International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 13 (2017) pp. 4015-4022

COVERED: Content-Version based Removal of Duplicates

Jyoti Malhotra[1] and Jagdish Bakal[2]

[1,2]Department of Computer Science & Engineering,
G. H. Raisoni College of Engineering, Nagpur,
RTM University Nagpur, MS, India

[1]ORCID: 0000-0002-1147-4549, Scopus Id: 56890914200

Abstract

Nowadays deduplication is becoming a promising way to provide more storage space by wiping out the unwanted data, particularly duplicate and similar data copies. The similar data copies, an integral part of data versioning is seen everywhere. This paper presents a post process key to integrate data versioning, deduplication and data archiving. Version and content-based similarity detection are attained by finding the similarity scores using shingles and cosine similarity. Data from primary storage is shedded by devolving the older versions as ghost entries to archive. A novel probability model is presented which decides the permanent removal of ghost entries as per their access probabilities. The described work is evaluated on the real and synthetic datasets and active storage space was successfully released by diverting unwanted data to archive.

Keywords: duplicate, post-process, version, checksum, similarity, archive

INTRODUCTION

According to IDC [1], the evolution of data has noted a critical growth from terabytes to zetabytes in the recent years. Large data has been populated from various sources and in various formats. This digital growth makes it difficult to store and manage the data with new storage technologies such as deduplicated storage. Data backup and storage solutions run into a common paradox. Data retention is achieved by keeping multiple copies of data at multiple different locations. Organizations and individuals do not want to tie up their storage devices as it occupies more space. Data deduplication mediates this paradox and plays an important role in creating more storage space for the devices; where a dichotomy appears between the data as duplicates and non-duplicates. Duplicates are the ones which have the maximum or exact similarity, which is later removed keeping only one copy of data.

Non-duplicates can be further classified as close duplicates, where there exists significant similarity in the data; and far duplicates, where files are either unique or very less similar. Focusing on the local or personal storage solutions; the user has a tendency to store multiple copies at multiple locations. Sometimes, various versions of the same file are stored at the different locations with minor semantic modifications; creating multiple similar copies. Users hardly refer these versions, because they need the most recently modified data. Here comes the importance of removing these unwanted or unaccessed files and mounting the storage capacity to store new files.

Storage capacity can be augmented either by compressing the data or by deleting the data whose contents are rarely accessed, or by deleting the data with multiple versions keeping the recent one or by deleting the exact duplicates. Third party data compression techniques such as WinZip, Winrar, tar, etc. helps in reducing the file size thus helping in reducing storage space, but it is a time-consuming episode and gives a better result if there are duplicate content within the file. Likewise, deleting the data which is accessed rarely may result in losing the only available copy. Whereas in deduplication, the application looks at the content of the file to find its photocopy; if found only one instance of data is saved, and metadata pointers are set for the duplicates, further deleting the duplicates.

The timing of finding the duplicates is one of the key dimensions in the deduplication process [2]. Timings are defined as namely – inline and post-process deduplication. A process of finding duplicates as a part of a regular storage request is called as inline deduplication. Where, duplicate contents are discarded at the time of storing the data and this result in higher resource utilization such as storage time, computation energy etc and also enlarge the backup window. While, in post-process deduplication, an application at regular storage request, writes the data directly to its storage space; and later a scheduled process is executed periodically on this stored data to interrogate for the duplicates as shown in Figure 1.

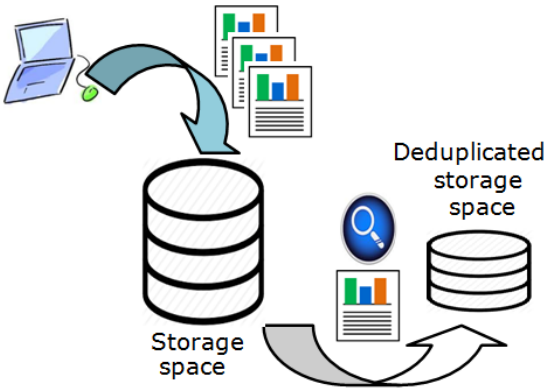

Figure 1: Post-process timing of deduplication

Deduplication operates on all kind of files and all type of database records. However, deduplication on database records has not concerned in the scope of this paper. In this paper, we put forward two-phase solution towards duplicate elimination namely, instant removal of exact duplicates by comparing their content-based hash values [13] and preserving a copy of close duplicates or similar content as the ghost entries which are deleted if they remain un-accessed for a particular period, freeing the storage space.

This paper aims to remove the duplicates from a personal collection of various files and a synthetic dataset generated according to the principles discussed in [12]. Our contribution includes:

- Content-based and version similarity based post-process duplicate elimination.
- A probability model, which is deciding the existence factor of files residing in the ghost list.
- Evaluate the performance of various versions of files. The result shows that the process works better with a small comparison window.

The organization of rest of the paper is as follows: next section concise the existing work on similarity. The further section describes the Content-Version (CV) based duplicate elimination algorithm. A probability model of ghost entry existence is discussed later. After that the next section presents the performance evaluation, and finally, conclusion and scope for future work are summarized in the end.

RELATED WORK

This section portrays the gist of prior work done in context to similarity and deduplication. Area of work ranges from finding similarity in the blocks of the file, superblocks of the file or to finding similarity in the entire file.

Ricardo Koller et. al [3] proposes a storage solution using data similarity to improve I/O operations thereby reducing the seek time. The author accomplishes an improvement in I/O operations by the content-based cache, where cache replacement decisions are made on the data content rather than its location. Content-hash values are getting stored in the cache, and these entries are indexed by a list of locations where the content is duplicated. Further for content retrieval, the list is referred to redirect IO requests based on the number of duplicate references. Authors have assumed similar content as duplicate content, and the deduplication strategy is used to avoid duplicates in the cache, thus improving cache efficiency and disk access.

Inline deduplication for primary storage assuring data integrity is discussed in [4] by eliminating redundant data using their similarity values. Dimension for comparing redundancy is minimized as similar files are grouped into same categories. The file is divided into fixed length blocks, after which these blocks are further divided into multiple sequences, and a rolling hash checksum is calculated for every sequence. For every segment, a category identity is calculated. Segments with same category identity are grouped together; which helps to achieve more duplicate removal ratio. It is observed that deduplication throughput is better with small segment size, whereas read throughput is better with large segment size.

In [5], the similarity and locality approaches are addressed to find more duplicates with less disk access. Work is divided into two phases namely, similarity detection and redundancy elimination. Similarity detection helps to achieve more duplicates, as it targets only similar blocks. The author aims to achieve good deduplication ratio with larger file blocks.

Delta compression techniques for deduplication are investigated in [7], [8]. This work is extended as the super-feature approach as discussed in [6] to achieve efficient resemblance detection using duplicate-adjacency information. It implements a good delta compression by considering unique segments neighbouring to non-unique ones as similar, thus improving locality and similarity. The concept of duplicate elimination using text similarity is more explored in [9], [10], [11] for the various applications namely semantic similarity of texts, finding duplicate bugs in the bug report based on text similarity and measuring similarity in multi-source downloads. In [11], the author aims to find more similarity in media files which includes music and video content. However, this paper targets to remove duplicates from the non-media files. Understanding the demand of data integrity and data recovery of multiple versions of the same data, the author proposes a multi-version checkpoint method [15] on the multiple versions of the data. Duplicate content is managed by supporting duplication and tracking of duplicate chunks and avoiding unwanted rollbacks.

COVERED: CONTENT- VERSION BASED REMOVAL OF DUPLICATES

The flow of Content-Version (CV) duplicate removal process as depicted in Figure 2 is aligned with the Boolean rule which states when there is a match between the file A and B as illustrated in $Rule_1$ below:

$Rule\ 1$: $isMatch(A, B) \rightarrow \{True, False\}$

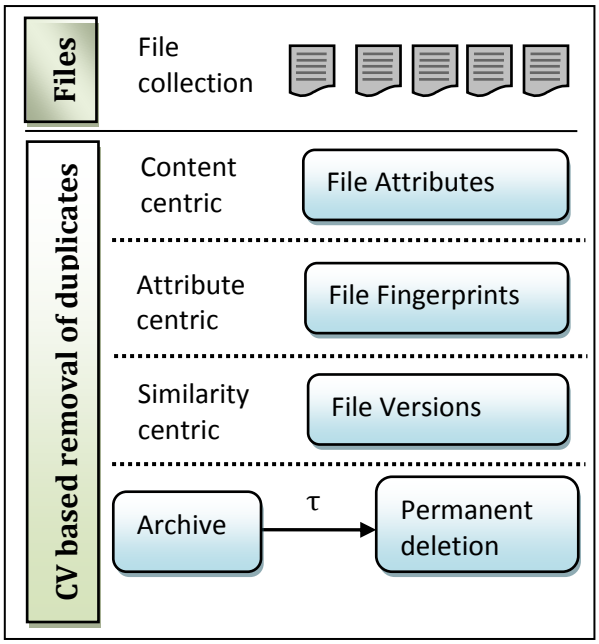

Figure 2: Flow of version content duplicate removal process

This file match can be further extended by comparing the file attributes $Attr_i$, file fingerprint FP_i or file versions Ver_i as mentioned in the $Rules\ 2, 3, 4\ and\ 5$ below. Notations used for file i are illustrated in Table I.

Table I: Notation List

Notation	Meaning
$Attr_i$	File attributes
FP_i	File content fingerprint
Ver_i	File version
cd	Creation date
lat	Last access time
lmt	Last modified time
fs	File size
Sim_{file}	Similarity score of file
θ	Similarity constant threshold between the files
τ	Time threshold deciding file existence factor in the archive

$Rule\ 2$: $isMatch(Attr_A, Attr_B) \rightarrow \{True, False\}$

$isMatch(cd_A, lat_A, lmt_A, fs_A;\ cd_B, lat_B, lmt_B, fs_B)$

$Rule\ 3$: $isMatch(FP_A, FP_B) \rightarrow \{True, False\}$

$Rule\ 4$: $(FP_A = FP_B) \vee (Attr_A = Attr_B) \rightarrow \{True, False\}$

$Rule\ 5$: $Max(Sim_{Ver_A}, Sim_{Ver_B}) \Rightarrow Match$

Rule 2 is based on the commonly used file attributes namely creation date, last access time, last modification time and file size; which is prone to false positive i.e. equal files are evaluated as non-equal if the file attributes are different. Rule 3 is based on the file level veracity, where equality of two files depends on the comparison of their content based shorthand hash values [13] called as fingerprints. This rule does not work well with the structured documents, such as Microsoft Word and Adobe Portable Document Format (PDF) files. Every doc and pdf files have its own objects, such as header metadata, text body, trailer, media and many more [14]. They differ in the metadata holding the document attribute properties such as file name, file size, author, creation date, last access time, last modified time, etc; which makes them unsuitable candidates for checking their equality using traditional fingerprint approach. The process of deduplication on these types of files can be done by decomposing them into the mentioned objects and then compute the fingerprint of each object and authenticate their equality; this is a resource consuming approach. Whereas novel approach to find the equality can be measured by finding the similarity among the files, which is discussed in the section below.

The match Rule 2 and 3 are formulated to derive Rule 4. Rule 5, a content-centric notion of applying file deduplication to the file versions; if multiple files, either of the same name by suffixing its version tag or different name is stored in the same folder or another folder, it will dedupe them based on their similarity scores. The file with maximum similarity score and latest modification date is kept for use and the other copy is deduped i.e. it is deleted from the primary location and is preserved in archive.

SIMILARITY-CENTRIC VERSION DEDUPLICATION

Deduplication performance is directly proportional to the degree of similar content and their similarity scores. This paper discusses and measures the performance by two methods namely, n-gram shingling and cosine similarity on

the ubiquitous text data. Deduplication process is achieved in two stages – preprocessing and compute similarity.

Preprocessing can be further achieved in two ways, namely shingling and vector model.

Shingling

Shingling a file creates the bag of word sequences; hash values of these n-word sequences referred as n-grams are compared for the similarity. As a known fact, deduplication performs well with added similar content. The probability of getting more similarity is based on the probability of getting the same word sequences. The probability of getting the last word k given previous k-1 words is related to the joint probability of words in the file as shown in Equation (1).

$$P(File_{word_{seq}}) = P(W_k|W_{k-1}) \quad (1)$$

i.e. probability of the sequence of words is the product of the probability of each word times the probability of prefix until that word is illustrated in Equation (2).

$$P(File_{word_{seq}}) = \prod_k P(W_k|W_1W_2W_3 \dots W_{k-1}) \quad (2)$$

File preprocessing and similarity computation steps with respect to shingles are listed in Algorithm 1. Files A and B are segmented into array of words $\text{word}_i[\,]$; for $i = \{A, B\}$. For both the files, hash values $Hash_i$ of n-gram shingles are calculated from this word array as shown in Equation (3). Later, shingles checksum distance (SCD) is measured based on Jaccard coefficient and is compared with the predefined similarity constant threshold θ. If SCD of the files is greater than equal to θ, file is tagged as similar otherwise non-similar. SCD is the similarity distance metric of two files based on the n-gram shingles and their hash checksums.

$$For\ i = \{File_A, File_B\} \quad (3)$$

$$Hash_i = \sum_{i=0}^{files(A,B)} \sum_{j=0}^{file\_len} \sum_{k=0}^{Shingle_{len}} File_i(Hash_j, Hash_{j+k})$$

Vector Model

Files are represented in an algebraic vector model; these vectors are then compared to their vector cosine distance (VCD). VCD is the distance metric of two files based on their vector and cosine similarity.

Cosine similarity is a measure of similarity between two vectors; it is bounded between 0 and 1. It is measured by taking a cosine of vector angle. If vector angle is 0 degree then the documents are exact equal. If vector angle is 90 degree, means that the documents are exactly different. Two documents are equivalent if the angle between them is less and cosine of an angle is more. VCD for files A and B is the ratio of inner dot product of the file vectors ($Vect_A, Vect_B$) to the product of the two vector lengths. If VCD of files is greater than equal to θ, file is tagged as similar otherwise non-similar. File preprocessing and similarity computation steps with respect to vector model are listed in Algorithm 2.

Algorithm 1: SCD- Shingles Checksum Distance

Input: Source file (F_A); Target file (F_B)

Output: Similarity tag = {similar, non-similar}

Begin

for files F_i ; $i = \{A, B\}$

$\text{word}_i[\,] \leftarrow$ Segment_file(i)

$j \leftarrow 0$

while(EOF of F_i)

for $k \leftarrow 0$ to $shingle_{len}$ do

$Hash \leftarrow \text{word}_i[j + k]$

end for

$Hash_i[\,] \leftarrow Hash$

$j \leftarrow j + 1$

end while

end for

$$SCD(F_A, F_B) = \frac{Hash_A \cap Hash_B}{Hash_A \cup Hash_B}$$

if ($SCD(F_A, F_B) \geq \theta$) then ;

Files are similar;

End

Algorithm 2: VCD- Vector Cosine Distance

Input: Source file (F_A); Target file (F_B)

Output: Similarity tag = {similar, non-similar}

Begin

for files A and B

$Vect_A \leftarrow$ Convert File A to Vector

$Vect_B \leftarrow$ Convert File B to Vector

end for

//Measure angle between the files using cosine distance

$$VCD(F_A, F_B) = \frac{Vect_A . Vect_B}{|Vect_A||Vect_B|}$$

if ($VCD(F_A, F_B) \geq \theta$) then

Files are similar;

End

Different versions of the same files result in the multiple copies and probably user is in the need of the most recently modified copy. After finding the similarities using SCD or VCD metric, the older versions of the similar files with maximum similarity and older last modification time are shipped to the archive, a passive data repository.

DATA ARCHIVATION AND ITS PROBABILITY MODEL

In order to handle the data load from the capacity viewpoint; unwanted files are disposed off. File disposal can be either deleting the files permanently at one go or archiving them as the ghost entries. Data archiving is not a major new technique; it uses the old stop-gap technology of offloading the unwanted data to other location. Archiving files is an integral part of COVERED, a deduplication model that seeks to limit the unnecessary increase in the storage space. Archiving allows the application to dispatch the multiple versions of similar files except for the last modified one to other location. The application does not delete the file from the backup device; files are kept so that they can be always referred later if required. The files will be permanently deleted from the archive if they remain un-accessed for some period of time τ (in days/months/years). Files once deleted from the archive cannot be recovered. This offloading process results in more free storage space.

Probability model deciding the permanent removal of file based on the counter which counts the number of file access in a given time interval τ is described below.

Considering this length τ, we need to find the probability of getting random number of access to the files. As length τ can be in the days or months; these random accesses having a probability distribution $P(k, \tau)$ stating the probability of getting k access in interval of τ can be modeled by considering the small time interval δ and then moving ahead with the big time interval τ. We need to find the number of file access for the small time interval δ, followed by τ. During a very small value of δ, there is a probability that the file is getting some k accesses, which is $\alpha * \delta$ where α can be a access rate i.e. expected number of access per unit time.

The probability of getting n access in interval δ can be referred by the notion expressed in Equation (4); which is $\alpha\delta$ for exactly k access, the remaining probability goes to zero access. When δ is small, the probability of more than k access can be approximated to 0.

$$P(n, \delta) = \begin{cases} \alpha\delta, & n = k \\ 1 - \alpha\delta, & n = 0 \\ 0, & n > k \end{cases} \quad (4)$$

Expected number of access during a little interval δ is given by Equation (5).

$$E(\#\ of\ access) = \alpha\delta . \mathrm{k} = \alpha\delta\ (\text{if } \mathrm{k} = 1) \quad (5)$$

Extending this probability distribution of small intervals for the large time interval τ in our case; total number of intervals is illustrated in Equation (6). The expected number of file access in the big interval τ can be considered as the summation of file access in the small intervals δ.

$$Number\ of\ intervals = \eta = \frac{\tau}{\delta} \quad (6)$$

Probability of getting k access in the big interval τ referring Equation 4 and 6 is illustrated as below in Equation (7), (8) and (9).

$$P(k, \tau) = \binom{\eta}{\mathrm{k}} (\mathrm{p})^{\mathrm{k}} (1 - \mathrm{p})^{\eta - \mathrm{k}} \quad (7)$$

Where, $(p)^k$ is the probability of k[th] access = $\alpha\delta\ and\ \eta = \frac{\tau}{\delta}$ and we get,

$$P(k, \tau) = \binom{\eta}{\mathrm{k}} \left(\frac{\alpha\tau}{\eta}\right)^{\mathrm{k}} \left(1 - \frac{\alpha\tau}{\eta}\right)^{\eta - \mathrm{k}} \quad (8)$$

$$\text{i.e. } P(k,\tau) = \frac{(\alpha\tau)^k e^{-\alpha\tau}}{k!} \quad (9)$$

For example considering $\alpha = 1$, $\tau = 3.5$ (access in 7 days; scaled as 1 day =0.5) and various values of k in Equation (9), we get,

k=0 $P(0,\tau) = \frac{(1*3.5\tau)^0 e^{-3.5}}{0!} = 0.03$

k=1 $P(1,\tau) = \frac{(1*3.5\tau)^1 e^{-3.5}}{1!} = 0.10$

k=2 $P(2,\tau) = \frac{(1*3.5\tau)^2 e^{-3.5}}{2!} = 0.18$; and so on

As τ is a fixed value; the summation of probabilities of every k access is equal to 1 as shown in Equation 10.

$$\sum_k P(k,\tau) = 1 \quad (10)$$

Permanent deletion of the file can be measured by finding the probability of zero or a few access in the given time interval. Referring Equation (9), if the value of k is zero or minimum; then the file can be permanently eradicated from the ghost entry, resulting in maximizing the storage space.

RESULTS AND DISCUSSIONS

This section shows the presentation of the results obtained with various datasets. Datasets used in the testing of COVERED are files from [16] for finding exact duplicates and deleting the copies, pdf version files from [17] and a private collection of word, pdf and text files, which is synthesized to introduce multiple versions.

Data sets were tested with both the approaches namely SCD and VCD i.e. with shingles and cosine similarity. Figure 3 shows the similarity percentage of shingle based SCD and cosine based vector model, VCD on word and text files. As the duplicate content was created manually; both the methods detect a acceptable level of similarity percentage. Results observed a slight variation in similarity percentage of approximately 3-5% as SCD deals with the probability of a sequence of words and finds similarity on short-length content; while cosine similarity is based on the frequency of the terms and document.

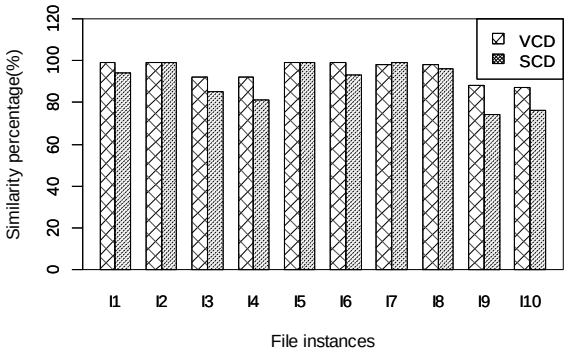

Figure 3: Similarity percentage of shingles and vector model for multiple word file instances

Figure 4 and 5 shows the probability density plot and cumulative distribution of similarities for different pdf files obtained with shingles and cosine distance. Sample files were taken from [17]; where each pdf file set (A, B, C) has 7 versions and each version differed from its previous versions. Table II illustrates the version history and the similarity score in the percentage of n^{th} version against the previous versions.

Table II: File Version History

Version/ File Set		V7	V6	V5	V4	V3	V2	V1
A	SCD%	62	61	64	55	62	46	15
	VCD%	93	91	94	87	91	82	31
B	SCD%	71	61	57	53	40	39	38
	VCD%	99	96	93	93	92	85	85
C	SCD%	57	41	60	58	59	54	57
	VCD%	96	87	95	94	94	95	95

Figure 4 and 5 shows the bell curve and slope of the similarity scores with distributed areas of SCD and VCD. Continuous distribution of the similarity scores obtained from SCD ranges from 40 to 100; a little similarity distribution is also seen in the range below 40 as can be referred from the Table II. On the other hand, continuous distribution of the similarity scores obtained from VCD limits from 30 to 99; majorly it is distributed in the integrals of 75 to 99. This distribution helps in deciding the threshold θ; which is dynamic. Furthermore, for SCD it can be considered as 95+ and for VCD it can be observed as 98+. If there are only two versions and $SCD(F_A, F_B) \geq \theta$ then earlier version can be moved to archive. With multiple versions of the same file θ, is the

maximum score among the versions. Referencing to the values from Table II; considering the example of Set A.

Case 1: SCD-

Threshold θ = maximum (62, 61, 64, 55, 62, 46, 15) is the score 64 of version 5.

Case 2: VCD-

Threshold θ = maximum (93, 91, 94, 87, 91, 82, 31) is the score 94 of version 5.

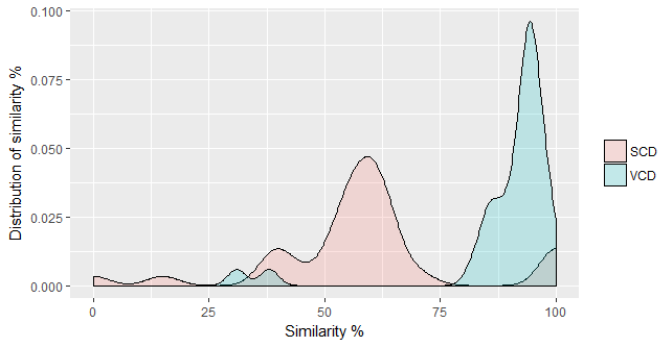

Figure 4: Distribution of similarity percentage of shingles and vector model for pdf file instances

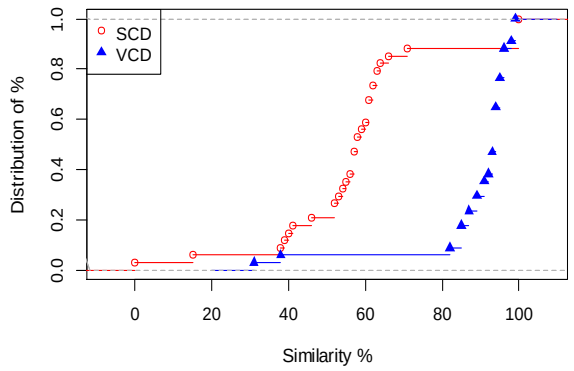

Figure 5: Cumulative distribution of similarity percentage of shingles and vector model for pdf file instances

After finding the similarities, the next step is to offload files with less similarity score to archive. Here though version 5 shows more similarity, the application doesn't unreasonably send the remaining versions to the archive; a match is done with the last date of modification. Files having fewer scores and older modifications are moved to the archive.

SCD and VCD give acceptable similarity scores when there is a similar content or slight variation in the versions. However, SCD gives the exact scores; when there is little or no similarity as illustrated in Figure 6.

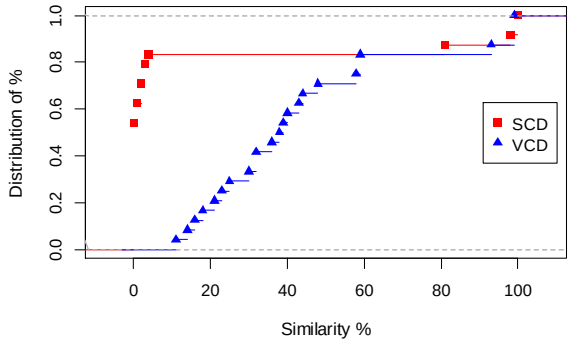

Figure 6: Cumulative distribution of similarity percentage of shingles and vector model for no similar and exact similar files

The Figure shows the distribution from similarity percentage with no similarity to exact similarity. For the files where SCD scores are integrated between 0 and 3%; VCD scores are integrated between 50 to 60%. After determining the scores on our dataset, we come to a conclusion that SCD is more powerful as it can work on short-distance scores.

We also observed the effect of the shingle length in finding similarity scores. It follows a tradeoff between the time taken for finding the similarity and percentage of similarity score. It is noticed that the performance of finding the similarity between different versions of the same file degrades with the increase in comparison window as shown Figure 7. Comparison window is measured in terms of shingle length and their hash values; smaller shingle length is better for the similarity detection performance.

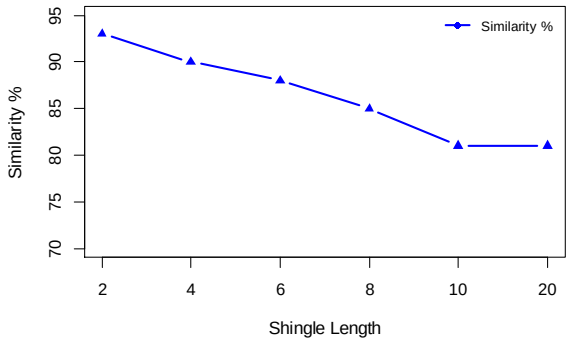

Figure 7: Performance of similarity percentage on shingle length

After finding the similarity scores and offloading the older versions to archive, the duplicate removal rate i.e. rate of unique versus similar data was calculated and it was seen that approximately 40% space was cleansed from the primary storage space.

CONCLUSION

This paper expresses an iterative process of duplicate removal; where we iterate on first detecting the unwanted files based on the file versions, similarity and the access time followed by eliminating them. We have used a shipper to transfer files from primary location to the archives instead of eliminating them permanently. Files are permanently deleted based on the number of probable file access in a given time interval. There are many alternatives for finding the similarity and duplicates. However, we have used the concept of shingles and cosine similarity distance and the reported work has demonstrated to diminish the storage space by 40% and the data offloading to archives results in the liberating more space for new data. We wish to address and extend this work on datasets versioning and removing the existing and un-accessed duplicates.

ACKNOWLEDGEMENTS

We acknowledge the support of everyone involved in the work for their help. The authors extend a special mention to all online resources for being a great help.

REFERENCES

[1] David Reinsel John Gantz John Rydning, “Data Age 2025: The Evolution of Data to Life-Critical”, IDC, 2017. Available at: http://www.seagate.com/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf.

[2] Nagapramod Mandagere, Pin Zhou, Mark A Smith, Sandeep Uttamchandani, “Demystifying data de-duplication”, Proceedings of the ACM/IFIP/USENIX Middleware '08 `.

[3] Koller, Ricardo, and Raju Rangaswami. "I/O deduplication: Utilizing content similarity to improve I/O performance." ACM Transactions on Storage (TOS) 6.3 (2010): 13.

[4] Xin Du, Weizheng Hu, Qiang Wang, and Fang Wang. "ProSy: A similarity based inline deduplication system for primary storage." In Networking, Architecture, and Storage (NAS), 2015 IEEE International Conference on, pp. 195-204. IEEE, 2015

[5] Yao, Wenbin, and Pengdi Ye. "Simdedup: A New Deduplication Scheme Based on Simhash." In International Conference on Web-Age Information Management, pp. 79-88. Springer Berlin Heidelberg, 2013

[6] Wen Xia, Hong Jiang, Dan Feng and Lei Tian. "DARE: A deduplication-aware resemblance detection and elimination scheme for data reduction with low overheads." IEEE Transactions on Computers 65.6 (2016): 1692-1705.

[7] Wen Xia, Hong Jiang, Dan Feng and Yu Hua. "SiLo: A Similarity-Locality based Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput." USENIX Annual Technical Conference. 2011.

[8] Philip Shilane, Grant Wallace, Mark Huang et al. "Delta Compressed and Deduplicated Storage Using Stream-Informed Locality." HotStorage. 2012.

[9] Frane Saric, Goran Glavas, Mladen Karan, et al. "Takelab: Systems for measuring semantic text similarity." Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation. Association for Computational Linguistics, 2012.

[10] Alina Lazar, Sarah Ritchey, and Bonita Sharif. "Improving the accuracy of duplicate bug report detection using textual similarity measures." Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, 2014.

[11] Himabindu Pucha, David G. Andersen, Michael Kaminsky. "Exploiting Similarity for Multi-Source Downloads Using File Handprints." NSDI. 2007.

[12] Tarasov Vasily, Amar Mudrankit, Will Buik, Philip Shilane, Geoff Kuenning, and Erez Zadok. “Generating realistic datasets for deduplication analysis.” In Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12), pp. 261-272. 2012.

[13] RFC 3174 - US Secure Hash Algorithm 1, September 2001. Available:https://tools.ietf.org/html/rfc3174

[14] Microsoft document item; Available at http://www.ecma-international.org/publications/standards/Ecma-376.htm

[15] Chou, Shih-Chun, et al. "Multi-version checkpointing for flash file systems." Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific. IEEE, 2016.

[16] http://qwone.com/~jason/20Newsgroups/

[17] https://www.philadelphiafed.org/research-and-data/real-time-center/greenbook-data/pdf-data-set