image processing operations, which may share significant duplicate perceptible contents that can be identified by extracting the steganographic watermark embedded in these images. Moreover, with the advancement in computer vision, it is easy to inject the steganographic watermark in an image and extract it from the image. Once we verified the perceptible equivalent image contents, it is safe to remove the duplicate contents and the super-resolution and impaint technologies can help WM-dedup to recover the perceptible equivalent image to tolerate some data loss without impacting the user's experience to perceive the same content from the image. More details about WM-dedup are described in Chapter 3.

Thirdly, we propose SES-dedup, an approach that bypasses the data randomization modules widely integrated in SSDs to enable the low-cost ECC-based deduplication. It is inspired by our deep analysis on existing SSD's internal architecture, which usually adopts a data randomization (scrambler) module to prevent the very similar data patten writing to the NAND flash chip to significantly increase its raw bit error rate. As a result, the same data will be randomized to different bitstreams based on their logical block addresses, thus resulting in generating different ECCs for these bitstreams. Therefore, it will prevent manufacturer from integrating the low-cost ECC-based deduplication in SSD. However, deduplication will not only enlarge the SSD's logical capacity but also reduce the program and erase operations to NAND flash chips to significantly improve both liability and life time, which make deduplication become an attractive feature. We have explore two design solutions, one is on the host side only and the other is on the device side only, to enable the low-cost ECC-based deduplication in SSD. More details about SES-dedup are described in Chapter 4.

1.3 Thesis Overview

As shown in Figure 1.3, we provide the overview of the versatile deduplication system proposed in this thesis. Besides the existing deduplication approaches, we have proposed three new deduplication appraoches that can help to detect and eliminate the duplicate contents, which existing approaches cannot handle with. In particular, we describe the Z-dedup approach in Chapter 2 to enable the versatile deduplication system process the

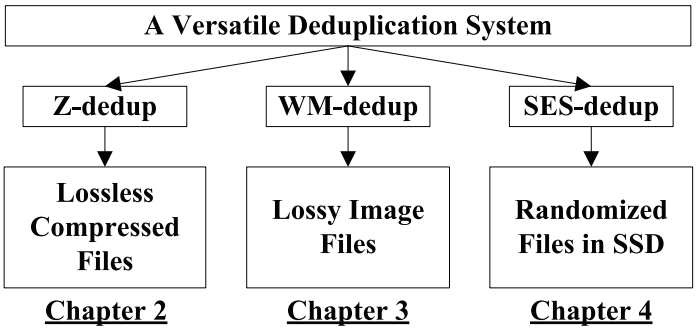

Figure 1.3: Overview of the versatile deduplication system.

lossless compressed files across different compressed packages. In Chapter 3, we describe the WM-dedup approach to show how to perform the lossy chunk-level deduplication on lossy compressed image files. We describe the SES-dedup approach in Chapter 4 to demonstrate how to bypass the randomization module in SSD to enable the low-cost ECC-based deduplication. As the ending, Chapter 5 draws our conclusions of the thesis, where we summarize the presented versatile deduplication system with our three novel approaches and highlight their contributions in practice and provide some future research directions on building the versatile deduplication system.

CHAPTER 2
MOTIVATION

From the perspective of technological development, there are mainly three different trends that motivate our research. First, the ever-wider gap between the data and storage capacity requires the novel data reduction technologies that can help alleviate the deficit of available storage capacity. Secondly, migrating data to the cloud has posed a lot of challenge to existing deduplication approaches that were initially designed for the backup application, whose data type is much simpler than the popular files stored in the cloud. Finally, new and emerging storage devices with different characteristics require necessary co-design mechanism to help deploy deduplication in these new devices at fairly low-cost. In this chapter, we will first review existing technologies, discuss the need to develop new technologies, and summarize our new technologies under the existing data reduction framework.

2.1 Storage Capacity Deficit

Big data are envisioned to deliver value, which makes them be considered the new oil in the big data era. However, one notable difference is that oil either already exists underground or slowly transformed by the resources that exist underground, but the data are generated as users are using new technologies, which is unstoppable, and its growth rate is getting faster with technological advancements such as the Internet, social media, internet of things, etc. In another word, there already exists a big enough container, which is the earth, to store all the oil, but there is no system that can help us store all the data.

Data are ubiquitous around us and become the key component in this information age. It is because the big data science will help us unveil the mysteries of nature, provide detailed insight into processes, and predict events that may not only be useful to the science, but also important to the business. As shown in Figure 2.1, due to the technological limitations of storage technology employed in data centers, there is a significantly increasing gap between storage demand and supply [6].

On the storage demand part, more and more institutions have recognized the importance and feasibility of the fourth paradigm, which was formally proposed by Jim Gray [29], that requires a large amount of big data to be stored before analyzing and ex-

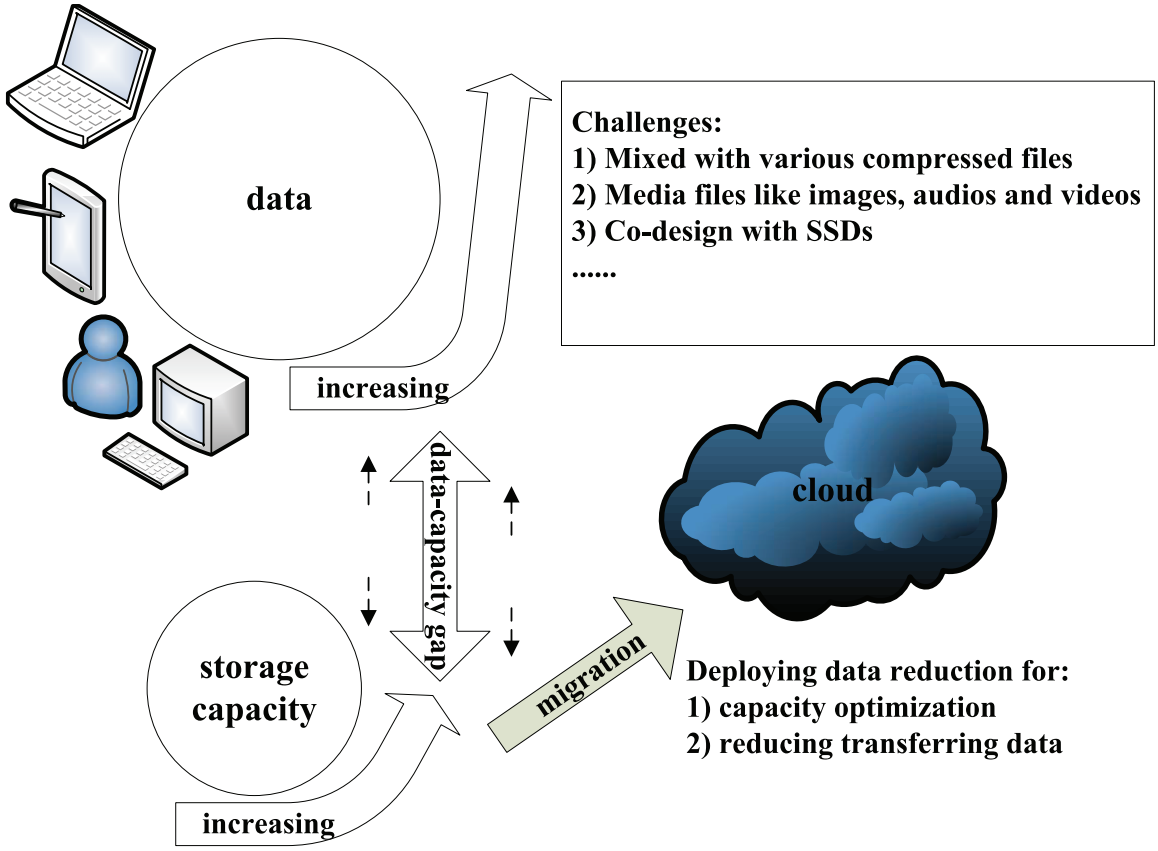

Figure 2.1: Inevitable data-capacity gap.

ploiting the value from these voluminous data, which has posed a big challenge in current and future storage technology to hold such voluminous data. For example, IDC has predicted that the amount of data generated globally will reach 44 zettabytes (ZBs) in 2020 and 163 ZBs in 2025 [48]. Moreover, the available storage capacity could hold around 1/3 of the digital universe in 2013, and it will reduce to 1/7 by 2020. From this report, there is a clear trend that the gap between storage demand and supply is acceleratingly expanding.

Meanwhile, on the storage supply part, because of current HHD's storage density is approaching the super-paramagnetic limit and the repeatedly delayed adopting of advanced technologies such as heat-assisted magnetic recording (HAMR) and bit patterned media (BPM) due to both the technical and commercial issues, which lead to the slowing down the growth in areal density and magnetic storage capacity. Moreover, SSD's capacity density increases around 40% annual growth by incorporating the TLC, QLC and 3D-stacking technologies, however, it will also approach its scaling limitation by 2020 due to the relia-

bility and heating reasons. As a result, we will face the inevitable data-capacity gap in the big data era, and it is important to reduce the volume of storage demand by applying the data reduction technologies.

2.2 Technical Trends

2.2.1 Data Reduction Technologies

Data reduction technologies such as compression and deduplication are typical solutions to storage capacity optimization, which can reduce the volume of storage demand without losing meaningful contents. In most cases, data reduction technologies have the potential to reduce the storage demand by optimizing capacity and reducing data footprint via applying non-trivial computation costs. As a result, it is necessary to analyze the trade-off between capacity savings and extra computation costs.

Compression can be treated as an encoding procedure that use fewer bits to represent the original content. It can be either lossy or lossless, while the former will reduce bits by removing unnecessary or less important information and the latter reduces bits by identifying and eliminating statistical redundancy that is widely used in the storage systems to optimize the storage efficiency. Besides reducing the data footprint, compression can also reduce the random I/O operations that may incur non-trivial disk access overhead on HDDs. From this aspect, lossless compression will reduce disk's energy consumptions and enhance disk's throughput, in addition to reducing the overall disk's capacity demand. However, there is no free lunch. Lossless compression has to encode every possibly longest string within the compression window by its dictionary, which is usually automatically generated by scanning the strings. Besides the computation costs incurred by scanning and encoding the strings, compression dictionary's size is usually proportional to the size of the sliding window and the maximum length of the possible string without considering the statistical characteristic of input files. As a result, current compression algorithms usually find the string containing up to hundreds of bytes to balance the costs among CPU, memory and disk. Unfortunately, lossless compression usually suffers from the low data reduction ratios (around 1:2) that are limited by its feature of only removing the statistical redundancy within a small range.

Nowadays, deduplication is considered to be the most effective data reduction technology because its data reduction ratio is usually around 1:10 in common business workloads. One of its key difference from traditional compression is that deduplication tries to searches for duplicate data and then removes the duplicates at the global range, thus guaranteeing the whole storage system will only have one instance of a particular data chunk. However, its effectiveness is also highly dependent on the workloads. For example, the

reduction ratios are huge for the backup application, while they are smaller for a broader range of applications, which is more likely in the cloud storage environment. Furthermore, besides the necessary metadata information to be maintained by deduplication, deduplication also shifts the burden from network and storage to CPU, which leads to greater computation costs and energy consumptions that constitute an improtant part of the data center's total cost of ownership (TCO).

2.2.2 Migration to Cloud

In the big data era, every institution has to keep the operational data in its information system. For small and medium-sized businesses, their computer systems are becoming more and more complicated, and their maintenance costs are gradually exceeding budgets. As a result, the cloud computing model is proposed to provide a shared pool of configurable computer system resources and higher-level services that can be rapidly provisioned with minimal management effort over the Internet at the "pay-as-you-go" model. Cloud storage is the most popular cloud application that stores data to a cloud computing provider who manages and operates data storage as a service.

There are at least five incentives for migrating data from a local storage system to a cloud storage environment. First, it can reduce the total cost of ownership with the cloud storage because no hardware is needed to purchase while users can add or remove the storage capacity on demand. Secondly, there are no over-provision costs because users will only pay the cost what they actually use. Thirdly, users are saved from the complicated work of maintaining an efficient, reliable and secure storage system. Fourthly, it provides fast deployment service by considering of a lot of other users' experiences. Fifthly, it is easy to interact with other cloud services to analyze big data and exploit their values in a more efficient way.

As more and more data is moved to the cloud, the data in the cloud storage environment presents these four features: 1) there are a lot of file types generated by various applications, 2) data are mixed with various compressed files; 3) redundant data are proportional to the total number of users that use cloud storage serviceïijN̆ thus resulting in a high data redundant ratio on the cloud storage environment, 4) there are a lot of social media files like images and videos. Furthermore, to accelerate the data access, cloud storage system usually have a fast storage layer supported by SSDs. As a result, how to make deduplication system adapt to these new cloud storage characteristics become an interested topic that motivate our work.

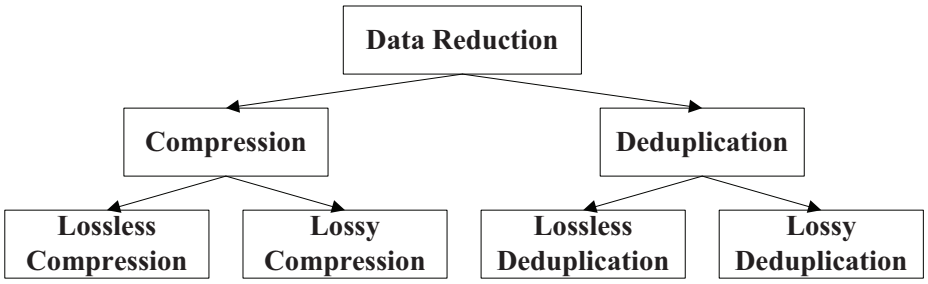

Figure 2.2: Taxonomy of data reduction

2.3 Advancements of Our Technologies

As illustrated in Figure 2.2, we list the taxonomy of data reduction technologies like compression and deduplication. In particular, plaintext files usually adopt lossless compression for data reduction so that they can be exactly restored upon decompression, while multimedia files like images tend to adopt lossy compression to significantly reduce a file's size without noticeable quality degradation but at the cost of not being able to restore to the original files. In fact, data deduplication can be viewed as a special computation efficient way to compress data, where all existing fingerprint-based deduplication methods are lossless data reduction capable of exactly recovering the original files. However, for most multimedia files, lossy data reduction is considered an acceptable tradeoff for significant space savings.

In this dissertation, we first consider how to deal with the duplicated contents compressed by various lossless compression tools. We propose the Z-dedup approach, which is designed for the cloud storage environment where aggregating a lot of compressed packages with huge data redundancy sealed in the compressed packages.

Secondly, we propose to explore a new category of "lossy deduplication" inspired by the concept of lossy compression algorithms, by designing WM-dedup to show how lossy deduplication can be deployed to reduce the redundant data of image files.

Finally, unlike the traditional deduplication work either optimizing its disk access operation on HDD or directly deploying deduplication on SSD, we show a low-cost SES-dedup approach to integrating deduplication on SSD to exploit its internal architecture by reusing its ECC engine and bypassing its data randomization module.

CHAPTER 3

Z-dedup:A Case for Deduplicating Compressed Contents in Cloud

In order to enable deduplication system to detect the potentially redundant contents wrapped by various compressed packages, we propose the Z-dedup approach, which leverages the invariable metadata information to help perform the file-level deduplication across the duplicate contents in different compressed packages.

3.1 Introduction

Information explosion has significantly increased the amount of digital contents in the big data era [26] [21]. For example, the world's technological capacity to store information grew from 2.6 (optimally compressed) exabytes (EB) in 1986 to 15.8 EB in 1993, over 54.5 EB in 2000, and to 295 (optimally compressed) EB in 2007. In fact, we create 2.5 quintillion bytes (EB) of data every day, and 90% of the current world's digital data are created in the last two years alone [57]. These ever-increasing digital contents require a large amount of storage capacity to keep them available for future accesses. Besides the storage capacity requirement, network bandwidth also becomes a bottleneck, and the upload bandwidth is usually an order of magnitude smaller than the download bandwidth. To meet performance, reliability, availability, power, and cost efficiency requirements, cloud storage service is becoming a core infrastructure to host these data. Therefore, how to manage such incredible growth in storage demand has become the most important challenge to the cloud storage service providers [21].

Lossless data reduction techniques, most notably, compression and deduplication, have emerged to be an effective way to address this challenge by improving both space and bandwidth efficiency in the cloud storage environment. We compare compression and deduplication in Table 3.1. Compression finds repeated strings within a specific range of a given file and replaces them with a more compact coding scheme with intensive computations. For example, the deflation algorithm used in gzip checks the repeated strings within a limited $32KB$ window and the longest repeated string's length is limited to 258 bytes [18]. On the other hand, deduplication divides a file into fixed-size (e.g., 4KB or 8KB in fixed chunking) or variable-size chunks (e.g., content defined chunking), identifies (i.e., lookup the hash table) and removes the duplicate chunks across all existing files by com-

Table 3.1: A comparison between compression and deduplication.

	Compression	Deduplication
Granularity	Strings (100s bytes)	Chunks (KBs)
Domain	Within a file	Across all files
Lookup	A sliding window	A fingerprint table
Bottleneck	Computation and Memory	Fingerprint lookup I/O

paring chunks' unique fingerprints (e.g., secure hash values), and reassembles the chunks to serve the subsequent access operations on the corresponding file. Therefore, compression can achieve the best data reduction ratio but at both high computation and memory costs, rendering it most suitable for reducing individual files and data items at small volumes locally, while deduplication can remove redundant data at a much coarser granularity to obtain a good data reduction ratio with a much lower computation cost, thus making it most effective in eliminating much larger volumes of redundant data across files and data items globally. In order to design an efficient cloud storage system, designers need to combine these two techniques to detect and remove redundant data by exploiting both global and local redundancies with acceptable computation costs.

However, existing deduplication approaches cannot detect and remove the redundancy wrapped by different compression approaches because they only rely on bitstream fingerprint-based redundancy identification. As a result, they are unable to identify redundant data between the compressed and uncompressed versions of the exact same contents as the compressed contents, being encoded by a compression algorithm, will have very different bitstream patterns from their uncompressed counterparts. In fact, our extensive experimental study indicates that they also cannot detect any redundant data compressed by different compression algorithms because the latter will generate different compressed data at the bitstream level of the same contents. Furthermore, different versions or input parameters of the same compression tool may generate different bitstreams of the same contents (e.g., different metadata information or compressed bitstream will be embedded in the compressed files), whose redundancy cannot be identified by fingerprint-based detection. Finally, very similar but slightly different digital contents (e.g., different versions of an open source software), which would otherwise present excellent deduplication opportunities, will become fundamentally distinct compressed packages after applying even the same compression algorithm.

In a multi-user cloud storage environment, the aforementioned compression scenarios are arguably widespread, which prevent the cloud servers from detecting and removing any data redundancy among these compressed packages. For example, in Figure 3.1, two

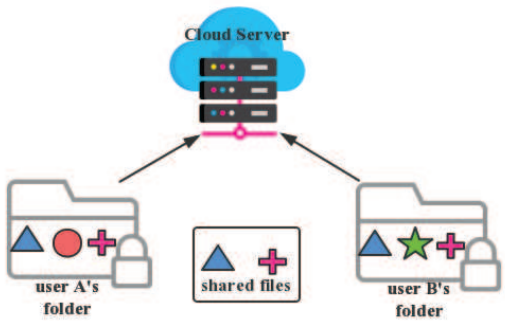

Figure 3.1: A simple cloud storage use case.

different users, A and B, choose a public cloud storage service to store their data and apply compression to reduce the transmission time before sending the data to the cloud. They share substantial contents between their individual local folders, as a result of, for example, taking the same courses, working on the same projects, having similar tastes in movies, music, etc.

(1) User A directly stores his/her folder to the cloud, while user B chooses a compression software (i.e., zip) to compress his/her local folder before storing it to the cloud.

(2) User A sends a package compressed by zip to user B while user B decompresses it, slightly adds/removes some files and re-compresses it by another compression tool such as rar.

(3) User A sends a package compressed by rar to user B while user B decompresses it, slightly adds/removes some files and re-compresses it by his/her rar program whose version number is different from that of user A's rar program.

(4) User A sends a package compressed by zip to user B while user B decompresses it, slightly adds/removes some files and re-compresses it by the exact same zip program as user A but with a different set of compression parameters.

There are more complicated scenarios that chain multiple compression algorithms and pre-processing filters together to gain higher compression ratios [45]. All these scenarios indicate that several factors limit the applicability of deduplication in the cloud storage environment, leading to failures in identifying and removing potentially significant data redundancy across different compressed packages. These factors include compression algorithms, compression parameters, input file stream, etc. In general, two users can generate the same compressed packages only when they compress the same files with the same compression parameters by the same version of a compression tool. Obviously, it is impractical to assign the exact same specific compression mode to all the users in the cloud

storage environment. 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 or other differently compressed files, even though they may have the same semantic contents [54]. If there are 10 compression tools that are widely used for a cloud storage environment, where each has 10 versions and 5 sets of compression parameters, then a given file can in theory have up to 500 different compressed instances in the cloud, yet with no redundancy detectable by conventional deduplication. Moreover, if this file is compressed in conjunction with other different files (e.g., by solid compression detailed in Section 3.2.2.1), we can expect even more compressed instances of this particular file stored in the cloud. This kind of data redundancy is concealed by different compressed packages. As cloud storage is poised to become a digital content aggregating point in the digital universe, 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 data redundancy for a more efficient cloud storage ecosystem.

In this paper, we propose Z-dedup to support deduplicating the compressed data. Recently, compression tools begin to support cryptographic hash function, such as MD5, SHA-1, RIPEMD-160, SHA-256, SHA-512 or BLAKE2, as their checksum generators. Z-dedup extends the data format of each compressed package to incorporate the *SHA-256* checksum information, which makes it possible to be integrated in existing file-level deduplcation systems. Meanwhile, Z-dedup adds an ownership verification module by piggybacking some extra verification metadata for the client side deduplication to address the security threat in which a client might upload fake metadata to illegally obtain data from the cloud server. Our experimental results show Z-dedup can significantly improve both space and bandwidth efficiency over traditional approaches by eliminating up to 98.75% more redundant data among compressed packages.

3.2 Motivation

3.2.1 Background

There are two main motivations for both users and cloud storage providers to compress their data, leading to more compressed packages in the cloud. One is to reduce the amount of data actually stored in the system and the other is to reduce the time spent transferring a large amount of data over the networks. While the former helps reduce cost, both hardware and operational, the latter improves performance, both upload and download, and network bandwidth utilization.

Table 3.2: Broadband speed greater than 10 Mbps, 25 Mbps and 100 Mbps (2015-2020) [17].

Region	>10 Mbps		>25 Mbps		>100 Mbps	
	2015	2020	2015	2020	2015	2020
Global	53%	77%	30%	38%	4%	8%
Asia Pacific	53%	83%	30%	52%	4%	8%
Latin America	27%	39%	10%	15%	1%	2%
North America	64%	88%	38%	52%	5%	9%
Western Europe	54%	74%	32%	43%	5%	11%
East-Central Europe	58%	83%	33%	41%	3%	6%
Middle East & Africa	17%	20%	7%	8%	0.3%	1%

3.2.1.1 Cloud Storage in Face of Slow Networks

Studies by IDC and EMC have shown a continuously expanding, increasingly complex digital universe [26] [21]. For example, the size of the digital universe will grow from 130 exabytes to $40,000$ exabytes from 2005 to 2020. Within these 16 years, the size will increase by a factor of 300. On a per capita basis, there will be about $5,200$ gigabytes digital contents for each person on earth by 2020.

Table 3.2 lists the percentages of broadband connections faster than 10 Mbps, 25 Mbps, and 100 Mbps by region in 2015 and 2020, indicating that the global broadband speeds are still quite low considering that each person accounts for a considerable amount of digital contents. Moreover, upload and download bandwidths are usually asymmetric, with the former being one order of magnitude smaller than the latter. As a result, network bandwidth remains one of the main limiting factors in using cloud storage.

3.2.1.2 Compressions Are Increasingly Common

In order to deal with the incredible digital explosion, the application of data compression technologies, especially lossless data compression, is expected to become a commonplace throughout the life cycle of digital contents. In fact, a lot of systems and applications already perform compression and decompression without the users even being aware that it has occurred. However, from a storage perspective, this also means that there will be increasingly more compressed packages as more compressions of different formats and algorithms are performed.

Table 3.3 shows the compiled results by Calicrates Policroniades and Ian Pratt [46] of popular files' extensions and their relative popularity in an Internet server, indicating that a significant amount of different compressed contents occupy most of the storage space.

Table 3.3: Data profile of sunsite.org.uk [46].

Rank	Popularity		Storage Space	
	Ext.	% Occur.	Ext.	% Storage
1	.gz	32.50	.rpm	29.30
2	.rpm	10.60	.gz	20.95
3	.jpg	7.54	.iso	20.40
4	.html	4.83	.bz2	6.26
5	.gif	4.43	.tbz2	5.65
6	-	4.16	.raw	4.44
7	.lsm	3.74	.tgz	2.66
8	.tgz	2.90	.zip	2.53
9	.tgz2	2.35	.bin	2.00
10	.Z	2.12	.jpg	0.94
11	.asc	1.84	.Z	0.65
12	.zip	1.59	.gif	0.43
13	.rdf	1.39	.tif	0.31
14	.htm	1.21	.img	0.21
15	.o	1.06	.au	0.19
Total	-	82.26	-	96.92

This is consistent with our observation and the reason lies in the fact that the network bandwidth, particularly of wide-area networks, has become much more expensive than computing resources, leading more and more data being compressed before transmitting to the cloud. Besides saving the storage space, compression also converts a lot of random I/O operations to some sequential I/O operation when transferring a lot of small files to the cloud.

With the increasing numbers of compressed packages of different formats in the cloud, there will be even greater numbers of redundant files concealed in different compressed packages. How to organize and store these compressed packages becomes a challenge as we face a mounting demand to further reduce storage capacity requirement in light of the explosive growth of digital contents.

3.2.2 Data Compression and Deduplication

3.2.2.1 Dictionary-Based Compression

We focus on dictionary-based compression, especially the sliding dictionary compression algorithms (LZ77 and LZ78), which were proposed by Abraham Lempel and Jacob Ziv [63] [64]. These two algorithms form the basis for a plethora of their variations,

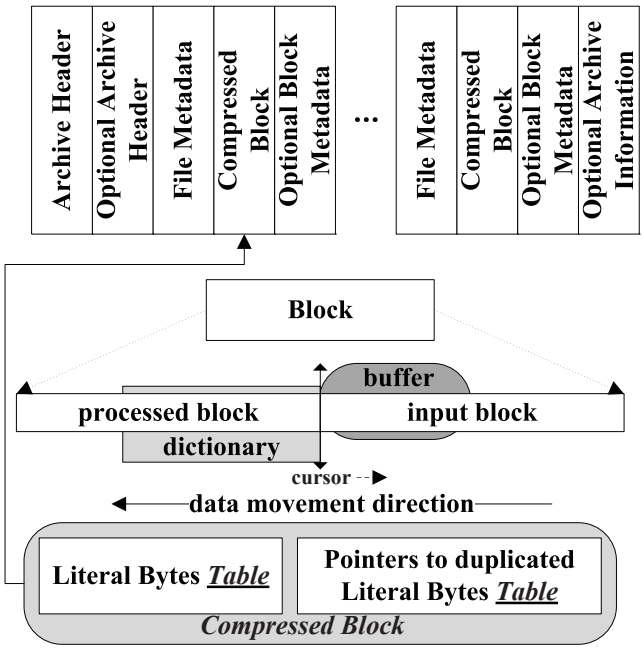

Figure 3.2: Sliding dictionary compression.

including LZW, LZSS, LZMA and others, that are collectively referred to as the LZ-family algorithms. As a result, dictionary-based compression has become the *de facto* standard for general-purpose data compression in almost every computer system [44].

By their initial designs, compression tools only compress individual files because the compression algorithm only focuses on a single input data stream. Then some compression tools allow for the compression of a directory structure that contains multiple files and/or subdirectories into a single compressed package. There are two common ways to compress the directory structure, namely, compressing each file independently and concatenating a group of uncompressed files into a single file for compression. The former is called *non-solid compression* and widely supported by the zip format [33], while the latter is called *solid compression* and natively used in the 7z and rar formats [45] [50]. Solid compression is also indirectly used in tar-based formats such as .tar.gz and .tar.xz, where users archive a directory containing multiple files first as a single file that is then compressed. In this paper, we only focus on these two common compression schemes while considering out of scope schemes that chain multiple compression algorithms and filters to gain higher compression ratios.

As shown in Figure 3.2, a compressed archive is organized in a data package consisting of an archive header, an optional archive header, several compressed blocks with the corresponding file metadata and optional block metadata, and optional archive information.

Archive header maintains the basic information on this compressed package. Block is the basic, atomic unit of compression, which can be of variable size and contain either some parts of a file or some files. Optional archive information contains the necessary information about directory's structure. A cursor will scan the input block and some parts of an already processed block to constitute a dictionary window that is used for the subsequent read buffer to find the longest repeated literal bytes. This process will generate an adaptive dictionary, as illustrated in Algorithm 1. After the cursor has scanned a whole block, it will generate another table containing literal bytes and a table containing pointers to duplicated literal bytes. Finally, these two tables will be compressed into a compressed block.

Algorithm 1 Adaptive dictionary compression.

```
while Not the end of block do
  Word = readWord( InputFile )
  DictionaryIndex = lookup( Word, Dictionary )
  if (DictionaryIndex IS VALID) then
    output( DictionaryIndex, OutputFile )
  else
    output( Word, OutputFile )
    addToDictionary ( Word, Dictionary )
  end if
end while
```

3.2.2.2 Deduplication

Deduplication is a special kind of lossless data compression, which aims to detect and remove the extremely long repeated strings/chunks across all files. It becomes popular in storage systems, particularly in backup and archiving systems, because of the datasets in these systems have significant amount of duplicative data redundancy.

Deduplication's basic idea is to divide the data stream into independent units, such as files or data chunks, and then use some secure hash values of these units, often referred to as fingerprints, to uniquely represent them. If the fingerprints of any two units match, these two data units are considered duplicate of each other. With this compare-by-hash approach, duplicate contents within a single storage system or across multiple systems can be quickly detected and removed.

Obviously, deduplication relies on secure hash algorithms to limit the hash collision rate to avoid false positives, where units of different data contents produce the same hash

values and thus are mistakenly considered duplicates. It also adds some extra operations, such as hash computation and hash index table lookup, to the I/O critical path. Another key issue of deduplication is its heavy reliance on the format of the digital content itself for the hash value calculation. For example, two identical files in different formats (e.g., due to compression or encryption) will generate essentially different hash values, completely preventing their data redundancy being detected and removed.

3.2.3 Research Problems

In above analysis, more and more compressed archives are expected to be stored in the cloud. In fact, some systems, such as ZFS and Flash Array, have already integrated compression and deduplication [10] [53]. Meanwhile, the cloud will be a convergence point to which increasing numbers of end users and systems upload their compressed packages. Within these compressed archives, there can be a great number of redundant files across different users, particularly if they happen to have similar interests in certain specific topics. However, due to the different ways in which compressed archives are generated, which can substantially hide the data redundancy detectable by the conventional deduplication technology, these problems arise when dedupication interplays with compression to motivate this work. First, how to construct a fingerprint for each compressed file across the different compressed packages to help detect redundant files? Second, how to remove such redundant files compressed in different packages? Third, how to integrate it into existing data deduplication system to detect and remove redundant data between compressed packages and uncompressed files? Fourth, how to deduplicate the redundant files among solid compression packages?

3.2.4 Related Works

Deduplication was proposed to remove redundant data in backup application [62]. As more and more data migrates to the cloud, it has been integrated within the cloud platform [56]. 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 [42]. 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 bitstream itself without touching its internal information. Migratory compression [40] and Mtar [39] try to reorganize data to improve space efficiency. X-Ray Dedup [60] has correctly identified the problem of the hidden redundancy among compressed packages and proposed a preliminary solution that combines

file's size and CRC-32 metadata as its signature to detect and remove potential redundant files. However, it fails to provide a complete and generalizable system design. More importantly, it lacks an in-depth and comprehensive study of the collision problem and completely ignores the security issue. For example, we find that the 32-bit CRC checksum is not sufficient to avoid the collision for a large-scale system. By the birthday paradox theory, the collision rate will be 0.1% when there are 1.9×10^8 different files stored in the system. Z-dedup can scan and extract metadata information to further help identify and remove redundant files across the compressed files and uncompressed files.

3.3 Design and Implementation

3.3.1 Main Idea

A compression algorithm will encode an input stream into an essentially different output stream, which makes it difficult to generate a simple but unique indicator for mapping between the input stream and the output stream. However, a critical observation we gained through our analysis is that most compression algorithms will pack files' metadata information within the compressed packages and, importantly, some metadata, such as original file's size and checksum, will be invariant across all compressed packages. Therefore, the main idea here is to combine this file-level invariant information as a new and unique indicator for the mapping to help detect redundant files across all the compressed packages globally.

This approach can be integrated into existing deduplication systems to help remove redundant files if these files are compressed in the non-solid compression method (Section3.2.2.1). For solid compression, we can leverage this feature to estimate the content similarity with any existing compressed packages. If the incoming compressed package is considered sufficiently similar to existing compressed packages, meaning that many (but not all) of the individual files within the packages are possibly identical to the files in existing stored compressed packages, then the system can decide to deduplicate by first uncompressing and then detecting and removing redundancy in the conventional way. This approach helps data deduplication system quickly find out the solid compressed packages that contain significant data redundancy. Otherwise, the system will either ignore deduplicating solid compressed packages or uncompressing every solid compressed package to perform deduplication work.

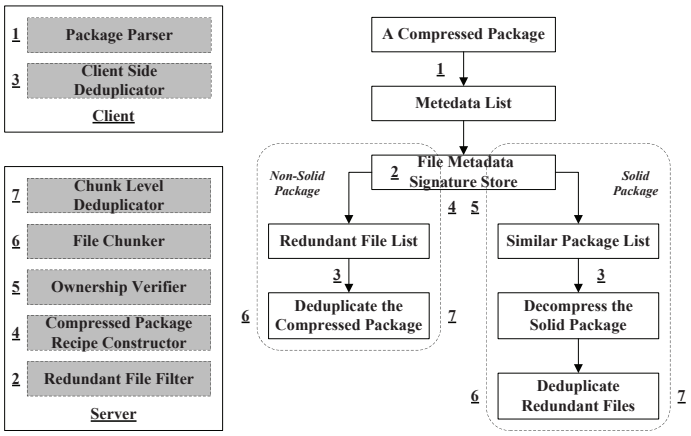

Figure 3.3: Z-dedup's main modules and workflow.

3.3.2 System Design and Implementation

3.3.2.1 An System Overview of Z-dedup

In Z-dedup, once a compressed package is ready to be uploaded to the cloud, the client and the cloud server will exchange some metadata information and perform deduplication at the client side, which will generate a deduplicated compressed package. This scheme not only decreases the cloud capacity requirement, but also reduces the upload time from the client to the cloud server.

Figure 3.3 shows 7 main functional modules of Z-dedup, each enclosed by a dashed gray box. The numerical numbers next to them both label them and indicate the order in which they process the incoming compressed package. *Package Parser* is responsible for parsing each compressed package to extract its file-level signature metadata list. *Redundant File Filter* detects duplicate files by looking up the file metadata signature store, and for solid package, it also helps query its similar package list. *Client Side Deduplicator* is responsible for either removing compressed files in a non-solid package or deciding whether or not to decompress a solid package and duplicate redundant files at the client side. *Compressed Package Recipe Constructor* books the necessary information for recovering a deduplicated compressed package back to its original format. *Ownership Verifier* verifies the client's ownership to prevent an attacker from illegally accessing the file by uploading a fake metadata list. *File Chunker* and *Chunk-level Deduplicator* perform conventional chunk-level deduplication.

3.3.2.2 Deduplicating Non-Solid and Solid Packages

Figure 3.3 also describes an overview of how Z-dedup handles different packages generated by the non-solid or solid compression method to remove the redundant files. We list the main functional modules' labels (numbers) next to the structure maintained by Z-dedup to indicate the interactive relationship between a particular functional module and its different operational steps. More details are discussed in Figure 3.4.

For the non-solid compression, each individual file is compressed independently. As a result, a non-solid compressed package has already embedded sufficient metadata information (i.e., each file's original length and checksum information) to easily locate a specific compressed file within the compressed package. Z-dedup can directly detect and remove the redundant files within such a compressed package. On the other hand, for the solid compression scheme, multiple files are first combined into a single file by concatenating them in a certain order, then compression is performed on the single file to store the compressed stream into a compressed block. As such, a solid compressed package lacks adequate information to locate a specific compressed file within its compressed block because multiple files are scattered across the compressed block and boundaries between files are blurred at best. As a result, Z-dedup can not directly detect and remove the redundant files within the solid compressed package.

In Z-dedup, we inject metadata information (i.e., each file's original length and checksum information) per file into each compressed block, as detailed in Section 3.3.2.4, so as to help estimate the content similarity between any two different compressed packages by the percentage of the shared redundant files, (e.g., the higher the percentage the more similar to the two packages are to each other). The similarity detection in Z-dedup entails estimating the likelihood of finding many duplicate files between the incoming package and those already stored in the cloud. Since extracting and separating the compressed stream of a specific file from a solid package incur very high overheads, Z-dedup's similarity detection of solid compressed packages aims to ensure that an incoming solid compressed package is uncompressed only if it is likely to contain a sufficient number of duplicate files.

3.3.2.3 Workflow and Key Operations

Figure 3.4 illustrates an example of deduplicating and restoring a non-solid compressed package without considering the security issue that is discussed in Section 3.3.3.2. Initially, two non-solid compressed packages, P1 and P2, are stored in the cloud server, and their invariant metadata is stored in the *File Metadata Signature Store*. Now, a non-

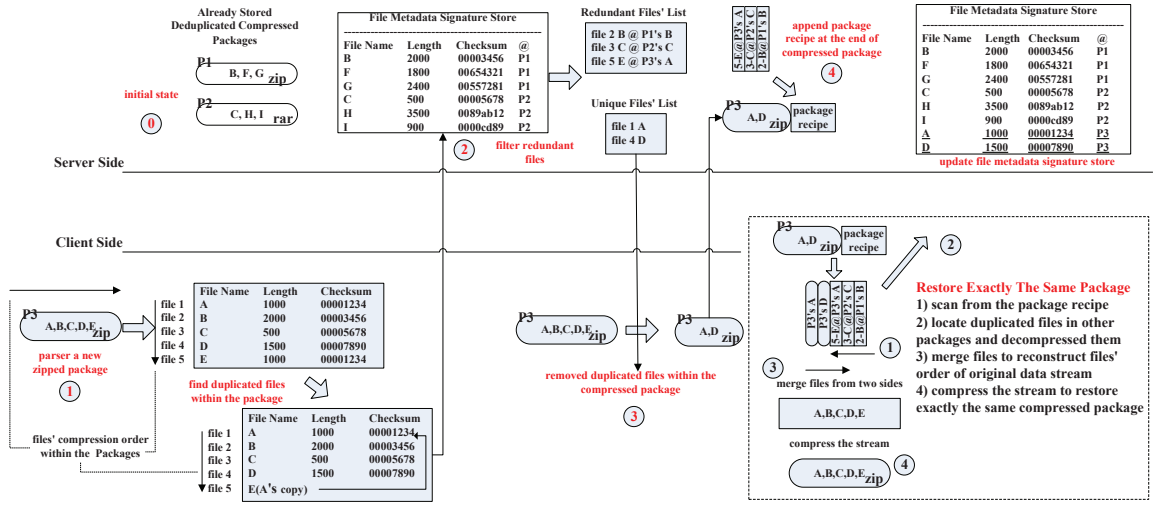

Figure 3.4: An example on a non-solid package's deduplication and restoration.

solid package P3 is ready to be stored to the cloud. P3 is parsed by the *Package Parser* to extract its files' metadata. Moreover, it also detects the locally redundant file (i.e., E) within P3. Meanwhile, files' compression order is implicitly maintained by the *Metadata List*. Based on *File Metadata Signature Store*, the *Redundant File Filter* can divide files into two different groups: redundant files' list (i.e., B, C and E) and unique files' list (i.e., A and D), where the former also records each file's order and pointer of the corresponding file. The *Client Side Deduplicator* will remove duplicated files to generate a *Deduplicated Compressed Package* that only contains A and D. At the server side, the compressed package recipe is generated and appended to the end of package by the *Compressed Package Recipe Constructor*. Finally, it will update the *File Metadata Signature Store* to incorporate P3's metadata information.

In order to restore a deduplicated non-solid package, Z-dedup must reassemble a package whose files' order is the same as the original compressed package. It will scan from both sides of a deduplicated compressed package, in which the left side contains compressed unique files within itself and the right side contains the order and pointer information of the redundant files. These redundant files can be decompressed from other packages. Then based on files' order information, the original data stream will be reconstructed, and then compressed by the original set of compression parameters to restore the originally uploaded non-solid compressed package. We have summarized the main restoration process in Algorithm 2.

Now we assume that the incoming package P3 is a solid compressed one, whose file-level metadata information is already injected at the end of the package as described

in Section 3.3.2.4. Once the redundant files' information is obtained, Z-dedup will decide whether to remove the redundant files or not. This is done by estimating how much redundant data is contained within the incoming package P3 after querying the already stored contents in the cloud (Section 3.3.2.2). Based on the characteristics of source code packages, which are chosen to evaluate the effectiveness of Z-dedup approach in handling the solid compression scenario, we find that it is simple but effective to find the most similar solid compressed packages to help remove the redundant files (i.e., find the most recent source code package to help remove redundant file). However, there is no way of quickly locating the compressed stream of a particular file in a solid package. Different from directly removing the redundant files within a non-solid package, once we have verified that the incoming solid package contains sufficient redundancy with the existing contents stored in the cloud, we need to decompress the solid package to locate and remove the redundant files first and then re-compress the remaining content into a solid package to send it to the cloud server. Moreover, it also maintains the concatenation order of compressed files in the original solid package. During the restore process, similar to the non-solid case, Z-dedup will first restore the file stream and then solid compress these files to restore the original solid package.

Algorithm 2 Restore a deduplicated compressed package.

```
FileCursor = 0
while FileStream is NOT FULL do
    Left = decompressFile(Package[LeftIndex])
    Right = findRedundantFile(Package[RightIndex])
    if (toMerge(Package[RightIndex]) is LEFT) then
        FileStream[FileCursor++] = Left
        LeftIndex++
    else
        FileStream[FileCursor++] = Right
        RightIndex++
    end if
end while
Parameter = getCompress(Package)
RestoredPackage = Compress(FileStream, Parameter)
```

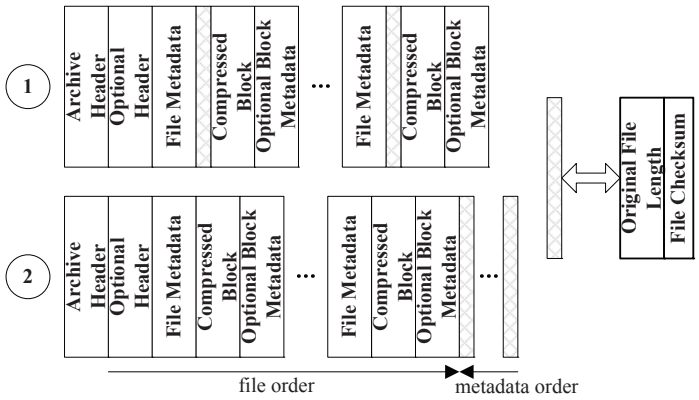

Figure 3.5: Extending data layout of a compressed package.

3.3.2.4 Extending Compressed Packages' Layout

In most compression algorithms, files' metadata information such as the sizes of the original files and their checksums is already packed within the compressed packages for the purpose of data integrity. In Z-dedup's implementation, we choose the combination of a file's SHA-256 value and its size to construct the file's signature. For those compressed packages that do not pack this information, we need to extend their data layout to record this information. It can be implemented by adding an optional feature to the compression tools to calculate and store this information in the compressed packages.

As shown in Figure 3.5, there are two different ways to pack this information. One is inserting it to file's metadata slot for each compressed file, which is similar to the structure of some compression algorithms that have already packed such metadata in their packages. The other is appending it at the end of a compressed package with a reverse order of the input stream. We choose the second way to pack files' metadata in this paper because it does not incur extra data movements within a compressed stream, which is compatible with existing compressed packages. Moreover, at the server side, each deduplicated compressed package will append its package recipe at the end of the deduplicated compressed package to help recover the compressed package, where more details are given in Section 3.3.2.1.

3.3.3 Other Design Issues

3.3.3.1 Collision Analysis

Data deduplication requires a hash function that can generate a unique fingerprint for each unique data block. Suppose that a system contains N different files where each file is represented by a b-bit hash value with a uniform distribution, the probability p that there

Table 3.4: Collision rates for fingerprints of various lengths.

fingerprint		number of hashed elements such that {probability of at least one hash collision $\geq$ p}							
checksum	length	p=10^{-18}	p=10^{-15}	p=10^{-12}	p=10^{-9}	p=10^{-6}	p=0.1%	p=1%	p=25%
32	64	6.1	1.9×10^{2}	6.1×10^{3}	1.9×10^{5}	6.1×10^{6}	1.9×10^{8}	6.1×10^{8}	3.0×10^{9}
64	96	4.0×10^{5}	1.3×10^{7}	4.0×10^{8}	1.3×10^{10}	4.0×10^{11}	1.3×10^{13}	4.0×10^{13}	2.0×10^{14}
128	160	1.7×10^{15}	5.4×10^{16}	1.7×10^{18}	5.4×10^{19}	1.7×10^{21}	5.4×10^{22}	1.7×10^{23}	8.5×10^{23}
160	192	1.1×10^{20}	3.5×10^{21}	1.1×10^{23}	3.5×10^{24}	1.1×10^{26}	3.5×10^{27}	1.1×10^{28}	5.6×10^{28}
224	256	4.8×10^{29}	1.5×10^{31}	4.8×10^{32}	1.5×10^{34}	4.8×10^{35}	1.5×10^{37}	4.8×10^{37}	2.4×10^{38}
256	288	3.2×10^{34}	1.0×10^{36}	3.2×10^{37}	1.0×10^{39}	3.2×10^{40}	1.0×10^{42}	3.1×10^{42}	1.6×10^{43}
384	416	5.8×10^{53}	1.8×10^{55}	5.8×10^{56}	1.8×10^{58}	5.8×10^{59}	1.8×10^{61}	5.8×10^{61}	2.9×10^{62}

will be at least one collision is bounded by the number of pairs of blocks multiplied by the probability that a given pair will collide, which is $p \leq \frac{N(N-1)}{2} \times \frac{1}{2^b}$ [47].

Table 3.4 shows our estimated lower bounds on the numbers of hashed elements with different collision rates under various lengths of the checksum. For example, we can expect that, if the total number of unique files (i.e., hashed elements) is around several millions and the fingerprint is of 64 bits in length (32-bit chechsum plus 32-bit file length), the collision rate will be less then 10^{-6}. From this table, we can deduce that in a large-scale system, we might need to choose a checksum value whose length is larger than 128. In Z-dedup, we propose to store files' SHA-256 checksum information at the end of each compressed package, which is explained in Section 3.3.2.4. The extra space overhead is negligible as explained in the example next. Assuming that the average file size is 32 KB with an average compression rate of 4, a 288-bit signature only adds 0.45% extra space overhead to the compressed package. Moreover, we can integrate the file metadata signature store into the existing lookup table of deduplication systems.

3.3.3.2 Security of Client Side Dedup

Z-dedup performs deduplication at the client side without transferring all compressed data to the server side, which exposes a common security vulnerability in which an untruthful client may submit a faked metadata list to mislead the cloud server into believing that it owns the specific duplicated files already stored in the cloud server, enabling it to illegally access such duplicated files.

In order to overcome this security problem, the *Ownership Verifier* is added to verify if the client really owns the duplicated files as indicated by its metadata list. The key idea is to test whether the client really has the digital contents of the redundant files by asking the client to send a randomly selected sample segment of each claimed redundant file as follows. In Step 2 of Figure 3.4, when a server has identified redundant files that are claimed by a client's metadata list, it will randomly generate an address range within the size of each redundant file, i.e., a randomly selected sample segment of the file, and send

this address information to the client. Meanwhile, it will decompress the corresponding sample data of the redundant files at the server side. In Step 3 of Figure 3.4, in addition to performing client side deduplication, the client needs to decompress the corresponding sample data of the redundant files specified by the sample segment address information, and send this data to the cloud server to verify the ownership of the corresponding files. A last step is to compare the sample data at the server side, where a match indicates the true ownership of the client and a fraudulent claim to ownership otherwise. Once the client has passed this test, its deduplicated compressed package will be accepted by the server.

3.4 Evaluation

3.4.1 Experimental Environment

We use three desktops to simulate a cloud storage environment, with details listed in Figure 3.1. Two different clients run under $Ubuntu14.04$ and $Windows7$ with EXT4 and NTFS file systems respectively. Both have installed some common compression tools that are listed in Table 3.5. One server runs under $Ubuntu16.04$ with all necessary compression tools to access the compressed packages received from different clients. All these three machines are equipped with one $i7-6700K$ processor, four 16GB DDR4 main memory, and one 6TB WD black HDD. We use Destor [24] as the traditional chunk-level deduplication engine. It is designed for backup applications with various chunk-level data deduplication algorithms. We have added a file-level deduplication process to implement Z-dedup in this simulated environment.

Due to the privacy issue, we are not able to collect any real users' data in the cloud storage environment. In order to proof our Z-dedup concept, we have collected several publicly available source code packages (i.e., coreutils, gcc, Linux kernel and Silesia corpus) plus some binary files, which include jpg, pdf, mp3, ppt, doc, and exe files, to form our datasets to test the effectiveness of integrating Z-dedup into an existing deduplication-based storage system to detect and remove the data redundancy concealed by differently compressed packages. The reason why we collect these binary files is because these source code packages have a large number of small files, lack large files and all are represented by English characters, whose features make it suitable for them to be compressed into a solid compression package. Moreover, we randomly select some binary files to synthetically generate a compressed package workload (syn_data) to represent the scenario of multiple users sharing various types of files compressed by different tools. These binary files are usually used to comprise the lossless data compression corpora, which include various types of files in different sizes, to evaluate the effectiveness of compression algorithms. We

Table 3.5: Compression tools in the two client sites.

	tar	gzip	xz	zip	7z	rar
windows 7	1.28-1	1.6	5.2.2	3.0	15.09β	5.31
ubuntu 14.04	1.27.1	1.6	5.1.0α	3.0	9.20	4.20
ubuntu-16.04	1.27.1	1.6	5.1.0α	3.0	9.20	5.30β2

Table 3.6: Sizes (KiB) of different compression formats under the Ubuntu and Windows platforms.

	coreutils-8.25	gcc-5.3.0	linux-4.5-rc5	silesia corpus
tar	49990/49990	601480/601480	642550/642550	206980/206990
xz	5591/5591	73815/73815	86287/86287	48047/48054
gz	12784/12784	121432/120154	132608/132609	66636/66640
7z	6169/5723	72256/71434	93561/89437	48563/48279
rar	12402/12401	148255/148177	156310/155135	53454/53451

use several common compression tools to translate these compressed packages into various compression formats, which include both non-solid and solid compression schemes.

3.4.2 Results and Analysis

In this section, we evaluate data redundancy among different compression formats, estimate how much redundant data can be deduplicated by the Z-dedup approach, and evaluate the overheads incurred by Z-dedup.

3.4.2.1 Analysis on Compressed Content

Table 3.6 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. Especially, rar generates the non-solid compressed packages by default while 7z generates the solid compressed packages. Solid compression has achieved significantly higher compression ratios than non-solid compression. Moreover, source code packages have gained much higher compression ratios than binary files.

We use a chunk-level data deduplcation engine to study data redundancy among the compressed packages. As shown in Table 3.7, we find that nearly all pairs of compressed packages have 0% data redundancy between them, a few pairs have 0.05%-7.63% data re-

Table 3.7: 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

dundancy. The only exception is "tar.xz", which has generated 100% identical compressed packages from both the Ubuntu and Windows platforms, indicating xz-5.2.2 and xz-5.1.0α share the same core algorithm. We further verify that these packages share 0% redundancy with the compressed packages generated by xz-5.0.8. Although most packages have similar sizes across the two platforms, our study shows that: (1) except for "tar.xz", the compressed packages are fundamentally different from one another even under the same compression algorithm; (2) for the same digital contents, different compression 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*.

3.4.2.2 Hidden Content Redundancy in Compressed Packages

In order to evaluate the real data redundancy among different compressed packages, we decompress various versions of compressed packages and apply the chunk level duduplication engine on them. As shown in Figure 3.6, 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. 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. *All these results indicate that there are a lot of duplicate files within these compressed packages, which can be detected and removed by Z-dedup*.

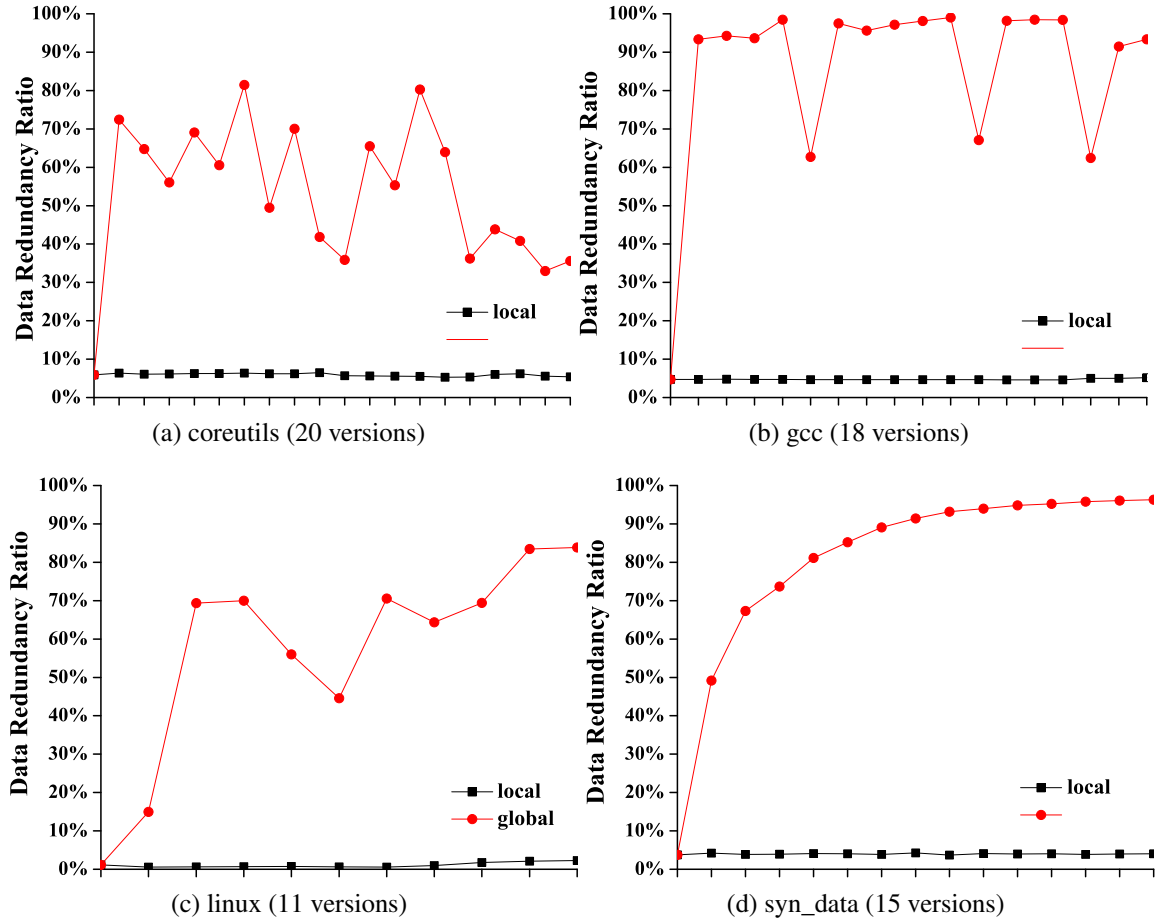

Figure 3.6: Real data redundancy throughout different versions of decompressed packages.

3.4.2.3 Performance of Z-dedup

File level redundancy ranges from 1.39% to 26.46% in coreutils, 95.21% to 97.55% in gcc, 9.04% to 55.55% in Linux, and 48.50% to 90.75% in syn_data. Z-dedup is designed to identify and remove this kind of hidden redundant data. Our experiments show that Z-dedup is able to reduce the size of compressed packages by 1.61%-35.78% in coreutils, 83.12%-98.75% in gcc, 11.05%-65.59% in Linux and 38.28% to 84.25% in syn_data with its file-level data deduplication. We find that some coreutils versions have major modifications made to most files, leading to a very low file-level redundancy. As a result, Z-dedup can scan an incoming compressed package, to estimate the level of redundancy in the package by checking its metadata information and opt out performing file level deduplication when there is very little file level redundancy for this package. However, extracting metadata information from these low-redundancy compressed packages is still necessary and in

Table 3.8: Various checksum algorithms' computation overheads with Intel 6700K processor.

Algorithms	CRC32	MD5	SHA-1	SHA-256	SHA3-256	RIPEMD-160	RIPEMD-256	BLAKE2b
MiB Per Second	610	742	695	341	453	302	584	1128
Cycles Per Byte	6.3	5.1	5.5	11.2	8.4	12.6	6.5	3.4

fact important because there could be high file-level redundancy in the compressed packages of the subsequent versions.

After evaluating the effectiveness of the Z-dedup approach, we want to evaluate the extra overheads that come with this approach. In Table 3.8, we obtain the computation overheads of several popular hash algorithms used to generate the checksum values within the compressed packages with Intel's 4GHz 6700K processor. We find that SHA-256 incurs $1.79\times$ computation overhead than the traditional CRC32 algorithm. By choosing SHA-256 as the checksum algorithm for Z-dedup, we can also reuse the existing fingerprint table, which contains the SHA-256 hash values, in existing data deduplication systems.

We use the source code packages that contain a large number of small files to evaluate the overheads on maintaining and processing the necessary metadata in the Z-dedup approach. In general, using a source code package as a target to simulate deduplicating a non-solid compressed package can incur the worst overheads in deduplicating compressed contents because it requires a number of operations proportional to the number of files in the package but saving very little space because all the files are small in size. At the same time, we use the synthesized packages that contain various types of large size files to more realistically evaluate the overheads on deduplicating non-solid compressed packages. We find that Z-dedup will slightly increase the size of compressed packages because it needs to inject the SHA-256 checksums within the compressed packages. In fact, this space overhead is found to be less than 0.45% of the total compressed package's size. *All these results indicate that Z-dedup can reduce a significant amount of redundant data in compressed packages in a traditional deduplication-based storage system without significant overheads*.

3.4.2.4 Network Traffic Reduction

We show the amounts of data transferring to the server side under different workloads in Figure 3.7, while each data is normalized to its corresponding uncompressed package's size on both non-solid and solid compression modes to learn the benefits of data reduction for network transmission. In particular, the 7z and rar compressions will reduce the amount of coreutils data to 10.69% and 23.76% of their original sizes in geometric average, while

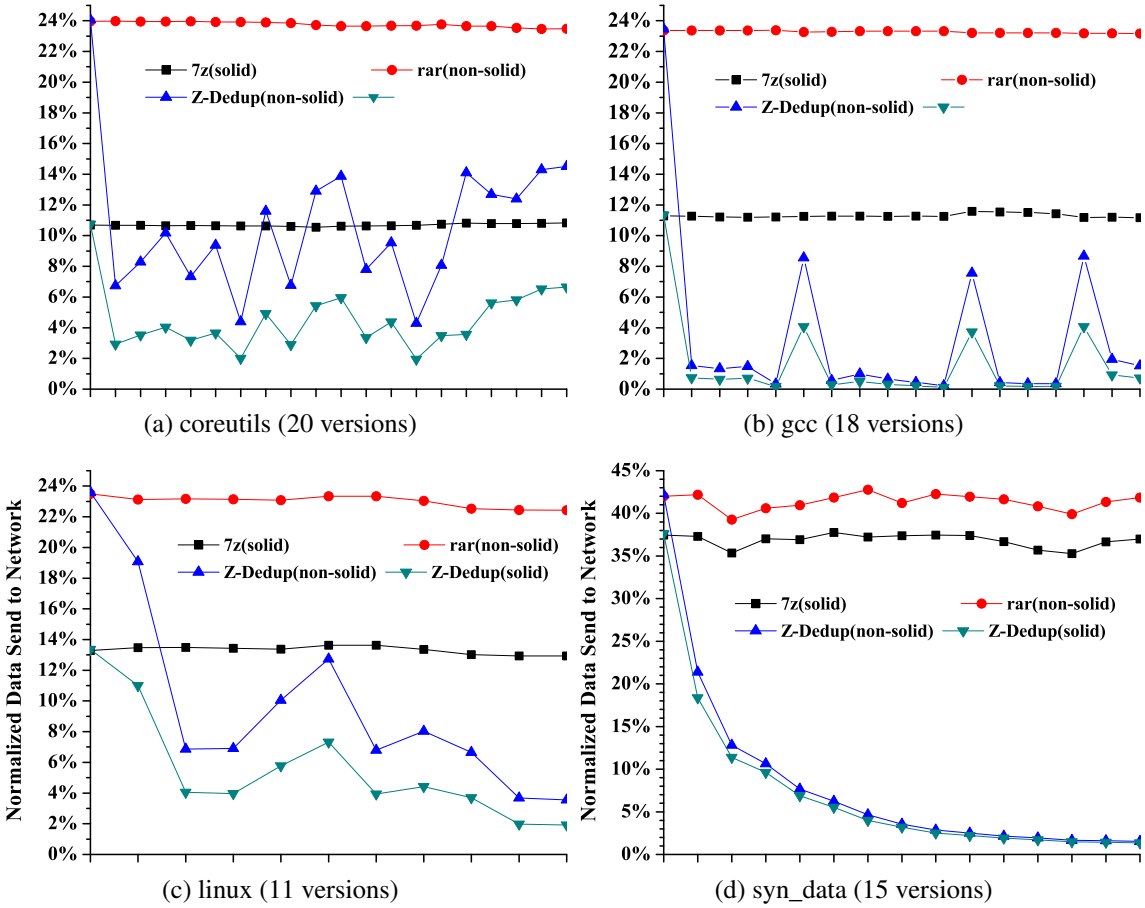

Figure 3.7: Data to be sent to the cloud server, while each number is normalized to the total size of its uncompressed package's size.

Z-dedup can further reduce these to 4.16% and 9.82% by exploiting the data redundancy embedded in these compressed packages. Moreover, the 7z and rar compressions will reduce the amount of gcc data to 11.30% and 23.28%, linux data to 13.32% and 23.01%, syn_data data to 36.83% and 41.37% in geometric average; while Z-dedup can further reduce these data to 0.62% and 1.28% in gcc, 4.69% and 8.30% in linux, and 4.16% and 4.70% in syn_data. Note, binaries files in syn_data show different redundant data curves than source code packages because these data usually have lower compression ratios than plain-text files. *From this evaluation, we further verify that Z-dedup can reduce most data transferring to the network*

Besides the reduced data network traffics, we have also evaluated Z-dedup's total processing times that include the local processing time and network transmission time. Z-

dedup can reduce the processing time by 10.71% to 31.52% without verifying file's ownership. It will reduce this time by 2.74% to 21.32% after adding the ownership verification.

3.5 Conclusion

With the exponentially expanding digital universe, compressed packages of files are poised to become an important form of storing and transmitting data in the cloud storage environment, for obvious economical and performance reasons. In this paper, we designed and implemented Z-dedup, a novel deduplication technique designed to detect and remove data redundancy in compressed packages for which conventional chunk-fingerprint based deduplication approaches have failed. The main idea of Z-dedup is to exploit some key invariant information embedded in the metadata of compressed packages, such as file-based checksum and original file lengths, as the unique signatures of individual compressed files. The design and implementation of Z-dedup also address a potential security vulnerability due to client-site compression, as well as packages generated by both non-solid and solid compression methods. Our evaluation of a Z-dedup prototype shows that it reduces up to 98.75% more redundant data in compressed packages than traditional deduplication system.

CHAPTER 4

WM-dedup:A Case for Watermark-based Chunk-level Lossy Image Deduplication

With the rise of mobile social media, various image editing and sharing applications will generate a large number of similar images, including a lot of redundant image contents. However, traditional data deduplication methods cannot detect such redundant image contents. As a result, we propose the WM-dedup approach, which leverages the steganographic watermark injecting and detecting to help perform the chunk-level image deduplication to improve the efficiency of the image storage systems.

4.1 Introduction

In social media applications, images are used to illustrate anecdotes or statuses that cannot be easily conveyed through words to express or emphasize the emotions or opinions. Therefore, it is easy to derive a lot of similar images from a single original image during a propagation of that image because users tend to modify the image to express their feelings. Meanwhile, an image can be compressed by several dozens of popular compression algorithms with different compression levels, thus resulting in a lot of compressed copies with same perceptual contents but totally different bitstream contents. In addition, the popular photo editing tools may also generate a lot of highly similar images, which share a lot of redundant perceptual contents. These application scenarios pose a challenge to the back-end storage of these social media or cloud storage service providers, which is how to efficiently organize and store such image data with very high semantic/perceptual content redundancy [30].

Similarity-based deduplication (SIM-dedup) approaches are proposed to filter out the near-duplicate images to save storage space [43] [19] [31]. They share a common workflow: 1) extracting features from images; 2) quantifying these features; 3) indexing features by a clustering algorithm; 4) querying an image's feature vector to detect the similarity. With a high similarity threshold, SIM-dedup approaches can select approximately duplicate images as image deduplication candidates that are then either deduplicated or retained based on the verification of their content equivalency. They work well on eliminating the perceptually identical images in their entirety, which amounts to the file-level dedupli-

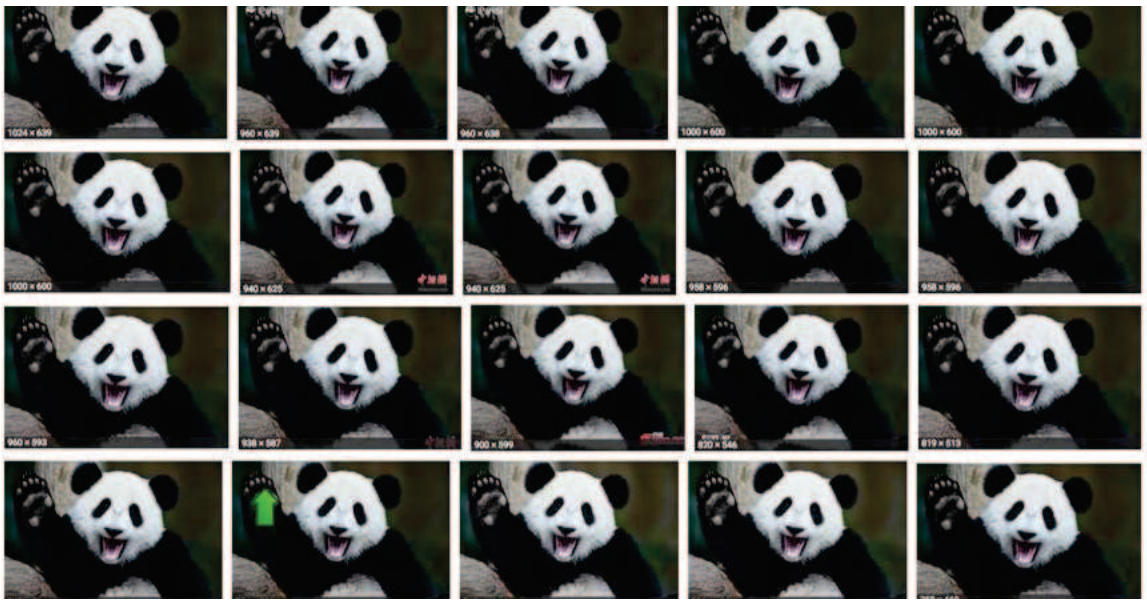

Figure 4.1: Examples of redundant image contents on the Internet.

cation, but cannot remove the partial perceptually identical contents. While the removal of perceptually identical images in their entirety amounts to the file-level deduplication, the partial removal of perceptually similar images is analogous to the chunk-level deduplication but at a significant computation cost on dividing the candidate images to perform sub-image deduplication. Moreover, for subsequent access operations, non-trivial costs are needed to store the sub-image's metadata. As a result, existing image deduplication approaches typically only adopt the file-level whole-image deduplication while ignoring the chunk-level sub-image deduplication.

With the plethora of ways in which images are produced, modified, shared, and stored, a large amount of redundant image contents will be introduced. As shown in Figure 4.1, which comes from the partial result snapshot of an image reverse search operation from Google, these images obviously share a large amount of human-observable redundant contents, which is the panda's photograph. However, there exist a lot of image instances with different aspect ratios and some of them have some slightly different logos embedded in the images by some users. The image storage system will incur more than $1000\times$ space to store all of these images without considering their redundant image contents. And it can reduce the storage overheads from more than $1000\times$ to around $100\times$ by removing the perceptually identical images in their entirety. In WM-dedup, we will further to detect and remove the sub-image content redundancy to improve the storage efficiency.

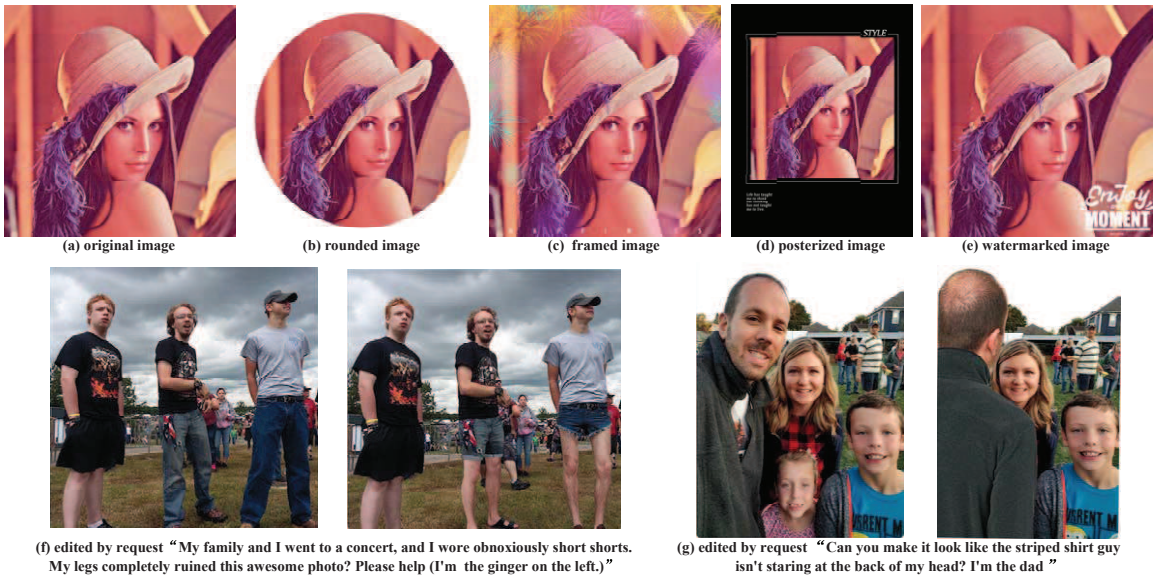

Figure 4.2: Examples of images with redundant perceptual contents, where fig.(b)-fig.(e) are results of applying simple picture editing operations upon fig.(a) and fig.(f)-fig.(g) show more sophisticated image editing operations.

In this paper, we propose WM-dedup, which leverages the embedded steganographic watermark [15] as a unique content tag to detect the redundant perceptual image contents. The main idea is that when a device or application generates a new image, certain descriptive information (e.g., user, device/application, date, location information) associated with this image will be injected into each chunk of an image generated by a predefined image chunking operation. This information will remain invariant in different variants derived from the original image without causing users to discern any perceptible differences, thus allowing WM-dedup to easily extract it to detect the redundant contents without excessive computation costs. We experimentally observe that WM-dedup reduces the image data size by 28.6% to 70.6% and increases the throughput by $4.5\times$ compared to SIM-dedup, by exploiting the chunk-level image redundancy. Moreover, as the first lossy chunk-level image deduplication system of its kind, all recovered images after WM-dedup do not have noticeable degradation in image quality.

4.2 Motivation

Currently, little research efforts have been spent on image deduplication because it is difficult to identify the equivalent perceptual image contents in a computationally efficient

manner. With respect to this work, the most relevant related work is SIM-dedup framework [43] previously mentioned in Section 4.1. SIM-dedup's foundation is based on the notion of reverse image search engine (RISE) [2] [1] or content based image indexing and retrieval (CBIR) [61] [19] [59].

4.2.1 Limitations of SIM-dedup

Usually, SIM-dedup framework comprises of two parts, one is to querying the image's similarity and another is to performing image deduplication. Before performing image deduplication operation, SIM-dedup validates the equivalent perceptual contents once detecting some potential redundant images. We implement a simplified SIM-dedup with image hash [35] to represent an image's perceptual contents. Image hash resizes the input image into a specific size before calculating the digital signals to obtain a hash value. To deduplicate redundant images, a threshold on a predefined similarity distance measure is used to allow for limited variances in perceptual contents. Table 4.1 shows hash values of fig.(a)-fig.(e) of Figure 4.2 generated by four image hash algorithms. The similarity among fig.(a)-fig.(e) cannot be detected by hash values, even though they share significant redundant perceptual contents. As a result, in our later evaluation, we implement SIM-dedup by extracting the high-dimension feature [51] and using the image hash to verify the content equivalent.

Based on our study, we also find that an image's aspect can significantly impact the accuracy detecting the equivalent image contents. As a result, image hash algorithms that only focus on a specific feature are only effective at detecting the exact or nearly duplicate image. They are not effective at detecting the redundant image contents of similar images. In other word, they work well on distinguishing the different image contents fail to effectively identify the redundant image contents. In this work, both SIM-dedup and WM-dedup employ image hash as an efficient way to verify image file/chunk content equivalent after they choose the potential redundant image file/chunk candidates. The reason of adopting image hash as the content equivalency checker are two-folded, first, its computation cost is low, second, distinct chunks will not generate the same image hash especially with the requirement of passing the first-round selecting of the potential redundant image file/chunk candidates without considering the intentional attacks from malicious users.

In order to defend against the intentional attacks, WM-dedup increases the difficulty of passing the first-round candidate selection procedure by encrypting the metadata information before injecting the metadata watermark. Moreover, we can adopt a set of image hashes in the second-round image content-equivalency verification stage to avoid the possible hash collision from distinct image files/chunks. Finally, existing image impaint

techniques can tolerate missing contents to restore the original image by the overall image style extracted from its chunks, which will be added to our enhancement model in future.

Table 4.1: Image Hash Values of fig.(a)-fig.(e) in Figure 4.2.

	aHash	pHash	dHash	wHash
fig.(a)	b69c3d890b0b8f8c	99c6562d7533a296	7670795b33135a38	be98bd890b0b8f8c
fig.(b)	f7838d01018183ef	bbc6f425c133c698	8e33795b33131b9e	ff898d89818183ef
fig.(c)	ff9d9d89890b0f0e	bbce94297133a296	6631795b33131a3c	ff999d89890b0e0e
fig.(d)	007e7e7e7e7e0000	91c46e342d6b936e	eeb4f89a96b8bc80	007e7e7f7f7e0000
fig.(e)	b61c3d09090b0f8f	99c6946d7533a296	7670795b33135a3c	b6989d890b0b8f8f

While other image features can be used in SIM-dedup implementation, they tend to be very computationally costly. Large-scale image retrieval systems use high-dimensional feature descriptors such as SIFT [41], SURF [5] for image analysis and retrieval. However, non-trivial SIFT variances exist in Figure 4.3 between the original and edited images even though they share a lot of redundant contents. It requires a significant amount of computation to detect the SIFT, SURF or other feature descriptors and run random sample consensus (RANSAC) algorithms to form a consensus feature set to check the redundant perceptual image contents [22].

Existing SIM-dedup approaches share the following drawbacks: 1) difficult to find panacea features at any first attempt; 2) re-extracting new features from all existing images and rebuilding the index when a new feature is added; 3) some simple image editing operations can lead to dramatic changes in the feature vector of an image although most parts remain intact; 4) lacking a practical method to perform subimage-level deduplication; and the sheer number of images, constantly on the rise, will exacerbate these problems.

These drawbacks can lead to at least three potential problems. First, they cannot identify complicated redundant images because it only focuses on the local image features while lacking sufficient global information despite of steps 2 and 3 that can blend in some global descriptors. Second, excessive computational overheads make them unsuitable for image dedup. Third, they only target near-duplicate images, images with most features being identical, while ignoring the non-trivial redundancy.

4.2.2 Case Study of Redundant Image Data

An image is typically generated by an electronic image sensor to record light as a set of electronic data for each pixel. The data is then electronically processed and stored in a

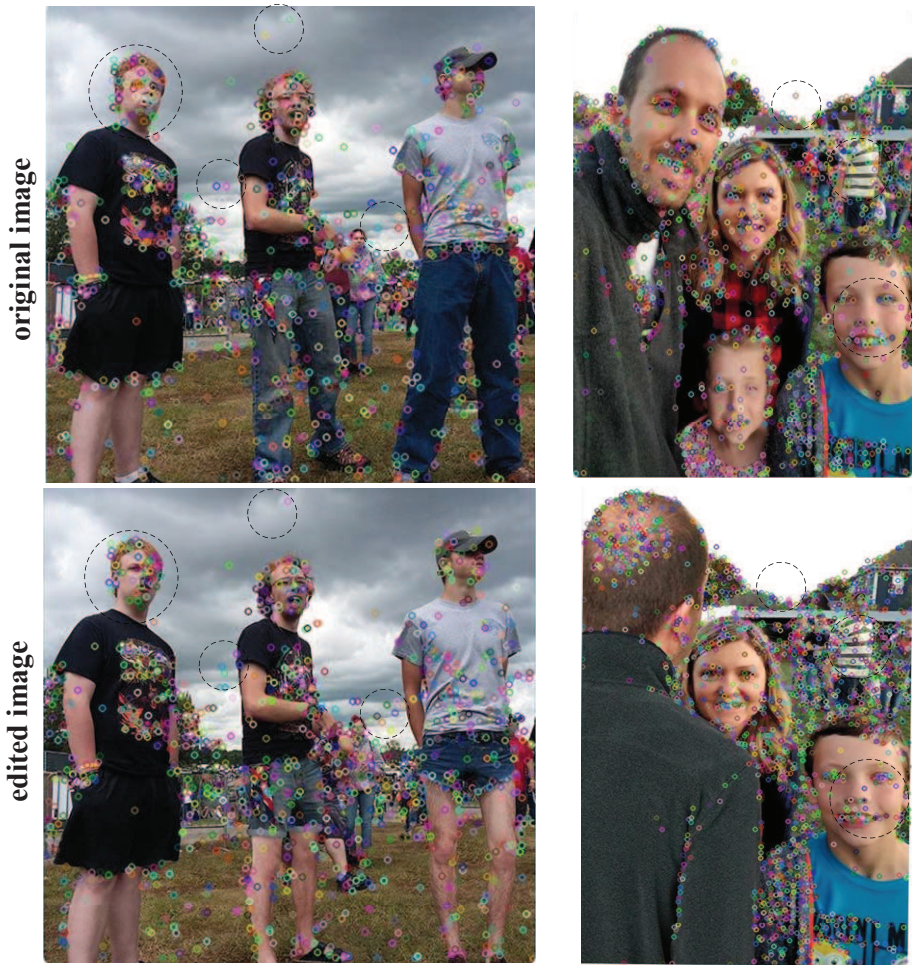

Figure 4.3: SIFT features, where the dotted circles contain the SIFT variances between the original and edited images.

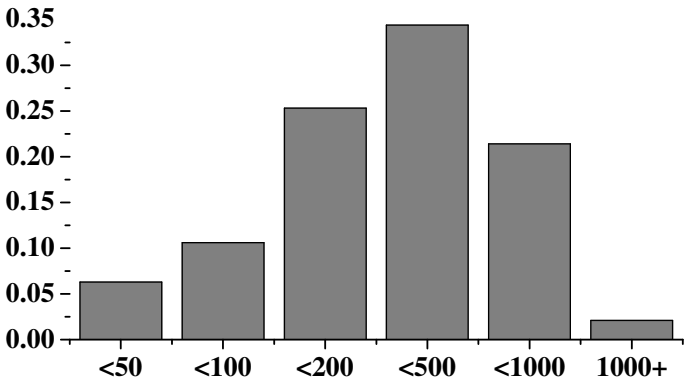

Figure 4.4: Distribution of all copies.

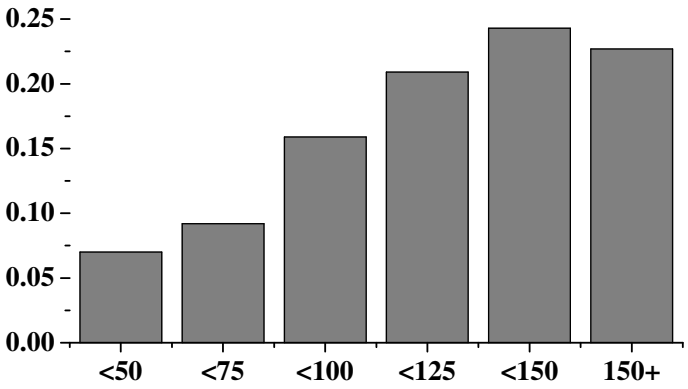

Figure 4.5: Distribution of similar copies.

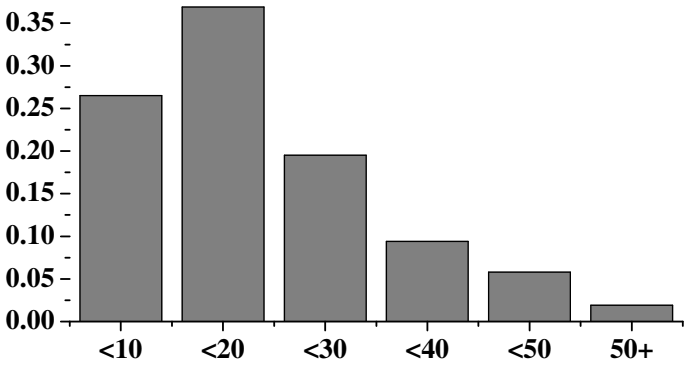

Figure 4.6: Distribution of distinguishable copies.

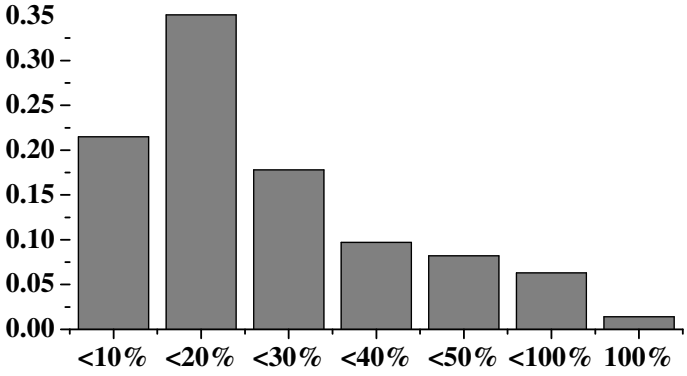

Figure 4.7: Distribution of fraction of modified content.

digital image file for subsequent displaying or processing, making it a highly manipulative medium. As a result, users can easily edit the same digital image to meet their personal requirements and tastes, leading to a large number of perceptually redundant contents among these derived image files, especially in the era of social media.

Near-duplicate image detection has been studied for many years [23]. These studies found that vast numbers of images on the web are duplicates or near-duplicates derived from the same original images. Specifically, a near-duplicate statistics study on 70K images randomly selected from 2 billion images has shown that 22% web images have near duplicates [58], where images with similar/identical semantic contents are considered near-duplicate images. However, existing research does not show how much redundant perceptual contents exist in these images, let alone eliminating such redundant image contents among these similar images.

In this work, we have randomly selected 585 original images from the Internet and used Google's image reverse search engine to obtain all searchable copies with similar contents, resulting in about 275K images that we refer to as "all copies". We analyzed these images to learn various distributions of redundant image contents. As shown in Figure 4.4, there exist a large number of fully identical images, about 200K out of the 275K images, of which most have more than 200 copies and some have more than 1000 copies. After filtering out the exactly identical images, we obtained the distribution of similar but non-identical copies, which resulted in about 75K images that we refer to as "similar copies", as shown in Figure 4.5, where most images have more than 100 copies, which are mainly caused by compressing or cropping an image into a series of different size images(e.g., cropping a 1920×1080 image to 1919×1079). Moreover, we filter out these similar copies by a similarity threshold of 0.95 to obtain the distribution of similar but distinguishable copies, which resulted in about 15K images that we refer to as "distinguishable copies", with different perceptual contents shown in Figure 4.6. We find out that most images have at least 10 distinguishable copies. Finally, Figure 4.7 shows the distribution of the fraction of the modified image's perceptual contents in those distinguishable copies, and we have observed that most of these copies have only less than 30% their perceptual contents modified, which means that a lot of perceptually equivalent image contents exist among these distinguishable images.

This case study strongly suggests that (1) significant perceptual content redundancy exists in image data generated in the Internet and (2) a substantial amount of this redundancy cannot be detected and removed by traditional image deduplication approaches such as SIM-dedup.

4.2.3 Lossy Image Deduplication and Recovery

Unlike computers, which can evolve in a short time to acquire more compute powers, there are some obvious limitations to human's capability to perceive information contained in images. These limitations stem from the physical structures of human eyes and brains to process visual signals, which evolve very slowly. Rapid technology advances have helped produce large volumes of images that are beyond human's perception limits, which provide an opportunity that machine can help process the images with human's indistinguishable changes to assist image deduplication. Lossy compression algorithms have already exploited this fact to reduce the image size significantly before image quality degradation is noticed by the end-users. Moreover, recent advances in machine learning have led to a series of techniques that can be employed to enhance the image perceptual contents. Specifically, there exist some techniques that can produce high-quality versions of low-resolution images, eliminate the undesired noisy signals and inpaint the whole image with some partial contents [49] [55] [32]. As a result, we believe that lossy image deduplication would be acceptable if a deduplicated image can be recovered by a combination of these techniques to be perceptually equivalent to its original image.

4.3 WM-dedup Design and Implementation

Existing SIM-dedup approaches perform image dedup at the file-level by identifying near duplicate images. However, their restore process only returns the near duplicate images, instead of truly restoring the eliminated images preferred by the end users. WM-dedup, on the other hand, contains both processes to deduplicate and recover images. SW-dedup is designed to restore deduplicated images to their original form or sufficiently close in that there will be no end-user discernable degradation in perceptual content quality. In particular, WM-dedup has three main functional components, watermark generation, deduplication and recovery, shown in Figure 4.8, to support its various operations.

4.3.1 Image Watermark Generation

In the image generation stage, WM-dedup needs to inject a unique steganographic watermark for each image. This function can be added to a digital camera's firmware or embedded in a specific application when it is to generate a new image file. In general, there are two kinds of watermarks, fragile and robust. A fragile watermark is usually used to check data integrity while a robust watermark is commonly used to identify ownership or copyright [11]. In the case of WM-dedup, we employ the *robust* watermark. Our steganographic watermark is a bitmap image for the metadata descriptive information, e.g.,

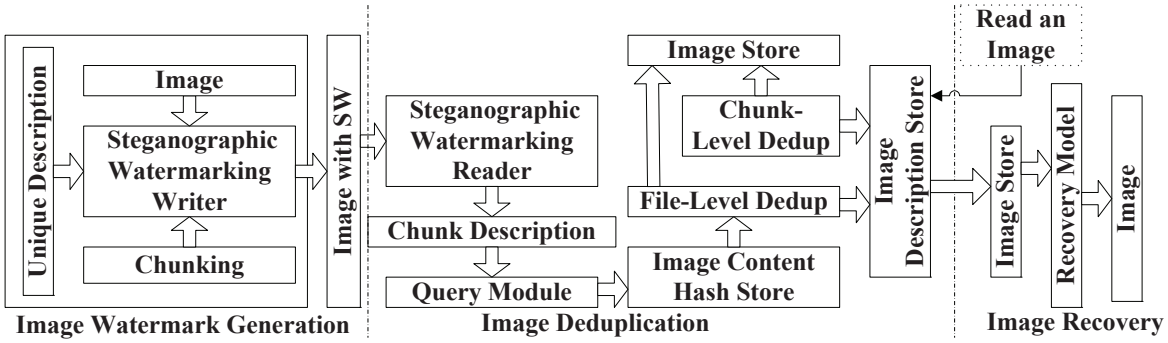

Figure 4.8: Architecture of WM-dedup.

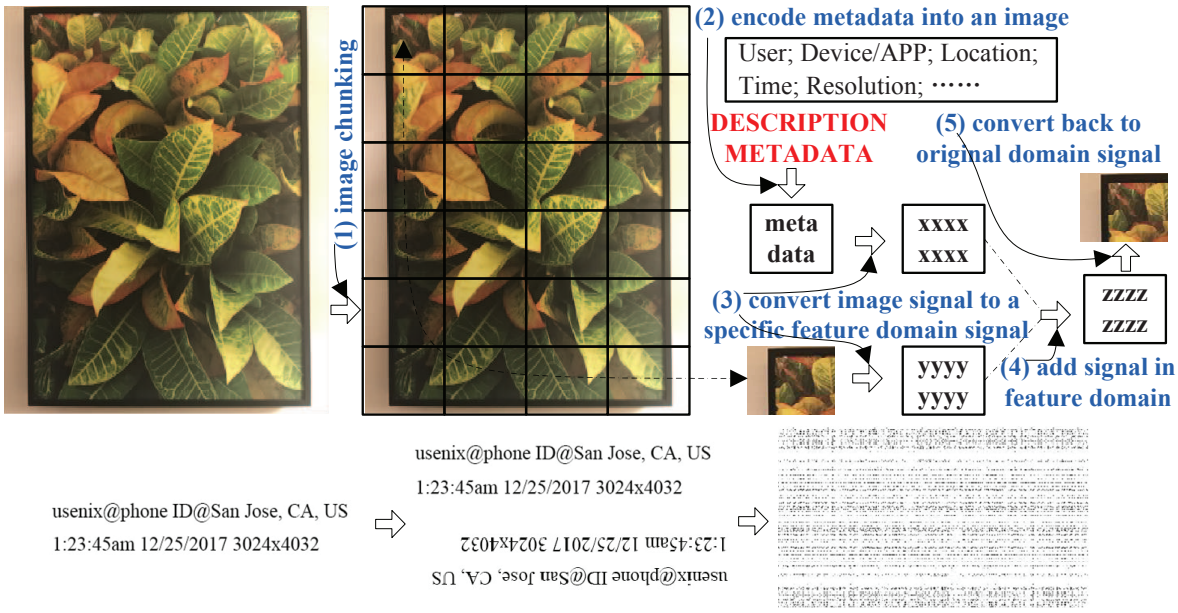

Figure 4.9: Watermarking an image.

user, device/application, date, location, resolution information, associated with this image. As a result, this steganographic watermark information can uniquely represent the image's perceptual contents.

Besides injecting the metadata as steganographic watermark to enable image deduplication, the steganographic watermarking writer module also applies a predefined image chunking algorithm to divide an image into multiple fixed-size subimages, hereforth referred to as "chunks". A simple image chunking algorithm is to divide an image into a series of fixed-size chunks with each chunk being capable of embedding the steganographic watermark.

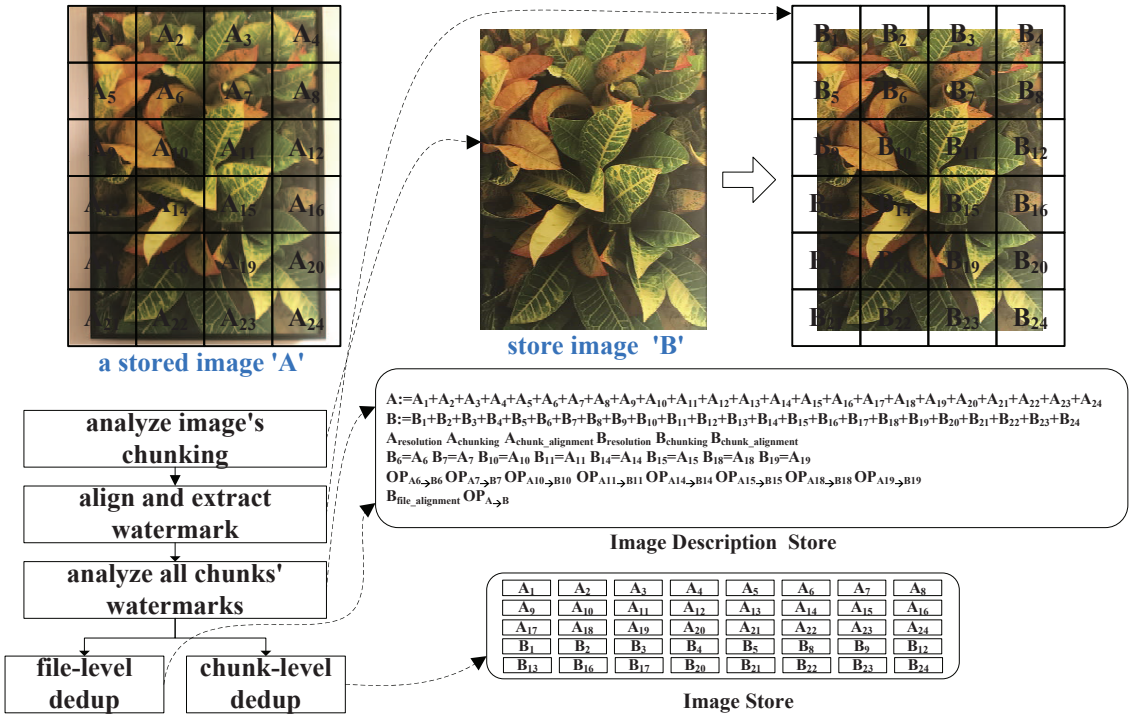

Figure 4.10: Workflow of image dedup

Figure 4.9 shows the process of injecting an invisible watermark of "usenix@phone ID@San Jose, CA, US 1:23:45am 12/25/2017 3024x4032". WM-dedup first encodes the metadata into a mirroring bitmap image, then converts both metadata image and image chunk into the Fourier transform domain. Finally, it adds their signals in the Fourier transform domain and converts the result back to the original time domain.

4.3.2 Image Deduplication

The steganographic watermarking reader aligns image chunks by the embedded chunking information. It extracts the watermark embedded in each chunk before recognizing the unique metadata and uses image hash to verify content equivalency of those potentially redundant chunks with the same metadata information. In Figure 4.2, (f) and (g) illustrate a situation that an image been stitched from several different images and each of these images may have its own image description information. To deal with this case, WM-dedup can query the chunk description for the possible redundant chunk to perform chunk-level deduplication directly. Moreover, when storing a unique chunk to image store, WM-dedup adds 10% redundant contents around its boundaries to help reassemble the corresponding image from chunks.

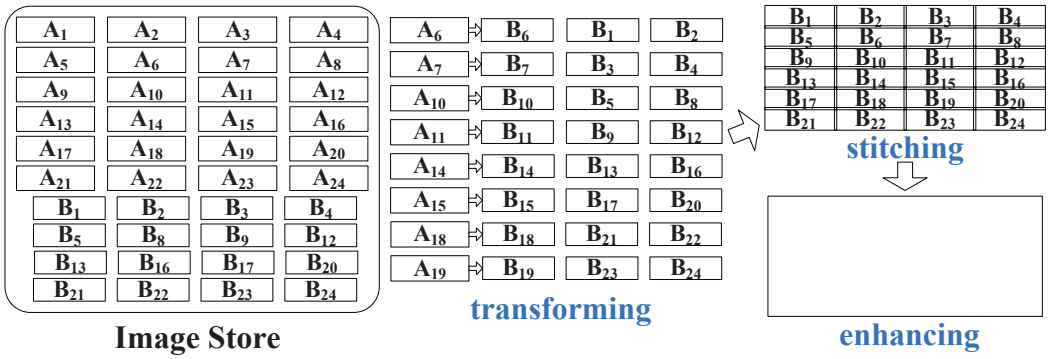

Figure 4.11: Workflow of image recovery.

In Figure 4.10, space can be saved when storing image ‘B’ because of the 8 detected redundant chunks, which is about half of the perceptual contents of image ‘B’. In order to detect and eliminate more redundant image contents, an image can perform file-level deduplication first if all its chunks’ descriptions are homogeneous. For example, image ‘B’ is recognized as a cropped subpart of image ‘A’. The file-level image deduplication can remove image ‘B’ if there is a record of how the cropping operation was done on image ‘A’ to produce image ‘B’. Image description store keeps all file-level and chunk-level description information in the WM-dedup system, plus the corresponding operations on how to convert existing contents to the eliminated contents. WM-dedup has defined some simple but common operations to help dedup and recover redundant images, such as cropping, resizing, rotating, applying a mask, adding a text, changing color channel. With these operations stored in the image description store, WM-dedup can quickly apply the corresponding operations to recover the eliminated image file.

Once a query module to the image store returns “true”, the incoming chunk may have a redundant copy existing in the image store. WM-dedup next verifies the perceptual content equivalence by comparing their image hashes before replacing the incoming chunk with the redundant copy and write the corresponding operations to convert the incoming chunk from the redundant chunk. WM-dedup adopts a high-resolution replacement policy, which prefers to replace low-resolution chunk with high-resolution chunk. This is because some low-resolution images may enter in WM-dedup before the high-resolution copies, due to the asynchronous nature of cloud storage.

4.3.3 Image Recovery

In WM-dedup, image recovery refers to the process of restoring a perceptually e-quivalent image to the one that was replaced by the redundant copy in the image dedup

stage. Based on the information stored in the image store and image description store, it deploys the corresponding operations to restore the image. As shown in Figure 4.11, image recovery entails three subtasks: transforming, stitching, and enhancing. The transforming task applies the corresponding operation to restore the chunks. If an image was deduplicated at file-level, it can bypass this task that tries to combine all the corresponding chunks to form an image. With the extra 10% redundant contents around each chunk's boundaries, WM-dedup can quickly stitch all chunks to generate a candidate image other than spending a long computation time to align and combine these chunks. Finally, the enhancing task further optimizes the image by using a super-resolution recovery model to enhance the quality of the candidate image, especially for the chunk's boundaries.

4.3.4 Put It All Together

We have introduced the main functional modules of WM-dedup. Here are some specific implementation parameters of WM-dedup used in this paper: 1) WM-dedup divides each image into a series of "256×256" sized chunks. 2) WM-dedup can inject up to 128 characters as image descriptive information embedded into its steganographic watermark. 3) WM-dedup overlays original and watermark image signals in the Fourier transform domain. 4) WM-dedup first filters in the image contents by the image description information recovered from the watermark, then compare their image hashes to verify the perceptual content equivalence on dedup. 5) Each chunk adds 10% redundant contents around it to help with the stitching operation. 6) WM-dedup adopts a super-resolution model to enhance the perceptual image contents.

Finally, in order to protect against counterfeit attack, we encoded the metadata into a specific Fourier transform domain and then combine it with the input image at this feature domain. As a result, directly slicing, modifying, or cutting the image does not impact the signal pattern in the specific feature domain. This suggests that the watermark process can resist such forgery attempts. However, it also indicates a potential attack to watermark, which is to forge the watermark information to counterfeit the images. WM-dedup uses encryption to provide the protection against this kind attack. It encrypts the description information before generating its bitmap image, and WM-dedup decrypts the description information after recognizing the characters of description information.

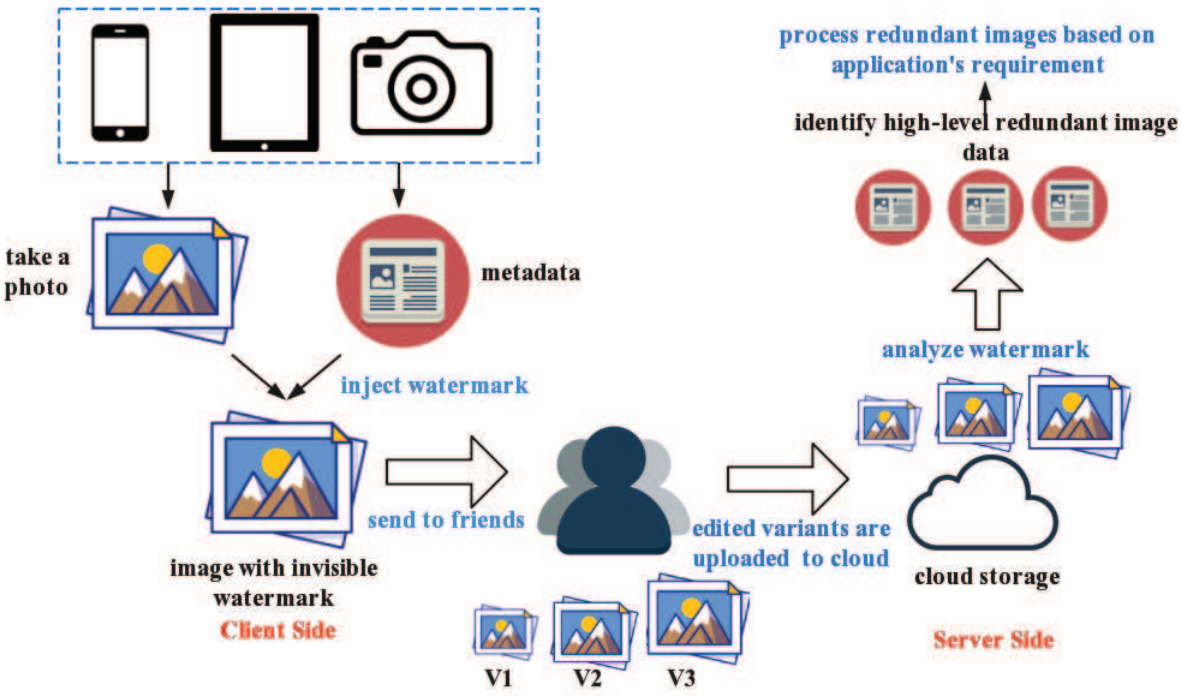

Figure 4.12: Evaluation system overview.

Table 4.2: Hardware Configuration.

CPU	Intel i7-6700K at 4GHz
GPU	Nivida GTX 980 Ti 6G
Memory	64GB DDR4 2800
SSD	1TB Samsung 850 EVO
HDD	6TB WD Black 7200 RPM
NIC	1000BASE-T Intel I219-V

4.4 Evaluation

In this section, we first describe the experimental environment used to evaluate WM-dedup. Next, we describe the image data set used for the evaluation. Finally we present and analyze our evaluation results.

4.4.1 Evaluation Setup

Figure 4.12 shows the integration of WM-dedup with a cloud based image storage application, where a user can take a picture and distribute it. The picture can also be edited prior to being uploaded to the cloud, which is a typical use case of cloud based multi-tenant image storage services. We use two directly attached workstations to simulate this

cloud image storage environment and their detailed hardware configurations are listed in Table 4.2. A standalone software module is running to inject the steganographic watermark. SIM-dedup can only perform file-level deduplication on near-duplicate images while WM-dedup can perform both file-level and chunk-level image deduplication.

We have implemented a SW-dedup prototype, which combines a machine learning model and image hash to construct the two-phase image content redundancy detection describe in Section 4.3. We have also built a SIM-dedup prototype, which allows 5% perceptual variances at most by setting the similarity threshold to 95% to perform the near-duplicate deduplication introduced in Section 4.2.1. We compare WM-dedup against SIM-dedup to evaluate its potential benefits by deduplicating similar but distinguishable images derived from all the tested datasets.

4.4.2 Data Set

As introduced in Section 4.2.2, we chose 585 different images from the Internet and used Google's image reverse search engine to get all searchable copies of each of these images. The search results, amounting to more than $275K$ image copies, are analyzed. Within these copies, there are over $75K$ similar but non-identical (unique) copies, which we use to profile image's redundant perceptual contents in the public network.

Meanwhile, we have also observed that there exist a large number of images that are stored in private or social networks photo storage applications such as iCloud, Dropbox, OneDrive, Google Photos, Facebook, and Amazon Prime Photos, which are not allowed to accessing their repositories to profile image's redundant perceptual contents in these private networks. As a result, we tried to get this information using an alternative approach as follows.

Next, we conducted an investigation to identify commonly used image editing operations. Initially, we planned to investigate user's behaviors but this would require accessing some private image repositories, which is unfortunately but understandably very difficult, if not impossible.

We analyzed top-rated photo editing applications in Apple Store and Google Play to learn the most popular photo-editing operations provided by these applications such as cropping, resizing, stitching, adding a text, adding a frame, adding a sticker, applying a filter, etc. Besides the image data set directly collected from the Internet, we collect some private images and run a script that randomly changes each image's perceptual contents to some extent (from 0 to 50%) to generate a synthetic image dataset by applying some of the top-rated photo editing operations to emulate the edited image data existing in private and social-networks photo storage repositories. Moreover, we also created another synthetic

Table 4.3: Data Set Summary.

	# of Ori Img	Ori Img Sz	# of Dist Img	Dist Img Sz	Per Redu Rt
Int_Set	585	1.3 GB	15543	22.8 GB	74.6%
Syn_Set1	220	750 MB	2640	8.2 GB	78.2%
Syn_Set2	102	324 MB	255	735 MB	62.3%

image data set by collecting the images on different "photoshop request" on the Internet. These two synthetic datasets are used to emulate both simple and complex edited image contents, whose redundant contents can only be detected and removed by the chunk-level deduplication function in WM-Dedup.

Because the watermark information is non-existing in these image copies, we resorted to manually comparing them, labeling the perceptually equivalent areas and then adding steganographic watermarks to these images. These two synthetic datasets are used to emulate the fact that most of the perceptual differences among similar images in the Internet datasets mainly come from adding some texts or stickers.

Table 4.3 shows a summary about these datasets with the numbers and total sizes of the original images and similar but distinguishable images, and the estimated perceptual content redundancy ratios, where "Int_Set" directly comes from the Internet data, "Syn_Set1" and "Syn_Set2" represent the simple and complex edited image data. We approximately estimate the redundancy ratios by a program that aligns each image with its original image, divides them into multiple 64×64 chunks ($\frac{1}{16}$ of our fixed-size chunk) and counts their equivalent perceptual chunks to obtain the redundancy ratios.

4.4.3 Evaluation Results

4.4.3.1 Data Reduction

As shown in Figure 4.13, "Int_Set" has 585 images with a total size of $1.3GB$, while all copies existing on the Internet amounts to $362.5GB$. The similar copies need $114.2GB$ storage space to store and the final distinguishable copies filtered out by SIM-dedup amount to $22.8GB$. Note that "all copies", "similar copies" and "distinguishable copies" are defined in Section 4.2.2. WM-dedup, on the other hand, can further reduce the data size to $6.7GB$, thus significantly reducing the storage space by 70.6%. We define *redundant image amplification factor* as the actual image data size divided by the original image data size because all of these image contents are derived from the original images. We calculate the redundant image amplification factors and find that WM-dedup has decreased it from 17.5 to 5.2 as shown in Figure 4.13.

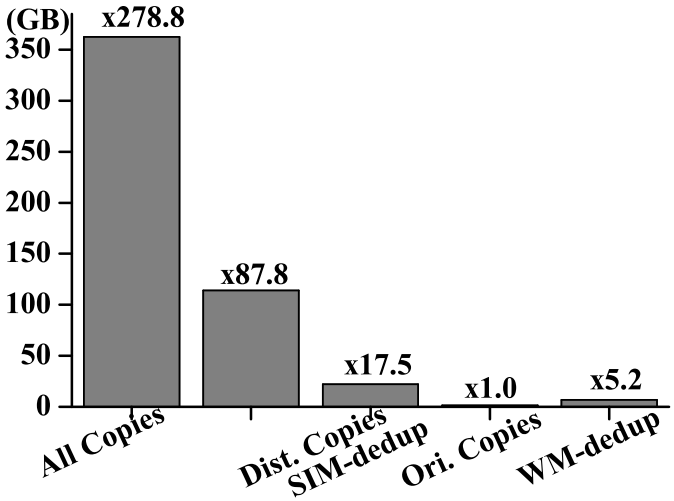

Figure 4.13: Storage sizes on Int_Set.

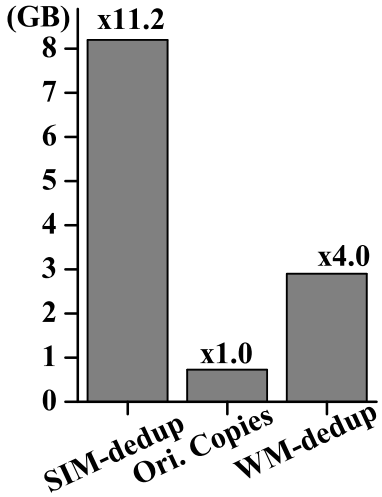

Figure 4.14: Storage sizes on Syn_Set1.

Figure 4.14 and Figure 4.15 show the storage requirements of “Syn_Set1" and “Syn_Set2". Their storage sizes can be reduced to $2.9GB$ and $525MB$ from $8.2GB$ and $735MB$, respectively after applying WM-dedup. This translates to 64.6% and 28.6% storage space reductions coming from the chunk-level image deduplication enabled by WM-dedup, which SIM-dedup is incapable of due to its file-level similarity detection. These data reduction ratios should increase as more redundant image copies continue to accumulate in image storage systems, on the Internet, the cloud, etc., thus making WM-dedup more appealing in the new social-media era.

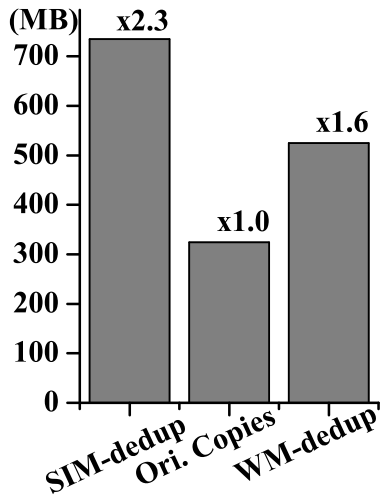

Figure 4.15: Storage sizes on Syn_Set2.

4.4.3.2 Execution Time

Table 4.4 lists the execution time breakdown by the main operations such as feature extraction, image hash, query and restore on both SIM-dedup and WM-dedup running in the single-threaded mode. SIM-dedup spends most of its time on extracting high-dimension feature vector based on the VGG16 network [51] with ImageNet pre-trained weights to automatically extract image's hierarchical features and build the image contents search engine. Specifically, it generates a 4096-dimension fc6-feature [37] for each image. Image hash can help quickly verify the image content equivalency. Its query operation is fast and the restore operation is mainly on the latency of selecting and loading the near-duplicate image. On the other hand, WM-dedup spends much less time on extracting the watermark, reading the bitmap image's content to recognize image description metadata. The boundary box information of watermarks provides hints to WM-dedup to align the incoming image to its potentially redundant copy existing in the image storage. Most of its time is spent on recovering the image by reassembling it with the perceptually equivalent chunks and applying super-resolution optimization on all boundaries between any two adjacent chunks to restore the deduplicated image. As a result, WM-dedup's deduplication throughput is more than $4.5\times$ higher than SIM-dedup's although it has provided more complex chunk-level deduplication operations, thus capable of processing about 10 images/second running in a single thread.

Our three image datasets have limited numbers of images and their content index can fit within the main memory, reducing the overhead from accessing data off main memory. The index query sensitivity study will be our future work, and we can adopt approaches by similar works on traditional deduplication systems to overcome this bottleneck [38] [62].

Table 4.4: Execution Time Breakdown.

	Ft. Extra.	Image Hash	Query	Dedup	Restore
SIM_dedup	482ms	4ms	25us/file	502ms	8ms
WM_dedup	84ms	4ms	12us/chunk	112ms	195ms

Table 4.5: Mean PSNR (dB) Comparison.

	Original	Injected	Restored (without SR)	Restored(with SR)
Int_Set	45.2	43.6	38.3	40.5
Syn_Set1	40.4	37.1	32.2	34.1
Syn_Set2	36.7	33.9	28.7	31.2

4.4.3.3 Memory Overhead

In terms of storage overhead, WM-dedup incurs no storage overhead on the watermarked image because it overlays signal at a specific feature domain. Because our feature domain is based on Fourier transform, the memory overhead incurred by our watermark is proportional to the resolution of the input image when loading its content into the memory. In the JPEG format, each pixel requires 24 bits to store its RGB information. A 12-megapixel JPEG image would require roughly 36 MB of memory to store its contents. This digital image processing can be treated as the various operations applying on a large size matrix containing the image contents. As a result, the total memory size of the input matrix, output matrix and watermark matrix is less than 100 MB when we process a 12-megapixel JPEG image. Comparing with the abundant memory resources in today's computer systems, this level of memory overhead is arguably acceptable. Given that more and more applications are poised to access cameras in mobile devices to take photos, it may be desirable to combine this watermarking process with the original JPEG compression to further reduce its computation and memory overheads. Besides the memory overhead on processing each image, there is an overhead of 128 bytes per image as its description plus the necessary pointers to maintain the index structure of image contents, which is similar to the index structure in existing deduplication systems whose overheads will be evaluated as our future work.

4.4.3.4 Image Quality Loss

WM-dedup is the first chunk-level lossy image deduplication system that replaces a chunk by its perceptually equivalent counterpart. In order to verify the lossy dedupli-

cation/recovery functionality, we conduct our evaluation using images with common resolutions ranging from 0.6 megapixels to 12 megapixels for social media uploads, check the contents of each lossy restored image, and do not notice any perceptual degradation between it and the original input image.

We choose the peak signal-to-noise ratio (PSNR) as a metric to evaluate the image quality degradation. PSNR is defined via the mean squared error (MSE). Given a noise-free m×n monochrome image "I" and its noisy approximation "K", MSE is defined as $\frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$; and the PSNR (in dB) is defined as $20 \times log_{10}(MAX_I) - 10 \times log_{10}(MSE)$, where MAX_I is the maximum possible pixel value of the image.

PSNR is the most widely used objective measure for image quality assessment, but the existing work also indicates that the evaluation results of PSNR cannot be exactly the same as the visual quality seen by the human eye. It is possible that an image with a higher PSNR may appear to be worse than an image with a lower PSNR. This is because the human eye's visual sensitivity to errors is not absolute, the perceived results of many changing factors. For example, the sensitivity of the human eye to spatial frequencies is relatively low. The sensitivity of the human eye to the contrast in brightness is higher than the chroma. The perception of an area by the human eye is affected by the surrounding areas.

As shown in Table 4.5, we have listed the mean PSNR values of original images, original images with injected watermark and restored images without or with the super-resolution optimization of our three different data sets. Injecting steganographic watermark decreases the PSNR to some extent and the recovery process usually further decreases this metric. However, we find that PSNR has nearly the same value of injected copy when WM-dedup restores the deduplicated images. We believed that it may be due to the super-resolution optimization that can generate the best interpolation values to reduce the MSE. We plan to add more restore optimization models as our future work to further control the image quality loss.

4.4.3.5 Study on Predefined Chunking Method

In general, injecting watermark requires a specific size of the receiving chunk to hold the bitmap of metadata information. If a chunk is smaller than the required minimum size for injecting watermark, some metadata information could be lost when WM-dedup tries to extract the watermark. In WM-dedup, we define each chunk as a square and use it to divide an input image into multiple full chunks and some truncated chunks (e.g., on the edges of the image), where the latter will be expand to a full chunk with blank filling.

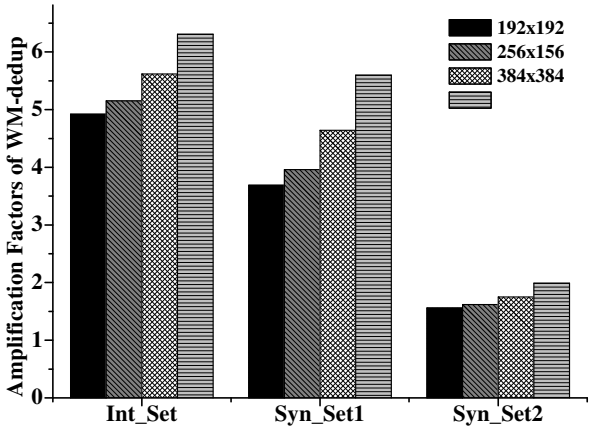

Figure 4.16: Amplification factor as a function of chunking size.

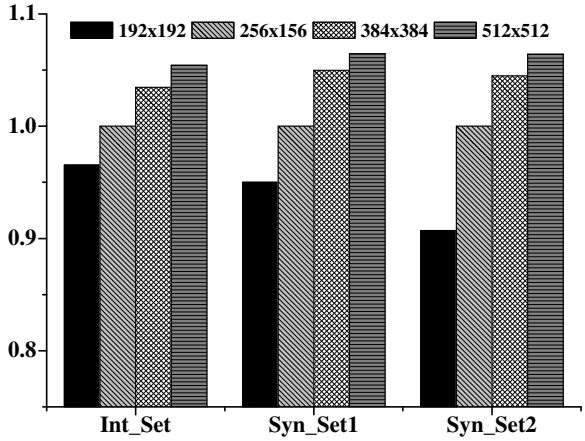

Figure 4.17: PSNR value as a function of chunking size.

Specifically, we use a 12×12 bitmap to hold a printable character, and provide a maximum 128 characters to describe the metadata information. With the mirroring operation, the minimum size of a chunk will need at least 36 kilopixels ("$12 \times 12 \times 128 \times 2$") to hold this bitmap image. As a result, WM-dedup requires that an image chunk must have enough space to hold at least one chunk watermark (36 kilopixels) at 192×192. Fortunately, it should not impact the target photo storage application because the sizes of high quality photos tend to be much larger than this required minimum size.

WM-dedup chooses the chunking size of "256×256" to divide the images because it can achieve a balance between high data reduction ratio and image quality loss. As shown in Figure 4.16 and Figure 4.17, we have listed the final data amplification factors and the PNSR values changing under 4 different chunking sizes, "192×192", "256×256", "384×384" and "512×512". Currently, we have only implemented fixed size image chunking in WM-dedup, and we plan to explore the variable size image chunking as our future work.

4.5 Conclusion

In order to exploit the perceptually redundant contents existing among similar images, we propose WM-dedup, the first lossy chunk-level image deduplication system, which embeds an invariable image chunking and content tag by injecting steganographic watermarks to an image when it is generated. Based on our evaluation, WM-dedup can further save the image data storage space by 28.6% to 70.6%, compared to the existing file-level image deduplication approach SIM-dedup, by exploiting the chunk-level redundancy without noticeable degradation of image quality in recovered images. Moreover, it also outperforms SIM-dedup in deduplication throughput by about $4.5\times$, owing to its novel content representation approach.

CHAPTER 5

SES-dedup: a Case for ECC-based SSD Deduplication

With the decline in unit price and significant advantages in performance, solid state disks (SSDs) are gradually replacing traditional hard disk drives (HDDs) as the main storage medium for storage systems. So far, their unit price is still around one order of magnitude higher than unit price of HDD, and they may suffer from the inherent disadvantages of a limited number of programming and erasing (P/E) operation, which degrades their reliability and performance. Therefore, integrating deduplication within the SSD becomes an attractive solution. However, SSD's internal architecture cannot directly employ the low-cost deduplication engine, we propose the SES-dedup approach, which aims to overcome this problem to bypass the widely used data randomization module in SSD to enable the low-cost ECC-based deduplication.

5.1 Introduction

How to manage the explosive growth of data is a top-priority problem in the big data era. There exists a significant amount of redundant data in the exploding digital universe [26], which makes data deduplication (dedup), a space and compute efficient solution for storage capacity optimization, become a standard feature for a lot of storage products and installations such as backup systems, file systems, all-flash arrays, cloud storage systems and so on.

Several studies have proposed integrating data deduplication into SSDs to leverage their internal processing capability to achieve such single-instance storage within the SSD. These deduplicatable SSDs can not only exploit the benefits of reducing the Program/Erase (P/E) operations to increase SSD's lifespan but also proportionally enlarge its logical capacity to improve the performance of its behind-the-scenes tasks such as wear-leveling (WL) and garbage-collection (GC) [36] [16]. Moreover, deduplication will further improve the SSD's reliability because the raw bit error rate will increase sharply with the number of P/E operations [9] [27]. However, deduplication will incur notable computation and space overheads [28]. As a result, designers must balance the potential overheads and benefits of deduplication to make the right design choice.

Different from traditional data deduplication systems running on general-purpose computer systems or servers, deduplication within SSD is usually constrained by the very limited resources within the SSDs (such as embedded processor and DRAM). In traditional deduplication systems, data streams will be dynamically divided into either variable-length or fixed-size chunks, where a cryptographic hash (such as SHA-256) is calculated per chunk as the fingerprint to uniquely represent the chunk's content, and chunks detected to share the same fingerprint with an already stored unique chunk are considered duplicates and eliminated by replacing them with a pointer each to the unique chunk. Unfortunately, dynamically chunking and computing cryptographic hashes will incur a great deal of computational overhead to SSD's embedded controller, which will affect its frontend I/O performance, thereby offsetting deduplication's advantages. For example, even for some mid-range all-flash array products with the server-level processors, their performance is CPU-bound, which is mainly caused by the data reduction tasks of deduplication and compression [3] [34]. Therefore, we believe that may be able to significantly improve the performance of these systems by removing substantial overhead calculating the fingerprints in their deduplication modules.

Recent studies have proposed some low-overhead approaches such as fixed-size chunking (at page size) and using a page's ECC instead of SHA-256 as the fingerprints to help detect the duplicate chunks, which leverages the SSD's internal page mapping mechanism and the page-level ECC without incurring extra computation cost and at a small space cost other than necessary space to maintain the hash table for data deduplication [36] [16]. Unfortunately, on SLC, MLC, TLC or QLC flash chips, different data patterns written have been proven to exhibit different raw bit error rates because of various electronic interference effects [12]. To reduce the raw bit error rates induced by these similar data pattern's disturbances, modern SSDs have integrated a data scrambler module to randomize the incoming data before storing it to the NAND flash chips. As a consequence, ECCs for duplicate data blocks with different LBAs will be rendered completely different, which makes it impossible to leverage this built-in ECC function as a content identification function to detect the duplicate data on flash chips.

In order to solve this problem, we propose a Scrambler-resistant ECC-based SSD deduplication, called SES-dedup, which can be implemented in either the host or device side to break through the data scrambler module so as to enable ECC-based deduplication on modern SSDs. Through extensive evaluations on an simulated SES-dedup system, SES-dedup is shown to effectively exploit SSD's built-in ECC module to calculate the hash values of stored data for data deduplication. Specifically, our SES-dedup approach can remove up to 30.8% redundant data with up to 17.0% performance improvement by feeding our collected data traces to the SSD simulator. Through our experiments, we find that when

Table 5.1: Page-level operation latencies on different flash chips

NAND Type	Read	Write	SHA-256
SLC	23.4 us	262.6 us	226.5 us
MLC-1	33.5 us	390.0 us	
MLC-2	43.3 us	1084.4 us	

stored data has a redundancy ratio larger than 11.0%, this approach can increase SSD's storage density without sacrificing performance and reliability.

5.2 Motivation

Continued capacity growth and lower unit prices have made SSDs a mainstream storage device. However, future increases in density will result in a significant drop in performance and reliability. As a result, SSD manufacturers and users will carefully weigh the cost, performance, capacity, and reliability. In this work, we propose to integrate deduplication in SSD to exploiting its built-in ECC engine to help detect the potentially redundant data at fairly low computation cost.

Redundant data are prevalent in the digital universe, which makes data deduplication and compression viable and profitable. Given the minimum chunking cost of fixed-size chunking in deduplication, it is widely used in flash-based consumer electronics, and we want to know the most effective fixed-size chunking granularity to help detect the duplicate data in these consumer electronics such as laptops and desktops. As such, we have collected the daily office-workload data from two laptops and four desktops applied different fixed chunking sizes to the data, and analyze their data redundancy ratios through a data deduplication engine. As illustrated by Figure 5.1, we have obtained two important observations: one is that there exists a lot of redundant data in these laptops and desktops, which is up to 37.0% on desktop 4; the other is that most redundant data are found in 8 KB chunks, whose size is close to modern SSD's page size. This study affirms that using SSD's page size as the fixed chunking size can detect most redundant data, which in part motivates this work.

Table 5.1 lists the page-level read and write operation latencies on several typical NAND flash chips, and the latency of calculating one SHA-256 hash of an 8KB page on ARM Contex processor running at 400 MHz. Obviously, this hashing operation will add non-trivial latency to the write latency if we adopt traditional SHA-256 as the fingerprint for SSD deduplication. Moreover, the longer write operation may also increase the read latency, which will degrade both the read and write operations if SHA-256 is used as the

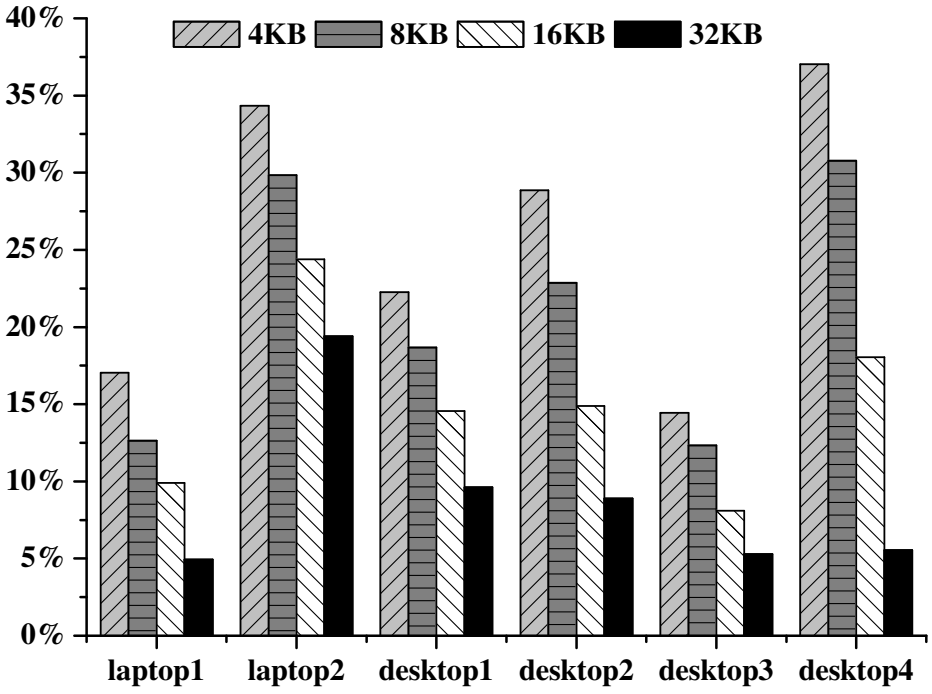

Figure 5.1: Data redundancy rates of fixed-size chunking.

fingerprint for SSD deduplication. As shown in Figure 5.2, the SSD performance drops after enabling SHA-256 based deduplication on different NAND flash chips, because the SHA-256 hashing overhead is incurred on the critical path of every write operation. These mixed read-and-write workloads are generated by the FIO tool to process the input data without any deduplicatable pages to learn its deduplication overheads mainly caused by calculating SHA-256. When we feed our collected datasets with up to 30.7% redundant pages at our selected 8 KB page size, this SHA-256 deduplicatable SSD can slightly improve the baseline SSD's write performance up to 5.8%. By this test, we find that SHA-256 based deduplication in SSDs incurs non-trivial overheads (up to 17.3%), that will significantly offset the most benefits of deduplication and make it unattractive to be integrated within the SSDs.

Meanwhile, NAND flash chips usually face much more transient failures caused by program disturb, read disturb, over-programming or charge loss due to their inherent device-level characteristics. As a result, ECC has become a mandatory feature integrated into SSDs to detect and correct these transient failures to make SSDs useable. The SSD controller will first divide a page into one or more blocks when a host writes a page of data to SSD. Then, its encoding module will generate a codeword consisting of the block and

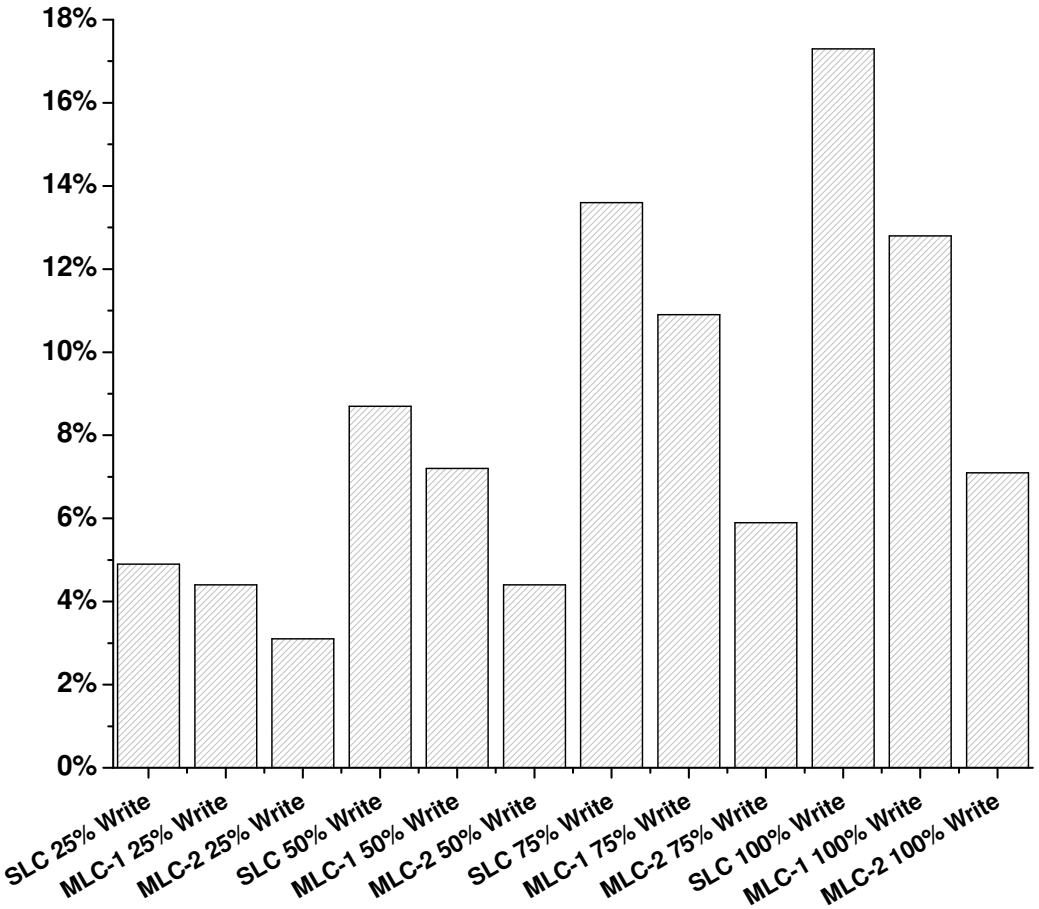

Figure 5.2: SSD performance degradation after adding SHA-256 based deduplication on different types of NAND flash chips with different mixed random read-and-write workloads on fixed chunking of size 8 KB.

ECC for each block by its generator matrix. A page's ECC can be treated as the concatenation of all block ECCs within that page.

In this work, we argue that this kind of ECC information sealed within SSD can be leveraged as the content identification of the stored data in lieu of the costly cryptographic hash computation of the stored data for the purpose of redundancy detection. In general, the computation overhead of calculating ECC is much lower than SHA-256, which motivates us to reuse this ECC as the first-round fingerprint to filter out the most distinct data for SSD deduplication that is backed up by a second-round byte-to-byte comparison to verify the duplicate contents for possible false positives (due to hash collisions). This can help filter out most distinct contents in an efficient way since there is no false negative in ECC value comparisons.

Moreover, within a single SDD, its total number of data pages is not that high (i.e., a 1-TB SSD with 8-KB page will have 1.25×10^8 data pages at most compared with the 10^{14} data pages in an exabyte data store) and we can generally expect a much lower potential hash collision probability by Formula 1.1 in Chapter 1. Meanwhile, we can further exploit the asymmetric latency of read and write operations, which is shown in Table 5.1, to support the byte-to-byte comparison operation. For each write operation, we filter out the distinct data pages whose ECCs don't exist in the fingerprint table, and for those data pages sharing the same ECCs, we will read these already stored data pages and compare them with the currently written data pages byte-by-byte to avoid the possible false positive by adopting ECC as the content hash. In particular, the data deduplication ratio is in the range of 12.6% and 30.8% at the fixed chunking granularity of 8-KB in our collected data from laptops and desktops, which means that most distinct data pages will be filtered out by the first-round ECC comparison, and the second-round byte-to-byte comparison will add an extra page read operation to those write operations that the written pages need a second-round comparison, which is a small portion of total write operations.

5.3 SES-dedup System Design

In this Section, we show the basic design of SES-dedup, which can be implemented at either host-side or device-side plus some necessary modifications in device's FTL.

We observe that the scrambler module usually adopts a Linear Feedback Shift Register (LFSR) to generate a scrambling vector by using the LBA of the scrambled data block as the randomization seed [13] [14]. This scrambling vector will then be XORed with the origin data to generate the scrambled data. Upon a read request, the scrambled data will be descrambled by the same logic because the XOR operation is reversible. Therefore, we can break through this scrambler module by generating the scrambled data at the host-side, so as to perform the on-demand deduplication on SSDs by exploiting their built-in ECCs, whose dataflow view is shown in the lower part of Figure 5.3.

Figure 5.3 shows a high-level architectural view of host-side SES-dedup design. Its right upper part shows the major functional components integrated within a typical SSD controller. In particular, the incoming data will be scrambled before calculating their ECCs, and then written to the chips. This makes ECC-based deduplication impossible because the ECC values of duplicate data blocks with different LBAs will be completely different after passing through the scrambler module. In this case, a deduplicatable SSD must keep extra information, such as the fingerprints, to help deduplication, which will incur both computation and space overheads. As shown in the left upper part of Figure 5.3, we add a

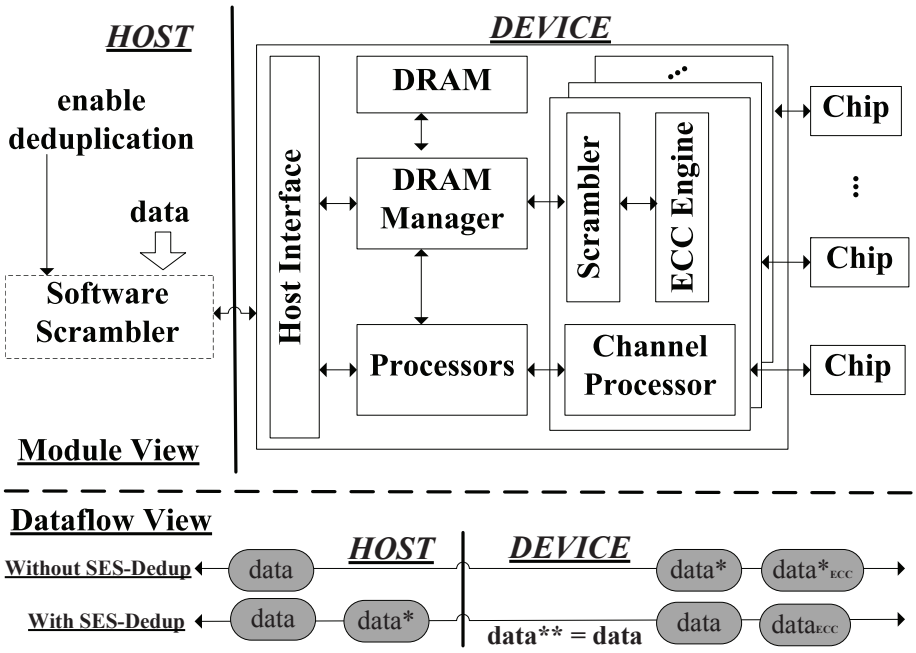

Figure 5.3: Architecture of host-side SES-dedup system.

software scrambler module at the host side to help SSD reverse the randomized data to the original data written by the host and store them on the flash chips.

For a specific SSD controller, we can easily recreate the scrambler in software by writing some predefined data patterns (like all '0') to the SSD. In this case, we know the content of the input data and its LBA, and we can dump the chip's image to obtain the scrambled version of the input data so as to reverse the scrambler function and recreate the software scramble module at the host-side. By this approach, we can perform the on-demand deduplication on SSD by enabling the host-side software scramble module to randomize the data before writing to the SSD, then it will be scrambled again in SSD, and finally storing the origin data on NAND flash chips. Thus leveraging its ECC to detect the duplicate content to perform the fix-sized chunk data deduplication algorithm on SSD.

Host-side SES-dedup system can selectively bypass the embedded scramble module in SSD, but it will store the data without scrambling on the NAND flash chips, which might increase the raw data error rates, although we can modify FTL to help redirect writing the original data to different physical units to avoid/limit writing similar patterns in the same block. In order to overcome this issue, we have proposed the device-side SES-dedup system, which is based on the distributive property of matrix multiplication. In particular, as shown in Equation 5.1, ECC vector is usually calculated by a generator matrix multiplication operation, whose input is a scrambled data vector obtained by origin data XOR the

scramble vector. Based on Equation 5.2, we can obtain the original data's ECC as its content identification because the generator matrix and all scramble vectors are known, which can be integrated within the firmware to support dedup in SSD. As a result, we can implement the device-side SES-dedup system without storing the origin data on NAND flash chips. Once we read the ECC from NAND flash chips, SSD controller can further calculate its origin data's ECC and use it as the fingerprint to detect the duplicate data.

$$([V_{Data}] \oplus [V_{Scrambler}]) \times [M_{Encoding}] = [ECC] \tag{5.1}$$

$$[V_{Data}] \times [M_{Encoding}] = [ECC] \oplus [V_{Scrambler}] \times [M_{Encoding}] \tag{5.2}$$

In SES-dedup system, an ECC-based fingerprint is used as the first-level filter to identify the potential duplicate page, then this page will be processed by a byte-to-byte comparison with the other page's 32-byte sampled data stored in the Out-of-Band area to verify the redundancy. Because the FTL has a lot of other functions (e.g., logical block mapping, wear leveling, garbage collection, write amplification, bad block management, etc.) to perform, there is very limited computation power and memory space left in the SDD controller that can be used for in-line deduplication. As a result, we believe that post-processing deduplication will be a must-have option for SSDs to deduplicate the redundant contents. That is, periodically scanning the SSD to identify and remove the redundant data in the background or during idle periods by leveraging the ECC information as its fingerprint, which saves a lot of costs associated with computing SHA-256 to support deduplication on SSDs.

The integration of the data deduplication feature in FTL will change SSD's logical-to-physical mapping from 1-to-1 to n-to-1. Multiple LBAs will be mapped to a single Physical Block Address (PBA), which is fine for normal read/write operations. However, this will not work for the GC task, because it must notify all associated LBAs that their corresponding PBA's content will be moved to another PBA. As a result, SES-dedup adds a reverse lookup mechanism to check all LBAs associated with a specific PBA for garbage collection.

Host-side and device-side SES-dedup systems are designed for different application scenarios. The host-side design is suitable for personal usage that provides a flexible on-demand interface to enable the deduplication feature on SSDs. There are two main reasons for this application scenario. First, different data generated by different applications have shown to exhibit significantly different data redundancy characteristics, i.e., very little redundancy exists among data generated by different applications [25], making it unnecessary to deduplicate data generated by fundamentally different applications. Second,

applications or file systems may write the same metadata to multiple logical blocks to avoid potential data loss due to single-block failures, which means that devices should not eliminate such intentional data redundancy by the deduplication feature to reduce the risk of data loss. On the other hand, the device-side design is more suitable for large-scale data center, which contains a lot of different SSDs, whose host may not have sufficient computation power to running the software scramble module to support a lot of SSDs simultaneously. Moreover, it can seal the dedup function within the device that provides better compatibility.

Some SSD controllers may integrate a data compression engine to compress the incoming data before writing it to the flash chips. If the compression is performed after SSD's data scrambler module, there is no impact on our host-side SES-dedup design although it will hurt the compression ratio because the data has been randomized. When the compression module is placed before the scrambler module, we should disable the software scrambler module at the host side and write some extra ECC data, leveraging the compression algorithm's ECC as the content's fingerprints rather than the NAND flash's ECC [60], to the Out-of-Band space to help perform data deduplication on SSDs. On the other hand, in our device-side SES-dedup design, compression will not impact its function because all necessary work can be processed within the device.

A page has several codewords, which can provide a finer deduplication granularity at an ECC codeword other than a whole page. It will be especially useful for large flash's page size because we find that the most predominate fixed-size chunking granularity that can detect most duplicate data is around 8 KB. We leave this as our future work.

5.4 Evaluation

In this section, we will introduce the experiment environment, present and analyze our experiment results.

5.4.1 Experimental Environment

We evaluate SES-dedup on GEM5 full system simulator [7], whose SSD model is ported from the extended FlashSim simulator [8] and integrates with ECC-based deduplication functions. We set the major parameters of the host as a 1.6 GHz X86 CPU plus an eight-bank 8 GB DDR3-1600 DRAM. The SSD configurations are listed in Table 5.2, while the read and write latencies on different flash chips are listed in Table 5.1. We shrink stimulated SSD size to 32 GB with 64 MB DRAM to make our collected data easily saturate its capacity. Each codeword of 1 KB is protected by a code rate of 32/33 LDPC code

Table 5.2: Configurations of SSD simulator.

Description	Configuration
Flash Page Size	8 KB
Pages per Block	256
Block per Plane	256
Plane per Package	8
Number of Packages	8
Garbage Collection Threshold	5%
Flash Erase Latency	1.8 ms

(i.e., the coding redundancy is 256 B per 8 KB data page). In particular, calculating SHA-256 hash will take 226.5 us, host-side SES-dedup will not incur extra hashing computation cost on the SSD device and recalculating LDPC ECC will be 14.5 us on a 400 MHz ARM processor for device-side SES-dedup system.

Our data sets are collected from two laptops and four desktops, which contain the typical office workloads, such as coding, file editing, Internet surfing, emailing, file sharing, running virtual machine, etc. These data sets have normal redundancy ratios, which vary from 12.3% to 30.8% at 8KB fixed chunking size. We use the FIO tool [4] to create the synthetic data access traces based on these data sets to evaluate the performance of SES-dedup system.

5.4.2 ECC-based Fingerprint Filter

First of all, we must design a Fingerprint Filter to help reduce the size of the in-memory fingerprint table because SSD's embedded DRAM capacity is limited and cannot hold all fingerprints. In SES-dedup design, we truncate each codeword's ECC from 32 B to 4 B to form a page's ECC fingerprint, which has the total length of 32 B (256-bit). We do not observe any hash false positives by replacing SHA-256 with this 256-bit ECC because an SSD's capacity is small. Specifically, for a 32 GB capacity SSD with 8 KB page size, it only contains 4 M pages, and the 256-bit fingerprint length has provided enough hash space to avoid any collisions, which can even fully meet 8 TB SSD's requirement. However, 4 M 256-bit ECC fingerprints will occupy 128 MB memory, which is larger than the total simulated SSD's DRAM capacity (64 MB).

As shown in Table 5.3, the distribution of duplicated pages is highly skewed in these data sets. We have observed that 12.7% to 18.8% hot duplicated pages, whose reference count is larger than 2, have occupied about 72.1% to 89.3% of total redundant data. In other words, it provides an opportunity to optimize the fingerprint table, thus making it

Table 5.3: Skew-distributed duplicated pages

	Hot Fingerprint Ratios	Ratios in Redundant Data
laptop1	17.6%	74.1%
laptop2	13.8%	86.3%
desktop1	15.8%	79.8%
desktop2	14.9%	81.1%
desktop3	18.8%	72.1%
desktop4	12.7%	89.3%

small enough while containing most duplicated fingerprints. In order to achieve this goal, we have designed a fingerprint table that can store 15.0% of SSD's total number of the fingerprint. By this approach, the fingerprint table shrinks from 128 MB to 19.2 MB, but still occupying a lot of DRAM capacity because most DRAM is used to store FTL's mapping table. We further reduce each fingerprint entry's length to shrink the fingerprint table's size. In this design, we sample a quarter of the ECC-based fingerprint, which reduce fingerprint table size to 4.8 MB that can be fit within the limited DRAM (around 7.5% space overhead).

Figure 5.4 shows various random write performance improvements on simulated SLC SSD with different fingerprint table size ratio while 10% means this fingerprint table can store 10% of all possible fingerprints (0.4 M in this test), the MLC results with the same trend are omitted due to the space limit. From this test, we find that different data sets exhibit different random write performance improvements due to their different duplicated data distributions. When this table size ratio increases from 15% to 20%, the performance gains are diminishing, thus indicating 15% of max table size can obtain the best price/performance ratio, which is mainly determined by the skew distribution of duplicated pages. Specifically, SES-dedup system can improve up to 17.0% random write performance under this setting.

5.4.3 Effectiveness of Deduplication

SES-dedup can in-line deduplicate every redundant page (up to 30.8% duplicate data at 8 KB fixed chunking size) without considering the limited computation power of embedded SSD controller. In order to reduce the performance interferences caused by the in-line deduplication processing, both host-side and device-side SES-dedup systems try their best to processing deduplication in-line while leaving the other pages to be processed off-line. As a result, we want to know how much redundant data can be detected by in-line dedupli-

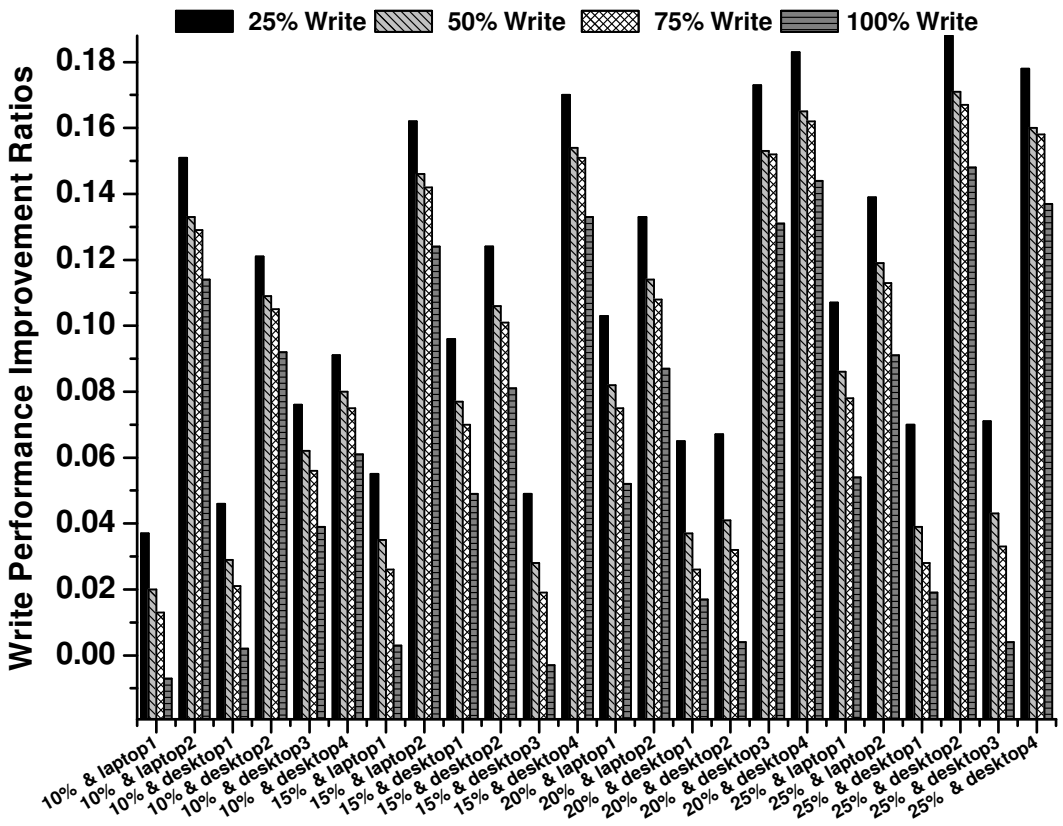

Figure 5.4: Study of write performance improvements on different fingerprint table size

cation because it will let us know several important metrics like how many P/E operations are saved, how much space can be reclaimed by post-processing deduplication, and so on.

The host-side SES-dedup system is controlled by the host-side module, which will limit the contents to be deduplicated on SSDs. It adds negligible overheads to check the fingerprint table for deduplication. We have collected the in-line and off-line deduplication processing ratios in Table 5.4. For example, 52.9% to 59.8% duplicated data is processed by in-line deduplication under SLC SSD. In other words, it can directly reduce write 6.5% to 18.2% data to SLC chips, which will reduce the corresponding P/E operations. Meanwhile, those remained duplicated page to be processed by off-line (post-processing) deduplication will further increase about 5.5% to 12.6% storage capacity on SLC SSD. In this case, we don't compare it with SHA-256 based deduplication because we don't change the device by adding extra hardware/software module to calculate SHA-256 except for some necessary deduplication changes, such as adding the fingerprint table and modifying FTL's mapping table. From this test, the MLC SSDs process less in-line deduplication data because its inherent access latency is higher than SLC device, which prolongs its write latency.

Table 5.4: In-line and off-line deduplication processing redundancy data ratios on the host-side SES-dedup system with 100% random write workload.

Data Set	In-line Dedup			Off-line Dedup			Duplicate Ratio
	SLC	MLC-1	MLC-2	SLC	MLC-1	MLC-2	
laptop1	7.1%	6.5%	5.4%	5.5%	6.1%	7.2%	12.6%
laptop2	17.4%	16.1%	12.9%	12.5%	13.8%	17.0%	29.9%
desktop1	11.0%	9.9%	8.1%	7.7%	8.8%	10.6%	18.7%
desktop2	13.7%	12.1%	9.9%	9.2%	10.8%	13.0%	22.9%
desktop3	6.5%	6.1%	5.2%	5.8%	6.2%	7.1%	12.3%
desktop4	18.2%	16.9%	13.6%	12.6%	13.9%	17.2%	30.8%

Different from the host-side approach, the device-side SES-dedup system will add the ECC processing latency to support its deduplication function. As shown in Figure 5.5, we draw the geometric means of in-line deduplication ratios of different data sets under different mixed read-and-write workloads. We find that majority of duplicated pages can be detected and removed inline while leaving some pages to be processed off-line in ECC-based SES-dedup approach, which means it will reduce the corresponding duplicate writes to the NAND flash chips. Specifically, it can process 19.9% to 42.8% more duplicated data in-line than SHA256-based approach, which means a lot of P/E operations can be saved by this ECC-based approach.

5.5 Conclusion

In this work, we propose the SES-dedup system to help bypass the data scrambler module within SSD to enable the low-cost ECC-based data deduplication on SSD. Our experimental results show it can extend the flash space by a factor of up to 30.8% with up to 17.0% write performance improvement than baseline SSD with our collected real-world data sets.

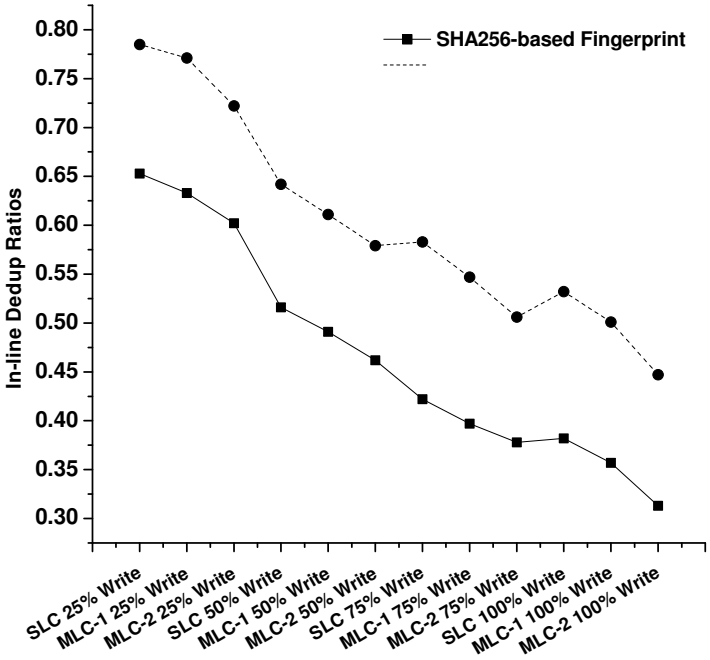

Figure 5.5: In-line deduplication ratios of device-side SES-dedup with different hash fingerprints.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

In this chapter, we will summarize our main contributions of the technologies proposed in this thesis and discuss the possible future work that can further strengthen the versatile deduplication system.

6.1 Contributions

The main contributions of this dissertation can be summarized as these four aspects. First, it tries to enable the deduplication system to process the popular cloud storage workloads such as compressed file and images to optimize the cloud storage efficiency. Secondly, besides the traditional bitstream's fingerprint-based redundancy-identification, it tries to exploit the deeper metadata information to assist the deduplication. Thirdly, unlike all existing lossless deduplication approaches, this dissertation proposes the first lossy deduplication approach to process the redundant image contents by eliminating their perceptual-equivalent contents at the user's perspective. Finally, it deeply analyzes the existing SSD's architecture and reuses the built-in ECC module to enable the deduplication at fairly low-cost on SSDs.

6.2 Future Work

This dissertation open a new dimension to make the deduplication system be able to process the files previously ignored by existing deduplication approaches. The main pain point lies in the inherent drawbacks on existing redundancy-identification mechanism. In this dissertation, we find that different kinds of data may have their own unique features that can be exploited to help detect the duplicate contents. As a result, it is necessary to analyze the popular data types and to enable the deduplication to process these data. We believe it may obtain some amazing results by combing the deep learning techniques to automatically learn the redundancy-identification mechanism from the raw data. Moreover, since the data will finally be stored on the storage media, as more new storage media, such as shingled magnetic recording (SMR), magnetoresistive random access memory (MRAM) and XPoint appeared to be the future storage devices, it is necessary to consider how to deploy the

low-cost deduplication on these new storage devices. Especially for the NVMe-based low-latency storage devices, it is critical to reducing the deduplication latency to fully exploit these high-performance storage devices.

REFERENCES

[1] Google images. https://images.google.com/.

[2] Tineye reverse image search. https://www.tineye.com/.

[3] Performance and tuning considerationsfor sas on pure storage fa-420 flash array, 2014. SAS Institute Inc.

[4] J. Axboe. Flexible i/o tester.

[5] H. Bay, T. Tuytelaars, and L. Van Gool. *SURF: Speeded Up Robust Features*, pages 404–417. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[6] W. A. Bhat. Is a data-capacity gap inevitable in big data storage? *Computer*, 51(9):54–62, September 2018.

[7] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[8] M. Bjørling. Extended flashsim. 2011.

[9] S. Boboila and P. Desnoyers. Write endurance in flash drives: Measurements and analysis. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, pages 9–9, 2010.

[10] J. Bonwick. Zfs deduplication. https://blogs.oracle.com/bonwick/entry/zfs_dedup, 2009. Online; accessed 1 November 2018.

[11] S. Bounkong, B. Toch, D. Saad, and D. Lowe. Ica for watermarking digital images. *J. Mach. Learn. Res.*, 4(7-8):1471–1498, Oct. 2004.

[12] Y. Cai, S. Ghose, and et.al. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proceedings of the IEEE*, pages 1666–1704, 2017.

[13] J. Cha and S. Kang. Data randomization scheme for endurance enhancement and interference mitigation of multilevel flash memory devices. *ETRI Journal*, 35(1):166–169, 2013.

[14] C. Chang, M. Bekerman, I. Swarbrick, and C. Hanson. Xor-based scrambler/descrambler for ssd communication protocols, 2016. Patent: US20160328567A1.

[15] A. Cheddad, J. Condell, K. Curran, and P. M. Kevitt. Digital image steganography: Survey and analysis of current methods. *Signal Processing*, 90(3):727 – 752, 2010.

[16] F. Chen, T. Luo, and X. Zhang. Caftl: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, pages 6–6, 2011.

[17] Cisco. Cisco visual networking index: Forecast and methodology, 2015ÂÍc2020. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html, 2016. Online; accessed 1 November 2018.

[18] P. Deutsch. Rfc1951 deflate compressed data format specification version 1.3. https://www.ietf.org/rfc/rfc1951.txt, 1996. Online; accessed 1 November 2018.

[19] W. Dong, Z. Wang, M. Charikar, and K. Li. High-confidence near-duplicate image detection. In *Proceedings of the 2Nd ACM International Conference on Multimedia Retrieval*, ICMR '12, pages 1:1–1:8, New York, NY, USA, 2012. ACM.

[20] T. Economist. The world's most valuable resource is no longer oil, but data. https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data, 2017. Online; accessed 1 November 2018.

[21] 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. Online; accessed 1 January 2017.

[22] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[23] J. J. Foo, J. Zobel, R. Sinha, and S. M. M. Tahaghoghi. Detection of near-duplicate images for web search. In *Proceedings of the 6th ACM International Conference on Image and Video Retrieval*, CIVR '07, pages 557–564, New York, NY, USA, 2007. ACM.

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

[25] Y. Fu, H. Jiang, and et.al. Aa-dedupe: An application-aware source deduplication approach for cloud backup services in the personal computing environment. In *2011 IEEE International Conference on Cluster Computing*, pages 112–120, 2011.

[26] J. Gantz and D. Reinsel. 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. Online; accessed 1 November 2018.

[27] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf. Characterizing flash memory: Anomalies, observations, and applications. In *42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 24–33, 2009.

[28] F. Guo and P. Efstathopoulos. Building a high-performance deduplication system. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, pages 25–25, 2011.

[29] T. Hey, S. Tansley, and K. Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. October 2009.

[30] D. R. Horn, K. Elkabany, C. Lesniewski-Lass, and K. Winstein. The design, implementation, and deployment of a system to transparently compress hundreds of petabytes of image files for a file-storage service. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation*, pages 1–15, 2017.

[31] Y. Hua, H. Jiang, and D. Feng. Fast: Near real-time searchable data analytics for the cloud. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 754–765, Piscataway, NJ, USA, 2014. IEEE Press.

[32] S. Iizuka, E. Simo-Serra, and H. Ishikawa. Globally and locally consistent image completion. *ACM Trans. Graph.*, 36(4):107:1–107:14, July 2017.

[33] P. Inc. .zip file format specification (version 6.3.4). https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT, 2014. Online; accessed 1 November 2018.

[34] Y. T. Jin, S. Ahn, and S. Lee. Performance analysis of nvme ssd-based all-flash array systems. In *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 12–21, 2018.

[35] B. Johannes. Imagehash. https://github.com/JohannesBuchner/imagehash.git.

[36] J. Kim, C. Lee, and et.al. Deduplication in ssds: Model and quantitative analysis. In *Proceedings of the 28th IEEE Symposium on Mass Storage Systems and Technologies*, pages 1–12, 2012.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[38] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky. Silt: A memory-efficient, high-performance key-value store. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 1–13, New York, NY, USA, 2011. ACM.

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

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

[41] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

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

[43] C. Ming, W. Shupeng, and T. Liang. A high-precision duplicate image deduplication approach. *JOURNAL OF COMPUTERS*, 8(11):2768–2775, Nov 2013.

[44] M. Nelson and J.-L. Gailly. *The data compression book (2nd edition)*. IDG Books Worldwide, Inc.

[45] I. Pavlov. 7-zip. http://www.7-zip.org/. Online; accessed 1 November 2018.

[46] C. Policroniades and I. Pratt. Alternatives for detecting redundancy in storage systems data. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATC)*, pages 6–6, 2004.

[47] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In *Proceedings of the 1st USENIX Conference on File and Stroage Technologies (FAST)*.

[48] D. Reinsel, J. Gantz, and J. Rydning. Data age 2025: The evolution of data to life-critical don't focus on big data; focus on the data that's big. https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf, 2017. Online; accessed 1 November 2018.

[49] Y. Romano, J. Isidoro, and P. Milanfar. Raisr: Rapid and accurate image super resolution. PP, 06 2016.

[50] A. Roshal. Rar. http://www.rarlab.com/. Online; accessed 1 November 2018.

[51] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[52] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. The first collision for full sha-1. In J. Katz and H. Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 570–596. Springer International Publishing, 2017.

[53] P. Storage. Pure storage flasharray. http://info.purestorage.com/rs/225-USM-292/images/Pure_Storage_FlashArray_Datasheet.pdf, 2015. Online; accessed 1 November 2018.

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

[55] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky. Deep image prior. *CoRR*, abs/1711.10925, 2017.

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

[57] B. Walker. Every day big data statistics - 2.5 quintillion bytes of data created daily. `http://www.vcloudnews.com/wp-content/uploads/2015/04/big-data-infographic1.png`, 2015. Online; accessed 1 November 2018.

[58] X. J. Wang, L. Zhang, M. Liu, Y. Li, and W. Y. Ma. Arista - image search to annotation on billions of web photos. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2987–2994, June 2010.

[59] X. J. Wang, L. Zhang, and W. Y. Ma. Duplicate-search-based image annotation using web-scale data. *Proceedings of the IEEE*, 100(9):2705–2721, Sept 2012.

[60] Z. Yan, H. Jiang, Y. Tan, and H. Luo. Deduplicating compressed contents in cloud storage environment. In *Proceedings of the 8th USENIX Conference on Hot Topics in Storage and File Systems*, 2016.

[61] A. J. Zargar, N. Singh, G. Rathee, and A. K. Singh. Image data-deduplication using the block truncation coding technique. In *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, pages 154–158, Feb 2015.

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

[63] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.

[64] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, Sep 1978.

BIOGRAPHICAL STATEMENT

Zhichao Yan received his Ph.D. in Computer Science and Engineering from the University of Texas at Arlington at 2018. Prior to beginning the Ph.D. program, Zhichao obtained his Ph.D degree from Huazhong University of Science and Technology, China in 2012 in Computer Engineering. His main research interests are computer architecture, storage system and data deduplication. He has published several papers in the top tier conferences in the literature such as the USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage), International Conference on Massive Storage Systems and Technology (MSST) and International Conference on Parallel Processing (ICPP).