Hindawi
Scientific Programming
Volume 2022, Article ID 9515181, 17 pages
https://doi.org/10.1155/2022/9515181

Research Article

Content Deduplication with Granularity Tweak Based on Base and Deviation for Large Text Dataset

S. Venkatesh Babu,[1] **P. Ramya**,[2] **and Jeffin Gracewell**[3]

[1]*Department of Computer Science and Engineering, Christian College of Engineering and Technology, Oddanchatram, India*
[2]*Department of Artificial Intelligence and Data Science, PSNA College of Engineering and Technology, Dindigul, India*
[3]*Department of Electronics and Communication, Saveetha Engineering College, Chennai, India*

Correspondence should be addressed to S. Venkatesh Babu; venkateshflower6@gmail.com

Received 22 July 2022; Revised 5 October 2022; Accepted 27 October 2022; Published 22 November 2022

Academic Editor: Zhiri Tang

The concept of storage optimization has evolved as one of the hottest research projects in big data which brings out better solutions such as data compression which almost converges towards the deduplication technique. Deduplication is a technique that finds and eliminates duplicate content by storing only the unique copies of data whose efficiency is being qualified based on the amount of duplicate content that they hideout from the data source. The deduplication technique is a well-established storage optimization technique, so in the due course of time, various tweaks have been provided for its betterment, but it quite has some limitations that it cannot determine the tiny changes that occur among similar contents, and the chunks which are generated by segmenting and hashing the data are more sensitive to changes which produce a new chunk for every small change which ruins the concept of storage optimization, so to tackle this, content deduplication with granularity tweak (CDGT) in the Hadoop architecture has been proposed for large text datasets. The CDGT aims to improve the efficiency of deduplication by utilizing the Reed Solomon technique. This pumps out more duplicate content by verifying both intracontent and intercontent as consequence performance enhancements are met, and this system incorporates cluster-based indexing to reduce the time involved in data management activities.

1. Introduction

The cloud is a wonderful option to store data from pervasive entities, which attracts the digital world toward itself, and as a result it becomes the universe for data and services. The flexibility provided by the cloud, like the accessibility of services and data by user from anywhere which attracted huge number of user towards cloud as a result of it. Large amount of digital content of zettabytes in size is being dumped into the cloud environment and to realize the scenario the amount of data stored in the cloud has neared one hundred zettabytes at the end of 2020 [1]. On another side, the internet-enabled physical devices are called as cyber-physical systems (CPS) which include simple embedded to complex healthcare devices, and they are growing in large number [2] by producing large amount of data. The pervasive nature of CPS facilities is the utilization of the existing cloud infrastructure for data storage and processing, with billions of devices working around the clock all over the world producing a rain of data with similar contents, and as a result the scarcity of storage space increases [3]. As almost 60% to 70% of data stored in the cloud [4] are accountable as duplicates, this invites the necessity of deduplication techniques.

Data deduplication is a technique that eliminates redundant or duplicated and stores only the unique copy of data, which considerably reduces the storage space [5]. It is performed in two ways such as file-level deduplication and subfile-level deduplication. The file-level deduplication is performed on the file, which eliminates redundancy by comparation of files and saves only one instance of the file if they are identical [6]. The hash value which is computed for

each file is used for comparison, which is sensitive to change, so two files with little change will produce different hash values, so these two files are considered different, so both of them get stored as a result and storage optimization is not achievable. This paved the way for a new concept called subfile-level or block-level deduplication.

In subfile-level deduplication, the files are divided into small chunks of fixed or variable in size. In the fixed-size chunking algorithm, the files are divided into fixed-size chunks based on the predefined offset value, and in a variable-chunking algorithm, an offset value is calculated by using various techniques based on the context of the file [7]. Thus, the deduplication technique mainly operates on files with similar contents among themselves or in other correlated files. When the similarity concentration increases, the deduplication efficiency drops; that is, it fails to figure out the small changes which occur at the chunk level. This enlightens the concept of content deduplication with granularity tweak based on base and deviation which is the main core of this paper, and the main contributions of this paper are as follows:

(i) A deduplication technique is proposed that works in conjunction with generalized and classical deduplication

(ii) The utilization of versioning concept, a feature implemented by utilizing a machine learning technique which improvises the deduplication efficiency

(iii) The base deviation which is produced by the generalized deduplication technique acts as the tweak

(iv) The grouping of index-centric data in a cluster will decrease the processing time

This paper is organized first with a detailed literature review, followed by the proposed system, and finally, a comparative analysis to justify the findings is presented.

2. Background

The modern digital ecosystem is shifted to a distributed environment where the demand for greater computational and storage requirements is satisfied. This shift also brings changes in the organization of data; that is, they are allowed to be stored in a distributed fashion. In such an environment, different text-based data-producing nodes produce data of different types with less and more similar identical data, so it is a point where classical deduplication finds its phaseout because it is designed based on the context, and the change in the context will degrade their performance.

The deduplication technique is mostly preferred among other compression techniques since the former will not require decompression for reconstructing the data, so it dominates the deduplication ecosystem [8]. Classical deduplication works by comparing the hash of the chunks which are obtained by segmenting the file and followed by hash computation [9]. The chunks are the heart of deduplication, and their quantities play a crucial part in the performance in terms of space and operational complexity. In [10], a rule of thumb states that for 1 TB of data, there must be 5 GB of RAM for processing in the Zettabyte file system (ZFS), so generalized data deduplication (GDD) which is recently handled in [11] is capable of converging a large similar chunk to a common base, which brings down the memory consumption and improves the performance.

Generalized deduplication (GD) [12] fits under the lossless compression approach, which dynamically eliminates both similar and identical data. It works by splitting the data unit into base (B) and deviation (D) with the help of the transformation function. For example, consider a chunk C_i of a file X which is transformed to B_i, D_i and in which only one copy of B is stored in similar chunks with all its changes as deviation $\{D_1, D_2, \ldots, D_i\}$.

To be practical, considering a house located in the countryside, a time-lapse photograph of the house will have the house as static, with only the environmental conduction and movement of the object changing. When such a scenario comes to cloud, the GD transformation will separate each detail of the photo to the base and deviation, in which the house becomes the base since it is the static object and the other things become deviation (morning, evening, snow, with a parked car, without a parked car, etc.). GD will generate the base deviation pair, and it will store only the deviation for the similar base. The reconstruction of the image will be simple; that is, GD will determine the correct base (house) and then apply the exact deviation (environmental changes). The efficiency of matching depends on the transformation function, and the heart of deduplication lies in the indexing mechanism which will speed up search operation [13, 14] and improve if the entities in the index are organized based on the similarity with the help of fuzzy classifiers [15].

3. System Model

3.1. Overview of the System Model. The system is built in a modular fashion which includes summary generation, chunk generation, and content deduplication with granularity tweak (CDGT). The files from different time domains that come with similar text content have to be compared with duplicate content in a short duration, so the proposed system employs a summary generator for every new file that enters the system. The summary of a file can act as the metadata, which enhances the process of finding out similar contents to eliminate interfile duplication. Content deduplication with granularity tweak (CDGT) is designed to perform deduplication at the chunk level with the support of the base and deviation which is produced by Reed Solomon [16].

3.2. Summary Generator Based on NLP. The document in every storage environment has to be organized to facilitate faster retrieval based on their content of interest and has paved the way to incorporate natural language processing (NLP). The large content of the document is squashed to miniature which includes mostly the important signature of the document, such miniature content is called a summary

[17], and it comes under the extractive summarization technique [18].

The summary extraction is performed in a sequence of steps as shown in Figure 1; when a document arrives, it will be put into the document pool, from where it is fetched for processing.

The process begins with the data in the files being divided into sentences/words called tokens, and then the cleaning process takes care of eliminating the most commonly occurring words and special characters, called stop-words, because they do not provide useful information for the NPL to learn [9, 19]. Then, the term frequency (TF) is calculated, which is stated as the total number of times a word(t) appears in the sentence to that of the total number of words in the sentence of the file [20] as shown in equation.

$$\mathrm{TF}_{F\_ID} = \frac{\text{Number of occurrences of a word (t) in a sentence}}{\text{Total Number of wor ds in a sentence}}. \tag{1}$$

Once the TF is calculated to bring out the common words in the file, then the IDF is calculated to bring out more unique words in the documents using equation.

$$\mathrm{IDE}_{F\_ID} = \mathrm{Log}\left(\frac{\text{Total Number of sentences in a File}}{\text{Total Number of sentences with the word (t)}}\right). \tag{2}$$

The calculated values from equations (1) and (2) are used to assign weights to the words as shown in equation (3), where t is the word in the sentence.

$$\mathrm{Wor\,d\_W\,(t)} = TF_{F\_ID} * \mathrm{IDE}_{F\_ID}. \tag{3}$$

The word weight calculated in equation (3) for each word in the file is used to calculate the score values for each sentence in the file. Equation (4) describes the score calculation, where N is the number of words in a sentence.

$$SEN\_\mathrm{S_K} = \frac{\sum_{i=1}^{N} \mathrm{Wor\,d\_W}(ti)}{N}. \tag{4}$$

The average score value calculated for a file using equation (4) is used to set the threshold for extracting unique sentences from the file. The value calculated in equation (5) is normalized by multiplying with some constant value to extract the most unique sentences from the file, and it is set to 1.3 in this proposed system.

$$\mathrm{TH_{F\_ID}} = \frac{\sum_{K=1}^{N} \mathrm{SEN\_SK}}{N}. \tag{5}$$

The sentence with a score value higher than that of the threshold value is considered the more unique sentence in that file. In this manner, all unique sentences are extracted to form a summary for a file which forms the metadata. This metadata can act as the backbone for the processes of versioning.

3.3. Chunking Mechanism. Almost all deduplication techniques centre on chunking; it is the process that breaks the file into a small number of pieces called chunks, which may be fixed or variable in size. These chunks are subjected to hashing whose output is called a fingerprint which acts as a unique identifier for the chunk. When a new chunk arrives, it is compared with the stored identifier in the index table, and if both chunks are different, the new chunk will get stored, or else it is a duplicate entity, so only a new reference pointer is created for that chunk [21]. Figure 2 shows the working model of deduplication.

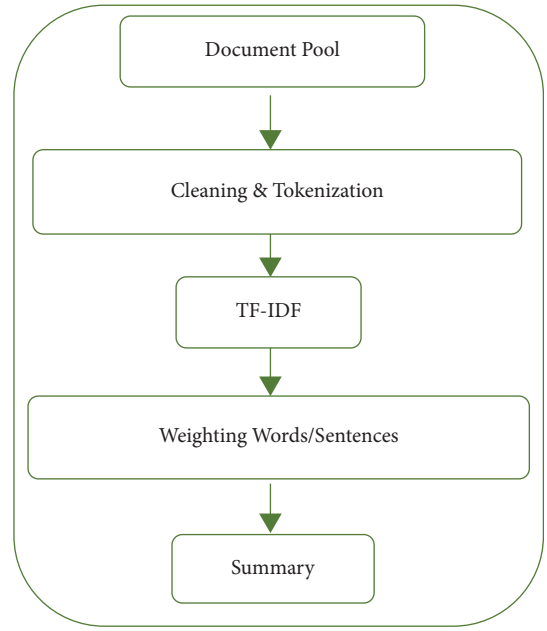

Figure 1: NPL processing.

In traditional compression techniques, the duplicates are eliminated over a small group of files or among contents in the same files, but in the deduplication technique, the redundancy in both interfiles and intrafiles over the largest data repository and even possible over multiple distributed storage servers are eliminated [22].

The variable size chunking method eliminates more redundant data such as the content defined chunking (CDC) method, which is more scalable and faster in processing. The CDC works by figuring out a set of locations to break the input data stream, and such breakpoints are called cut-points, and based on these points, the contents of the file are chunked. Many algorithms based on CDC will lead to poor performance when it produces chunks of smaller size, that is, below 8 KB [23]. In fixed-size chunking, all chunks will be of equal size, so the processing rate will be higher [24], but this method suffers from boundary shifting problems.

The boundary shifting problem will occur in a particular scenario when an already stored file arrives with some modification for backup, then while performing fixed-size chunking on the recently received file, the additional content which is being deleted/added/appended will relocate the information from one chunk to another chunk back and forth based on the nature of the modification. That is, the file with the inserted content will push the content from one chunk to another, and if the content is deleted from a file, the

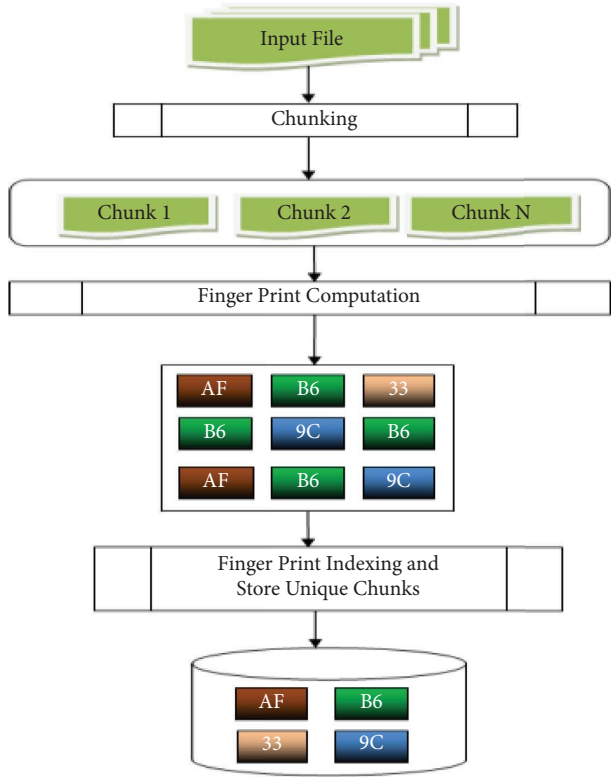

Figure 2: Deduplication.

chunks with the deleted part will bring up all information to fill up the desired chunk size from the chunks below. Thus, the chunks of the new file will pass the deduplication process, and this is not fair since the chunks with similar content with small deviated content will overload the storage space. In this paper, to tackle the problem, a hybrid method is utilized which is both dynamic and fixed in nature since the variable and fixed-size chunking provides the same result [25].

3.3.1. Cosine Similarity and Dynamic Content Adjustment Policy. The cosine similarity is used to measure the similarity between two documents of any size [26]. It measures the cosine of the angle between two given vectors in a multidimensional space; that is, the cosine of 00 is 1, and if the angle is between $(0, \pi)$, the value will be less than one. In another way, if the two terms are the same, then the cosine similarity is one; this value will reach 0 when the similarity between the texts reduces. The formula to calculate the cosine similarity is shown in equation.

$$\text{Cosine}(x, y) = \frac{x.y}{\|x\|\|y\|}, \tag{6}$$

where x and y are the two-term frequency vectors using which the angle between the vectors is calculated. When the system is unable to find a more similar file for a newly arrived file from the storage system by using a cosine similarity matching process, then in such a situation, the system uses perfect square divisor (PSD) to define the size of a chunk. For example, we consider a file size of 729 MB, and the perfect square of 729 is 27 MB, so the file is divided into chunks of the size of 27 MB each. However, in the case of prime numbers, there is no chance of getting perfect squares, and such a situation seeks the nearby square of a given number by using equation.

$$\text{Near\_Square}(\text{Prime}) = \text{floor}(\text{sqrt}(\text{Prime})) + 1. \tag{7}$$

The DCAP is activated whenever a file arrives in the storage system, and if it is a new file, then there will not be any copy of it which can be figured out by computing the cosine similarity among the available summaries. If the cosine similarity value between the newly arrived and already stored file is less than 1, then the stored file chunks will be fetched for deduplication. If the cosine similarity is 1, both files are the same and only pointers are adjusted.

The Dynamic Content Adjustment Policy (DCAP) can be explained by considering two sets of scenarios. The first scenario details the management of the boundary shifting problem for a file that has been modified by the insertion of new content, and the second scenario portrays the boundary shifting problem for a file with deleted contents.

(1) Dynamic Content Adjustment Policy (DCAP) Working Model. When a file named as File Version (FV2) has to be stored, it will trigger the cosine similarity function and fetch the File Version (FV1) without altering the chunk structure. The system will perform DCAP with the help of FV1 and FV2, which is explained with an example. Figure 3(a) shows a file FV1 with 30 characters and converted to hexadecimal values, then divided into equal-sized chunks. The chunks are named FV1C1 to FV1C5.

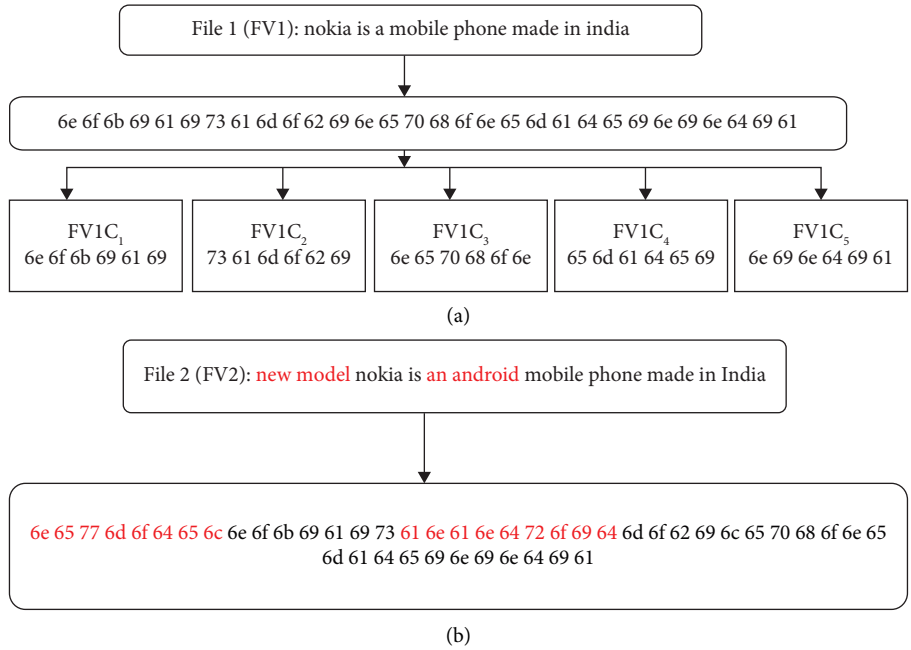

Figure 3: (a) Stored version and (b) modified version.

Figure 3(b) shows the new version of a file that is being generated by a typical user. The newly inserted characters and their hexadecimal values are marked in red colour. The objective of DCAP is to divide the content of FV2 into an equal number of chunks by identifying the newly modified contents. The content of FV2 is compared with the contents of FV1 by using a sliding window-like mechanism in which the content of FV2 is logically divided based on the window (W) size which is equal to the size of the chunk of FV1, and inside the W, two small seek windows (SW) are available; this SW is capable of moving across the windows. The SW located at the starting of the W is called the beginning seek window (BSW), and the one at the end is called the end seek window (ESW). The SW plays an important role in the DCAP, their size is fixed dynamically based on the W size or the number of characters in the chunks, and mostly, it is set to 64 bytes for larger files; in this example, the number of characters inside the W is 6, so the BSW and ESW are set to 2. Figure 4 shows the two files with the window and its corresponding seek window.

The file FV1 is taken as chunks, and the file FV2 is kept as a full file in which a window of size 6 will roll over the contents of file FV2. The DCAP algorithm starts by examining the file from the start. The characters inside the BSW and ESW in both files are compared, and if they are the same, then the characters in the current window over FV2 are grouped to form a chunk. In this example, as shown in Figure 4, due to the insertion of contents in FV2, the BSW and ESW have different characters when compared with FV1, which indicates that the file is modified.

Once the modification is figured out, the DCAP starts to compare the characters from the end of the file as shown in Figure 5. Here, the last chunk of the file FV1 is C5, so the operation starts from here, the character inside the BSW and ESW of FV2 is compared with that of FV1. In this case, the character is shown as follows:

(i) FV1: BSW {6e, 69}, ESW {69, 61}

(ii) FV2: BSW {6e, 69}, ESW {69, 61}

Both the SW of FV1 and FV2 have the same characters, and the characters under the current window over FV2 produce the same checksum as that of C5, so it will be considered a chunk that is similar to C5 of FV1. If the checksum is not the same, the window is considered a new chunk. Then, the window moves backward over the FV1 and FV2 to generate chunks.

Finally, the window reaches the point where the modification has happened, which is shown in Figure 6. The SW has a set of characters that indicate the occurrence of modification as shown in the following points:

(i) FV1: BSW {73, 61}, ESW {62, 69}

(ii) FV2: BSW {69, 64}, ESW {62, 69}

The characters under BSW of FV1 and FV2 have a different value, which indicates that the modification has happened in the previous chunk, which may push the characters towards the lower chunks, or the modification has happened at the same location.

In Figure 6, the characters under ESW of FV1 have {62, 69} which have the matching value at the same location in ESW of FV2, and the chunks C3, C4, and C5 of FV1 are similar to that of chunks in FV2 indicating that the modification has happened towards the beginning of the file.

The modification at the beginning of the file has pushed the characters forward, so the character {61, 69} of ESW of FV1 is missing in ESW of FV2 since it has moved in the forward direction. This states that the characters up to {61,

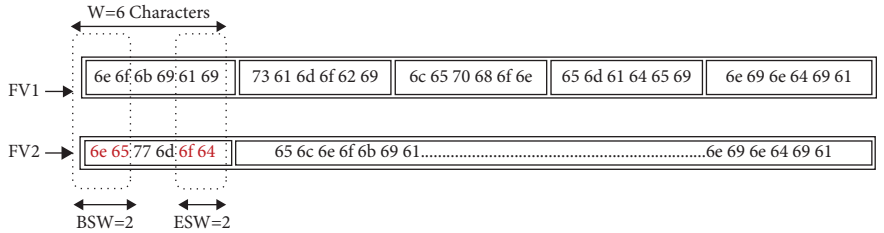

Figure 4: FV1 and FV2 with different contents.

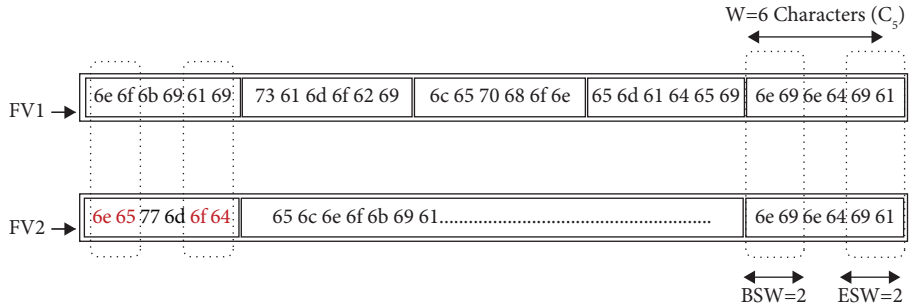

Figure 5: The comparison starts from the end of the file.

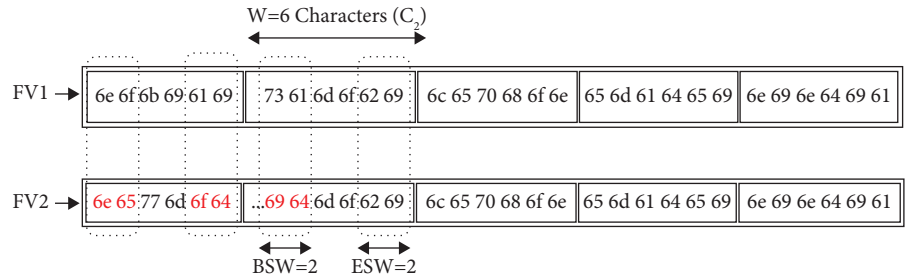

Figure 6: The window reaches the point of modification.

69} are being added and it has to be brought back to the same chunk so that it will eliminate the boundary shifting issues.

The BSW of FV2 has a set of values {6e, 65}, and {69, 64} has mismatched values at the same location in FV1 due to the movement of characters created by the insertion of characters. The BSW of FV1 has the following values: {6e, 6f} and {73, 61}; both do not have a match with FV2, and the first set {6e, 6f} is at the beginning, so there is no need to consider it. The second set of values {73, 61} is missing in BSW of FV2 at the same location, which states that the values up to {73, 61} have been displaced, so the goal is to search all values up to {73, 61} and put it back in the same chunk whose ESW value is {62, 69}.

In Figure 7 shows search operation, the ESW searches for the set of characters {61, 69}, and the BSW searches for {73, 61} in which the ESW moves in a forward direction and BSW in a backward direction. On the successful search, all bits up to {61, 69} are grouped as C1, and all bits up to {73, 61} are taken as C2 of FV2 and are shown in Figure 8 provided that

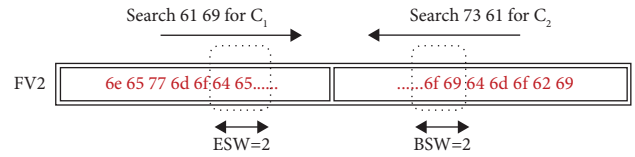

Figure 7: Search operation.

the newly formed chunks C1, C2 of FV2 should be less than or equal to 10 KB in size. C1 and C2 are the modified chunks that will have the newly inserted contents. The system will only consider the C1 and C2 of the file FV2 for performing GD and storing.

(2) Implementation of DCAP for Deleted File Version. When a node wants to upload a file to the storage system, we will first look for a similar version of the file in the storage system. If the system finds a file with similar content to the file that the user wants to upload, it will be sent to the user in

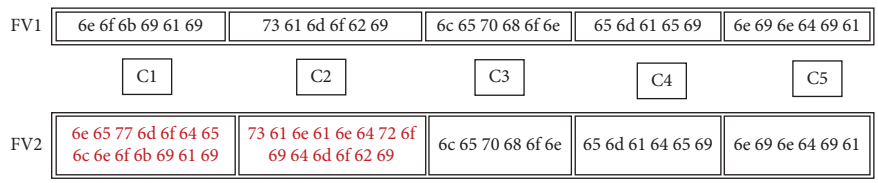

Figure 8: The final chunk of the file FV2.

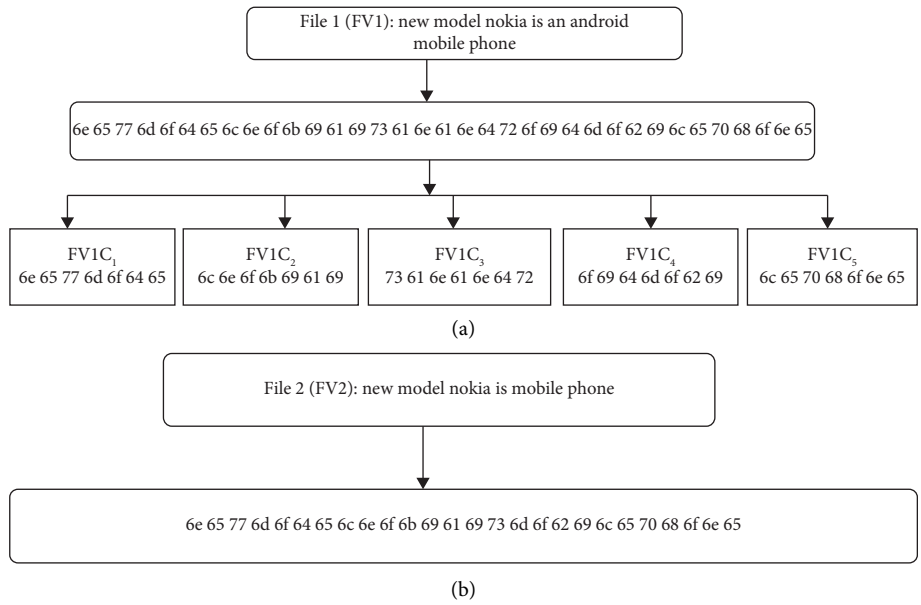

Figure 9: Different file versions (a) old version file FV1 and (b) new version file FV2.

chunks for verification. Figure 9(a) shows a file version that is already available in the storage system as chunks. The file is named FV1, and the chunks are named FV1C1 to FV1C5.

The DCAP is associated with window (W) which will roll over the files, and the beginning seek window (BSW) and the end seek window (ESW) are smaller windows that will perform the matching function by seeking. The chunk of FV1 has 7 characters, so the window size is set to 7, and the SW windows are set to 2 as shown in Figure 10. The DCAP tries to find the deleted portion in FV2 to form a chunk by retaining the remaining portion as the same as that of FV1; the working mechanism is shown as follows:

(i) The DCAP starts from the beginning of the file as shown in Figure 10. In this, the rolling window covers 7 characters of FV2, which is compared with the first chunk FV1C1 of FV1.

(ii) Inside the window, the BSW and ESW of FV2 and FV1 have the same set of characters as shown in the following points:

(a) FV1: BSW {6e, 65}, ESW {64, 65}
(b) FV2: BSW {6e, 65}, ESW {64, 65}

Since both files have similar content inside SW, and their checksum values are compared, and if it is found to be the same, the content under the window over FV2 is converted to a chunk with 7 characters which is similar to FV1C1. If they do not produce the same checksum, then the content under the window over the FV2 is taken as a new chunk.

Then, the window rolls over to the next set of 7 characters until the dissimilarity is found. The window is a set of 7 characters in FV2 which come under FV1C3 as shown in Figure 11, in which the content under SW is not similar as shown in the following points:

(i) FV1: BSW {73, 61}, ESW {64, 72}

(ii) FV2: BSW {73, 6d}, ESW {6c, 65}

The different content inside the BSW and ESW of both files FV1 and FV2 indicates the modification, so the DCAP will stop here and start the operation from the end of the file as shown in Figure 12. The end of the file is also modified since they have different values inside the SWs as shown in the following points:

(i) FV1: BSW {6c, 65}, ESW {6e, 65}

(ii) FV2: BSW {70, 68}, ESW {6e, 65}

The file FV2 is modified by deleting some content, and it happens in the middle, so to fill the deleted space, all content moves backward, and this states that the content under BSW {6c, 65} has moved backward in FV2. To determine {6c, 65}, the BSW of FV2 performs a search operation by moving backward as shown in Figure 13.

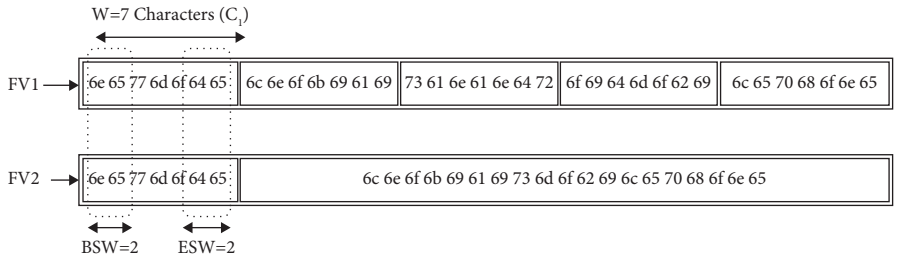

Figure 10: DCAP operation.

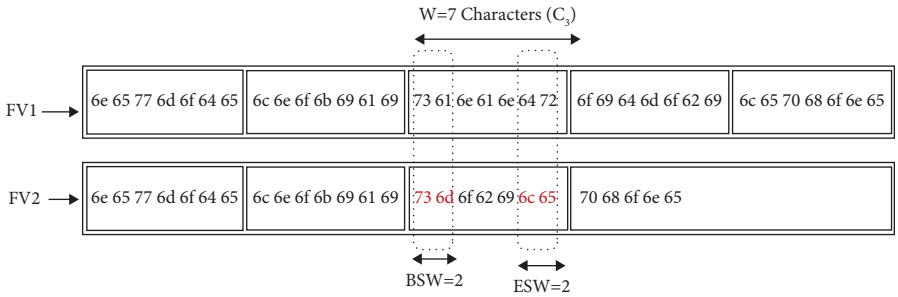

Figure 11: Modification point.

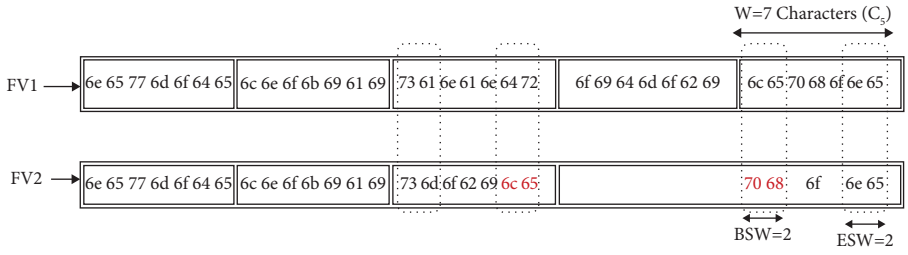

Figure 12: DCAP operation at the end of the file.

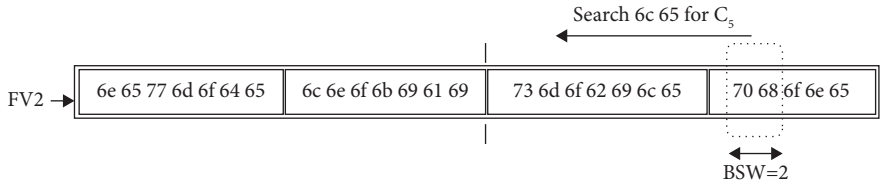

Figure 13: Search operation.

In Figure 14(a), the BSW of FV2 finds out {6c, 65}, and this is combined with the current window to form a chunk. The remaining characters of a new chunk are as shown in Figure 14, and the newly formed chunks should be less than or equal to 10 KB in size.

The search for the missing content in FV2 by BSW will move backward until it finds a matching value corresponding to the value of BSW of FV1. We need to stop once it finds a correct value. In some cases, if the character under consideration is deleted, the search operation fails. In such a situation, the DCAP will try to find a possible way to form a chunk similar to that of its older version as detailed next; for example, we consider two versions of files, the old version file (OVF) and the new version file (NVF). The OVF has 6 chunks. The first 2 chunks are matched with those of NVF, the 3rd chunk contains the modification, and then DCAP will start to perform its operation from the end of the file. The last window is unable to produce a chunk due to different values under BSW or ESW of NVF and OVF. In this situation, the BSW of NVF will try to find the value under

BSW of OVF by moving backward, and if the search fails, then BSW will try to find the value under BSW of OVF at C5; if the search is successful, then the content between the search matching position and the starting of C6 will be considered a separate chunk. If the entire search operation fails, the remaining characters are taken as separate chunks.

3.4. Fuzzy C-Means Cluster Assist Indexing. The performance of information retrieval is a more important feature that directly affects the deduplication throughput, so it has to be addressed with extra care. The deduplication process is designed to handle the text data, which is called documents or files which can be grouped based on the similarity that exists among them. The grouping of documents is termed clustering, and it is performed based on the relationship that preserves among the documents. The relationship can be drawn based on the terms and strong semantic connection among the terms using LSI [27], and fuzzy C-means clustering can be applied to group similar documents.

The mechanism behind document clustering is based on the term frequency extracted from all available documents and groups the documents which have more similar terms into one cluster. The texts in the document are converted into vector space, which grows along with the number of documents. The large size vector will become more complex while performing clustering, and the larger search space leads to inaccurate matching at the term level. Inaccurate matching will lead to the clustering of irrelevant documents. The application of latent semantic indexing (LSI) to the document vector will reduce the space, and more semantically related documents are grouped [28]. Therefore, clustering is performed after preprocessing the documents based on LSI.

3.4.1. Latent Semantic Indexing. The documents are associated with the summary which is being extracted based on the unique sentences available in it, and this summary is utilized for further processing. The first step is the tokenization process in which the files are divided into sentences called tokens by removing the stopping words to extract more unique words, and the collection of tokens is called a bag of words. Then, the term frequency (TF) is calculated based on equation (1), which brings out only the most common words from a particular summary. The second step is to create an IDF that contains only the more unique terms among all available summaries of "n" documents, which is computed based on equation (2). Then, the values of TF and IDF are used to generate the vector space based on equation (3), and this vector space is called a document vector.

The document vector space is very large, and it keeps on increasing each time when a new document is added to the system. Singular value decomposition (SVD) is used to reduce the document vector space and increase document retrieval accuracy. SVD is used as a tool for LSI. The normal document vector mostly finds it difficult to bring out the hidden concepts while applying some algorithms to retrieve documents. Let us consider that there are three terms {$t1$, $t2$, and $t3$} and {D1, D2, and D4} documents in which the term $t2$ appears in D1 and D4, so these two documents are logically related but not semantically connected. In this case, if a retrieval query Q arrives with a term $t2$ in it, then the matching document D1, D4 is given as matching, but D1 is the only document that semantically matches with the query Q. Thus, the LSI will map all terms and documents to a semantically connected vector space. The arrival of a new document induces the changes that have to be made to the existing vector space without recalculating SVD using a concept called folding-in; for more details refer to [29]. LSI is beyond the scope of this paper, so the reader may refer [44] for more information.

(1) Singular Value Decomposition for Latent Semantic Indexing. SVD is a factorization technique performed on matrix values to group documents into a defined number of topics. SVD is provided with the document matrix A of size $m \times n$ and along with the number of topics C. The dimensions m and n correspond to terms and document collection based on the number of summary provided at the initial stage. SVD produces three sets of matrices as shown in the following equation:

$$A = U\Sigma V^T, \tag{8}$$

where U is the eigenvector matrix of A^TA, V is the eigenvector matrix of AAT, and Σ is the diagonal matrix of singular values obtained by the square root of the eigenvalues of A^TA. Now, Σ constitutes too small singular values that are negligible since all values along the diagonal of Σ are listed in ascending order and it keeps only the top r values, which reduces the matrix of U, Σ, and V^T with U_r, Σ_r, and ${V_r}^T$. That is, it keeps the first r rows of U and r column of V^T. This process is named truncated SVD. The approximated matrix Ar is shown in the equation as follows:

$$Ar = U_r\Sigma_r V_r^T. \tag{9}$$

The following example is used to demonstrate SVD as shown in the following table. Table 1 contains nine documents and their set of bags of words which are tokenized words. This bag of words is used to calculate TF-IDF, and the resultant matrix is called a document vector.

Table 2 shows the ranking of documents; that is, {D1, D2, and D7} are having maximum scores in topic 1, and {D3, D4, D8, and D9} and {D5, D6} are in topics 2 and 3, respectively, based on the value they produce. Table 3 shows the new document D10 named pseudo document (PD), and its terms {John, gold, Juliet} are matched with the {T9, T10, T11} terms already available in the term vector space. Then, vector space for PD is calculated by taking the average of weighted terms in {T9, T10, and T11} as shown in Table 3, and from the result {0.05, 0.22, and 0.16}, it is clear that this document PD has a maximum score for Topic 2.

3.4.2. Fuzzy C-Means Cluster. To group similar files in a metadata table, FCM clustering is used. The resultant matrix V of LSI is a representative of document collection, which is

Table 1: Documents and bag of words.

D. no	Documents
D1	John has some cats
D2	Cats eat a fish
D3	Shipment of gold damaged in a fire
D4	Shipment of gold arrived in a truck
D5	Romeo Juliet
D6	Romeo died by dagger
D7	I eat a fish
D8	All that glitters is not gold
D9	Money makes many

Bag of words {"arrived: T1," "cats: T2," "dagger: T3," "damaged: T4," "died: T5," "eat: T6," "fishT7," "glitters: T8," "gold: T9," "John: T10," "Juliet: T11," "makes: T12," "money: T13," "Romeo: T14," and "shipment: T15," "truck: T16"}.

Table 2: Document vector space (V).

Document vector space (V)	Topic_1	Topic_2	Topic_3
D1	0.40	0.00	0.00
D2	0.97	0.00	0.00
D3	0.00	0.81	0.00
D4	0.00	0.78	0.00
D5	0.00	0.00	0.82
D6	0.00	0.00	0.82
D7	0.89	0.00	0.00
D8	0.00	0.63	0.00
D9	−4.46139E−17	7.64073E−16	4.36267E−16

given as input (X, $\{x_1, x_2, \ldots, x_n\}$) for FCM. The main motive of the FCM algorithm is to minimize the objective function [30], and it is shown in equation as follows:

$$\text{Obj\_Fun}(\mu, \text{C}) = \sum_{i=1}^{n} \sum_{j=1}^{c} \left(\mu_{ij}^{m}\right) \left\| x_i - c_j \right\|^2, \tag{10}$$

where $\|x_i - c_j\|$ is the Euclidean distance between the ith data point and the jth cluster, m is the fuzzifier whose value is set to 2, n is the number of data points in X, K is the number of clusters, and μ_{ij} is the membership degree of the ith data point in the jth cluster.

The steps of the FCM algorithm are as follows:

(1) Initialize the parameters such as number of clusters (k), initial membership matrix ($\mu^{(0)}$), cluster centre $C = \{c_1, c_2, \ldots, c_k\}$, termination criteria ($\beta$), and maximum iteration (Max_Iter).

(2) Calculate the cluster centre by using the equation as follows:

$$cj = \frac{\sum_{\mathbf{i=1}}^{\mathbf{n}} \left(\mu_{ij}^{m}\right) x_i}{\sum_{\mathbf{i=1}}^{\mathbf{n}} \left(\mu_{ij}^{m}\right)}. \tag{11}$$

(3) Calculate the Euclidean distance between the data point (X) and the cluster centre (C) using the equation

$$\text{d}(\text{X}, \text{C}) = \sqrt{\left(x_1 - c_1\right)^2 + \left(x_2 - c_2\right)^2 + \ldots + \left(x_n - c_i\right)^2}. \tag{12}$$

(4) Then, update the membership matrix (μ_{ij}) using the equation

$$\mu_{ij} = \frac{1}{\sum_{k=1}^{c} d_{ij}/d_{ik}^{\,2/m-1}}. \tag{13}$$

(5) If $||\mu^{(k+1)} - \mu^{(k)}|| < \beta$, then STOP; otherwise go to Step 2.

Here, k is the iteration step, and β is set to 0.01.

Similarly, when there are no changes in the cluster center [31], the algorithm can be stopped. The specified number of clusters will contain their corresponding documents. The cluster formed and the associated documents are shownas follows:

(i) Cluster_0 = {D5, D6}

(ii) Cluster_1 = {D3, D4, D8, D9}

(iii) Cluster_2 = {D1, D2, D7}

3.4.3. Fingerprint Indexing. The DCAP algorithm produces chunks, and these chunks have to be stored in the data node. The deduplication techniques work by finding similar chunks by comparing them, and to speed up the operation, a fingerprint is computed for all chunks that are stored in the data node. The fingerprint is an important entity in this system generated by using the SHA-3 (256 Bit) algorithm, and for this computation, the Keccak approach is considered which is based on the construction of a sponge. The SHA-3 uses a sponge to absorb the data and squeeze out the output. The message is subjected to various transformations in which the first phase messages are divided into blocks, and they are XOR'ed to get a subset of states [7]. Then, the entire states are transformed using the permutation function. In the squeeze phase, the output is fetched from the same states which are obtained from the transformation operation. Once the fingerprint is calculated, they have to be properly indexed for faster processing, so fuzzy C-means cluster assist indexing (FCMI) is used.

FCMI is performed in two phases; in the first phase, the LSI-ranked document is clustered, and in the second phase, the fingerprint values associated with the chunks of files inside a similar cluster are indexed based on B-tree.

3.5. Granularity Tweak Based on Base and Deviation. The chunks which are produced by the DCAP algorithm are subject to hashing which acts as the first stage index which will become clear in the later stage. The raw chunks will be subjected to base and deviation separation. The base and deviation are the entities that can be reversed to produce the original chunk, and this can be utilized for storage space optimization beyond deduplication. In normal hash-based deduplication, the more identical chunks are eliminated, but the less similar chunks find their way to the storage system, which leads to poor storage optimization. The storage optimization is still finely tuned with the help of base and deviation concepts, in which the less similar chunks are

Table 3: Pseudo document.

Pseudo document (PD)	T9	T10	T11	PD-vector space
John buys gold for Juliet	{0.00, 0.65, 0.00}	{0.16, 0.00, 0.00}	{0.00, 0.00, 0.47}	{(0.00 + 0.16 + 0.00)/3, (0.65 + 0.00 + 0.00)/3, (0.00 + 0.00 + 0.47)/3} = {0.05, 0.22, 0.16}

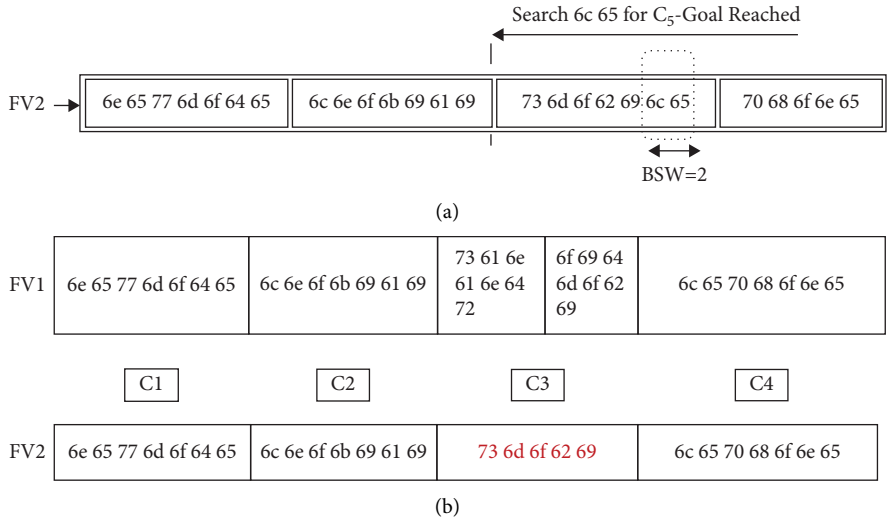

Figure 14: DCAP operation: (a) goal state and (b) chunking of FV2.

separated, which considerably reduces the storage requirement.

The idea is that each chunk is divided into a pair of deviation and base, in which the base contains the majority of information and the deviation comparatively contains small information. In case two, if chunks are similar, then it is enough to store a single base and their two different deviations; in this way, we can perform deduplication on the sorted base, which will considerably reduce the storage cost. The transformation of a chunk to the base is many-to-one mapping, the algorithm takes a chunk as input to produce the base and the deviation which shows the difference between chunks, and such kinds of transformations are performed by error-correcting algorithms.

3.5.1. Generalized Deduplication Algorithm. In this session, a generalized deduplication mechanism is presented, and its objective is to reduce the size of chunks. Let there be n chunks in the system set_$C = \{C_1, C_2, \ldots, C_n\}$ and let each chunk be of k bit length, so the chunks need the storage size as given in equation

$$\text{S\_Cost}(C) \triangleq nk \text{ bits}. \tag{14}$$

To picturize the case, we consider a typical system that produces 100 chunks at time t of size 64 bytes each, and then the storage space requirement will be $100 * 64 = 6400$ bytes; this can be further reduced with the help of GD. The GD system will divide the chunk into two parts: base (B) and deviation (D). Then, for some K numbers of B with qk bits, the cost for storing B with its identifier is shown in equation

$$S2\_B = qk + \lceil \log K \rceil. \tag{15}$$

Then, if there are N chunks that are associated with the $\lceil \log N \rceil$ bit for the identifier and also include the deviation, the storage required for each chunk is given in the equation

$$S3\_CBL = \lceil \log N \rceil + \lceil \log K \rceil + 1 \text{ bits}. \tag{16}$$

Then, the total cost required to store N chunks becomes

$$S\_T1 = N\,S3\_CBL + K\,S2\_B \text{ bits}. \tag{17}$$

Then, to make a comparison, the cost of storing N chunks without GD is shown in the equation

$$\text{S\_T2} = N(qn + \lceil \log N \rceil). \tag{18}$$

The compression factor is expressed in equation (19) which includes both the chunks and B, D, where $\text{CF} > 1$ states that the compression has occurred.

$$\text{CF} \triangleq \frac{\text{S\_T2}}{S\_\text{T1}}. \tag{19}$$

3.5.2. Deduplication System. Data deduplication is a technique that eliminates redundant or duplicated data and stores a unique copy of data. This technique considerably reduces storage [32]. Data deduplication is a need to be performed before inserting data into a data store. In this system, deduplication is performed based on the base and deviation of the chunks; before performing deduplication,

the system is preprocessed to enable smooth processing. The sequence starts with the chunking of files based on DCAP or user-defined chunking, which is followed by fingerprint calculation. Then, the base deviation of the chunk is calculated, and, on another side, the documents are clustered, and the fingerprints are indexed using the B-tree as shown in Figure 15. The files seeking storage space are collected from the file pool, and then these newly arrived files are subjected to summary extraction, which will help disclose their associated similarity with other files which are already finding their way in, and if this search is positive, then DCAP is performed which produces some sets of chunks, and on negative, the chunks are produced based on the user-defined chunking algorithm. (See Figure 16).

The base and deviation are derived from the chunk of the file which acts as the main base for the deduplication system. The two identical chunks produce the same base and deviation, but two less similar chunks produce the same base with different deviations, and the chunk can be reconstructed using the base and deviation. The deduplication system works in the following manner:

(1) The chunks of a particular file are subjected to fingerprint calculation.

(2) The base and deviation are computed.

(3) The file under consideration is searched for a suitable domain among the available clusters. On finding a suitable cluster, the fingerprint is searched for a match based on which the following activities are performed:

(a) If a fingerprint is matched, it indicates that there is a matching chunk, so it is not necessary to store the base and deviation of the currently working chunk and so only the reference pointer is adjusted; that is the currently working chunk is deduplicated

(b) If fingerprint matching failed, it indicates that the chunk is new, so a new index is created, and then the base pool is queried for matching

(i) If a base pool search is successful, it indicates that there is a chunk with similar data, so the current base is taken as a reference with a new deviation

(ii) If the base pool search is failed, then a new base and deviation are generated and they get stored in the system

3.5.3. Data Placement Policy. The CDGT is designed to work for text-based files, and it is well integrated with the existing Hadoop architecture. The base and deviation are the entities that are allowed to be stored and retrieved from the HDFS (Hadoop distributed file system), and the experimental setup consists of the name node (NN) and a set of data nodes (DN). The NN acts as the master, and DN are slaves used to store the data; in our case base and deviations, the fingerprint index is maintained in the MongoDB for faster processing. This system utilizes the in-line deduplication

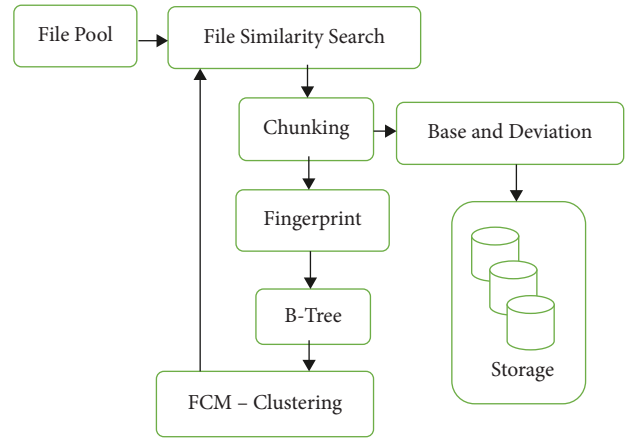

Figure 15: General system structure.

mechanism where the deduplication is performed before the data are written to the disk. The data flow is explained in .

(i) The base and deviation which are produced and finalized are the entity that has to be stored in DN

(ii) When any client uploads the file, the agent module in this proposed deduplication ecosystem will trigger the deduplication process as shown in Figure 17

(iii) The files which reach the agent module in the name node will get deduplicated

(iv) The final base and deviation that pop out of the deduplication system are stored in DN

(v) In case of the same base, the reference is adjusted and the corresponding deviations alone get stored in DN

(vi) The name node manages the storage of data, and the indexes are maintained in the MongoDB

4. Results and Discussion

4.1. Performance of Summary Generation. The proposed system involves a large amount of files to facilitate the processing of versioning, and this system uses summary extraction based on NLP, which is considered more advantageous than other non-NLP methods. The performance of the proposed summary extraction method is evaluated based on the parameters [33], such as precision (P), recall (R), and F-measure, which are mostly used in the field of information retrieval. Let the summary extracted by NLP be given by SNLP and the non-NLP is given by SnNLP, which are shown in the equations (20)–(22).

$$P = \frac{\left|S_{nNLP} \cap S_{NLP}\right|}{\left|S_{NLP}\right|}, \tag{20}$$

$$R = \frac{\left|S_{nNLP} \cap S_{NLP}\right|}{\left|S_{nNLP}\right|}, \tag{21}$$

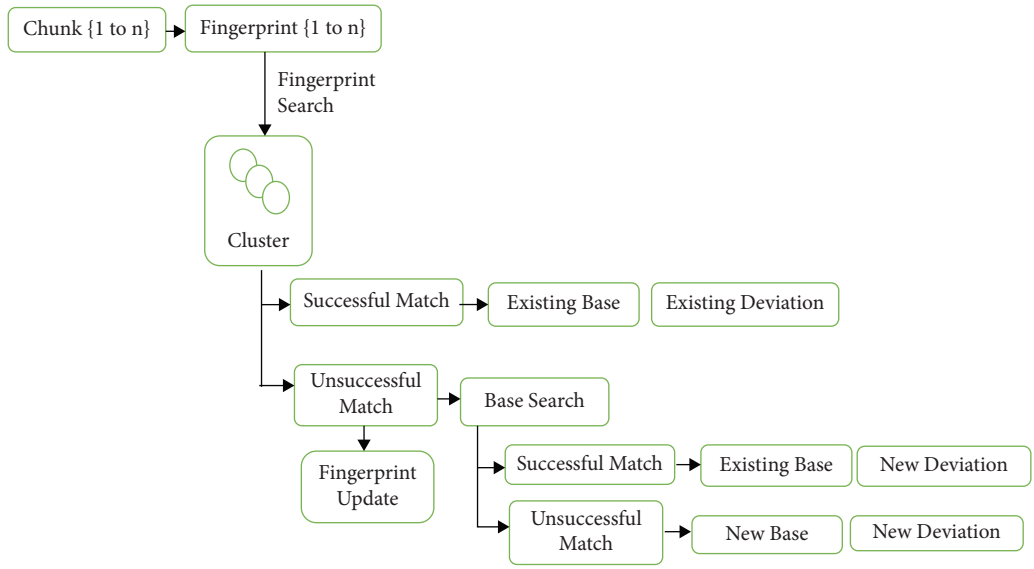

Figure 16: Deduplication system.

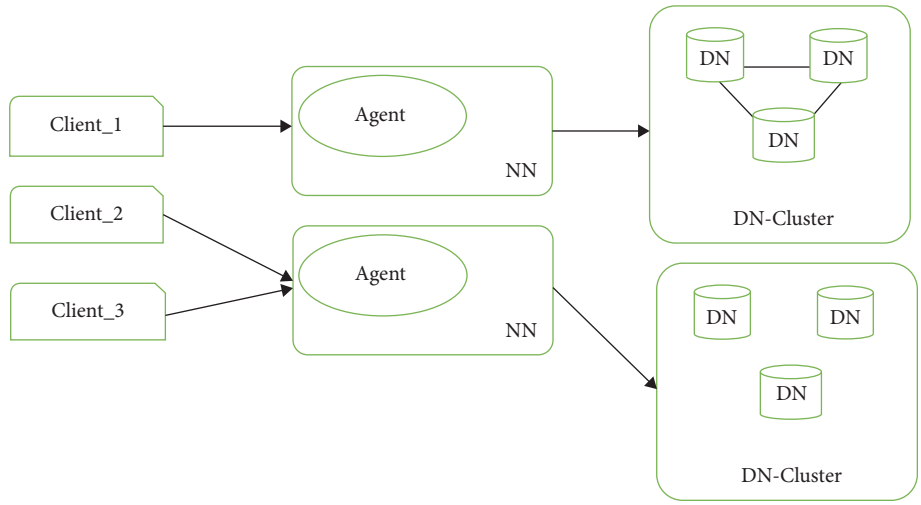

Figure 17: Hadoop architecture.

$$F = \frac{2PR}{P + R}. \tag{22}$$

The ROUGE method is also utilized to evaluate the performance, and this method measures the quality of the summary by counting the overlapping units such as N-gram, work pair, and sequences between the proposed and reference summary based on the DUC 2004 dataset [34]. The ROUGE-N is used to compare the N-grams between two summaries [35]. Table 4 shows the performance improvement by the NLP method.

4.2. Performance Evaluation of DCAP. To evaluate, the experiment is carried out using Pentium E5700 and Intel i5 9th generation processors since the system is built around a commodity system. The data nodes are virtually created in Hadoop and are allocated to disk space based on the data set which is being used to evaluate the parameters. The developed system is designed to work in an interactive environment, so the realistic data sets such as the backup of user text-based files from 30 users of Windows and Linux are collected at various time intervals such as 37 GB, 75 GB, and 98 GB. The proposed system uses the Dynamic Content Adjustment Policy (DCAP) whose performance can be compared based on the following parameters:

(i) Chunk overhead

(ii) Hash judgment time

The chunk acts as a processing unit in any typical deduplication system, so the number of chunks has a direct impact on the performance. The proposed DCAP algorithm produces a smaller number of chunks when compared with other methods. The DCAP uses the versioning concept which is the main responsibility for producing a lower number of chunks when the file size is 15 MB, and the DCAP produces 64%, 75% fewer chunks than that of fixed size

Table 4: NLP vs. non-NLP.

Methods	ROUGE-1	ROUGE-2	*F*-measure
Ranking [36]	0.38613	0.06918	0.42246
COSUM [37]	0.34124	0.09678	0.44365
NLP	0.43219	0.10642	0.46596

chunking (FSC) and content defined chunking (CDC), respectively. When FSC is compared with CDC, the FSC performs better by producing 39% lesser chunks than that CDC for a 23 MB file, but when compared with DCAP, the FSC outputs 74% more chunks for 23 MB of the file. The higher number of chunks produced by the CDC than that of FSC is because the CDC examines the content for breakpoints and based on it the files are chunked.

The FSC produces chunks of fixed size that are set to fine-tune the deduplication elimination ratio (DER), and if the chunk size is set to smaller KBs, then the chunk overhead increases, and the same phenomena are seen in CDC. Then, for 36 MB files, FSC produces 61% more chunks than that DCAP, and the same is seen for CDC with 75% higher value. The DCAP produces an average of 66%, 75% less overhead than that of FSC and CDC, respectively, as shown in Figure 18. The lesser chunk overhead of DCAP is due to the versioning of files, which helps to improve the processing efficiency of CDGT.

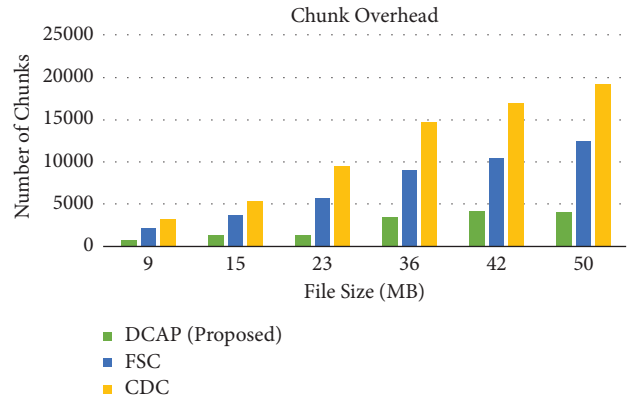

Figure 18: Chunk overhead.

4.2.1. Hash Judgment Time. The hash judgment time is defined as the time needed to find the duplicate entity for the newly calculated chunks in the already stored base by looking up in the index table which is built based on the hash values of the chunks and the base. The efficiency of hash judgment time depends on the nature of the index. The following are the evaluation parameters taken into consideration, and Table 5 shows the operating modes.

(i) Hash judgment time normal mode
(ii) Hash judgment time GD mode
(iii) Hash judgment time normal mode-FCMI
(iv) Hash judgment time GD mode-FCMI

The hash judgment time is measured for both methods that are normal deduplication without GD and deduplication based on GD. The hash judgment time in the normal mode (H-NoFCM-NM) consumes more time than that of the hash judgment time with FCM in the normal mode (H-FCM-NM) which is an average of 47.48 ms and is reduced by employing FCM in NM. The FCM indexes the entries, so the search time reduces considerably, which is seen in the lower hash judgment time in H-FCM-NM. The same kind of result is seen with deduplication based on GD; that is, hash judgment in the GD mode without FCM (H-NoFCM-GD) takes about an average of 20.66 ms more time than hash judgment in the GD mode with FCM (H-FCM-GD). The GD-based method takes comparatively lesser time than normal deduplication to determine the duplicate contents; that is, there is an average of 26.82 ms difference which is seen between them as shown in Figure 19. The average reduction in the hash judgment time in GD is due to GD which reduces the number of chunks with more fine granules such as the base and deviation.

Table 5: Mode of operation.

Mode of operation	Meaning
NM	Normal deduplication method
GD	Deduplication based on generalized deduplication

4.3. Performance Evaluation of GD. The deduplication elimination ratio (DER) is defined as the ratio between the file size before deduplication (FSBD) to that of file size after deduplication (FSAD) as shown in equation (23). The following points are the evaluation parameters taken into consideration:

(i) DER normal mode
(ii) DER GD mode
(iii) Saving percentage (SP)

$$\text{DER} = \frac{\text{File size Before Deduplication}}{\text{File size After Deduplication}}, \tag{23}$$

$$\text{SP} = 1 - \left(\frac{\text{Compressed data length}}{\text{Uncompressed data length}}\right) * 100. \tag{24}$$

Table 6 shows the DER comparison between deduplication with and without GD for some sets of files with varying file sizes. The system based on GD shows an improvement in DER of about 10.98%, which will help to reduce the storage space considerably.

The saving percentage (the amount of storage space occupied) has increased by an average value of 6.72% percentage when GD is employed in the system. The proposed system improvises storage utilization as shown in Figure 20.

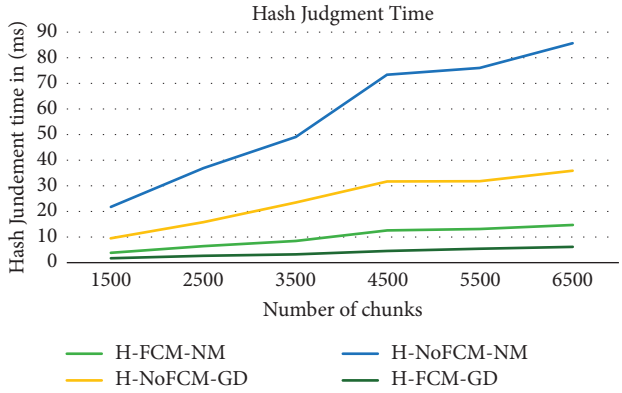

Figure 19: Hash judgment time.

Table 6: Deduplication elimination ratio.

Size (MB)	NM-DER	GD-DER
50	1.851851852	2.057613169
42	1.789518534	1.88781014
36	1.333333333	1.388888889
23	1.769230769	1.965811966
15	1.666666667	1.984126984
9	1.282051282	1.780626781
5	7.042253521	7.30994152
1	2.040816327	2.267573696

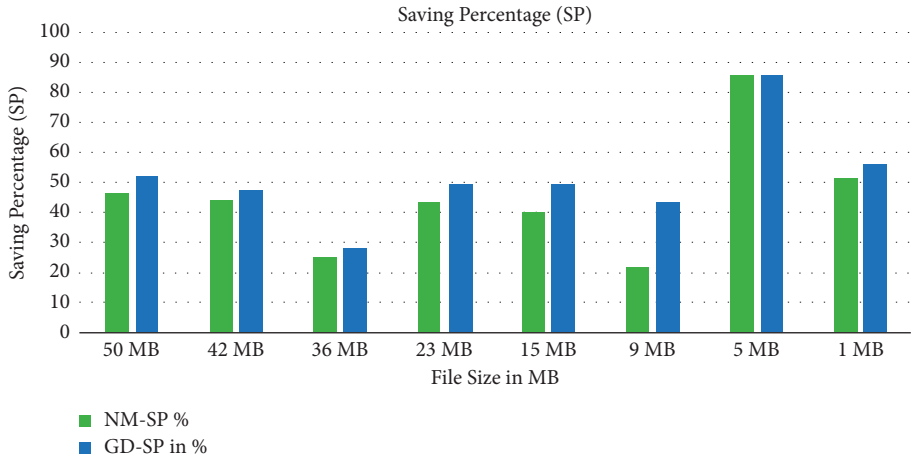

Figure 20: Saving percentage.

5. Conclusion and Future Enhancement

The content deduplication with granularity tweak based on base and deviation is a deduplication system that is designed to improve the deduplication efficiency in terms of deduplication elimination ratio (DER) and to optimize the storage utilization. The implemented system employs the Dynamic Content Adjustment Policy as a chunking method that performs chunking, and this method is assisted by summary extraction based on NLP. The utilization of the cosine similar mechanism improves the matching accuracy while performing versioning, which has put the proposed system in a promising position to achieve higher DER. The novel chunking mechanism DCAP is sharper in producing minimal chunks, which considerably brings down the overhead in view of computation complexity. The fingerprint of chunks and base deviation are organized as a cluster using FCM, and their entire ties are ordered, which brings

down the searching time and smoothly increases the accuracy rate irrespective of the size of the cluster and index and that there is an average of 68.5% efficiency which is recorded. The important performance parameter of any deduplication system is deduplication elimination ratio (DER), so this paper focuses on the improvement of DER. GD provided such a facility that fine-tunes the DER, and this can be seen in the result session; the GD-DER shows an improvement of about 10.98% over the deduplication system without GD. GD improves the storage utilization by 6.72% when compared with that of the system without GD.

Data Availability

The data used to support the findings of this study have not been made available because they contain more personal files (*The Digital Universe of Opportunities*. Rich data and the increasing value of the Internet of things and EMC Digital Universe with Research and Analysis by IDC, [Online]. Available: https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm).

.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] J. Gantz and G. Reinsel, "The Digital Universe in 2020: Big Data, Bigger Digital Shadows, Biggest Growth in the Far East," *IDC iView: IDC Analyse the Future*, vol. 2007, pp. 1–16, 2012.

[2] CISCO, *Fog Computing and the Internet of Things: Extend the Cloud to where the Things Are*, White Paper, San Jose, CA, USA, 2015.

[3] C. Stergiou, KE. Psannis, B.-G. Kim, and B. Gupta, "Secure integration of IoT and cloud computing," *Future Generation Computer Systems*, vol. 78, pp. 964–975, 2018.

[4] J. Stanek and L. Kencl, "Enhanced secure thresholded data deduplication scheme for cloud storage," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 694–707, 2018.

[5] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley-Interscience, Hoboken, NJ, USA, 2006.

[6] H. Hovhannisyan, W. Qi, K. Lu, R. Yang, and J. Wang, "Whispers in the cloud storage: a novel cross user deduplication-based covert channel design," *Peer-to-Peer Networking and Applications*, vol. 11, no. 2, pp. 277–286, 2016.

[7] C. P. Chen and C. Y. Zhang, "Data-intensive applications, challenges, techniques, and technologies: a survey on big data," *Information Science*, vol. 275, pp. 314–347, 2014.

[8] W. Xia, H. Jiang, D. Feng et al., "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[9] N. Mandagere, P. Zhou, M. A. Smith, and S. Uttamchandani, "Demystifying data deduplication," in *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion, ser. Companion '08*, pp. 12–17, ACM, New York, NY, USA, January 2008.

[10] C. Gonzalez, "ZFS: to dedupe or not to dedupe," 2011, https://constantin.glez.de/2011/07/27/zfs.to.dedupe.or.not.dedupe.

[11] R. Vestergaard, Q. Zhang, and D. E. Lucani, "Generalized deduplication: bounds, convergence, and asymptotic properties," in *Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–14, Waikoloa, HI, USA, December 2019.

[12] JK. Periasamy and B. Latha, "Secure and duplication detection in cloud using cryptographic hashing method," *International Journal of Engineering & Technology*, vol. 7, no. 1.7, pp. 105–108, 2018.

[13] T. C. Havens, J. C. Bezdek, C. Leckie, L. O. Hall, and M. M. Palaniswami, "Fuzzy c-means algorithms for very large data," *IEEE Transactions on Fuzzy Systems*, vol. 20, no. 6, pp. 1130–1146, 2012.

[14] I. Giangreco, A. I. Kabary, and H. Schuldt, "ADAM - A database and information retrieval system for big multimedia collections," in *Proceedings of the IEEE International Congress on Big Data*, pp. 406–413, Anchorage, AK, USA, July 2014.

[15] I. Wang, Y Tai, and M. Huei, "Fuzzy support vector machine for multi-class text categorization," *Information Processing & Management*, vol. 43, pp. 914–929, 2007.

[16] R. M. Alguliyev, R. M. Aliguliyev, N. R. Isazade, A. Abdi, and N. Idris, "COSUM: text summarization based on clustering and optimization," *Expert Systems*, vol. 36, Article ID e12340, 2019.

[17] S. Mohamed Saleem and M. Nikita, "Study on text summarization using extractive methods," *International Journal of Science, Engineering and Technology Research*, vol. 4, 2015.

[18] R. Khan, Y. Qian, and S. Naeem, "Extractive based text summarization using KMeans and TF-IDF," *International Journal of Information Engineering and Electronic Business*, vol. 11, no. 3, pp. 33–44, 2019.

[19] F. P. Martin, "An algorithm for suffix stripping," *Program: Electronic Library and information system*, vol. 40, pp. 130–137, 1980.

[20] S. Qaiser and R. Ali, "Text mining: use of TF-IDF to examine the relevance of words to documents," *International Journal of Computer Application*, vol. 181, no. 1, pp. 25–29, 2018.

[21] A. Venish and K. Siva Sankar, "Study of chunking algorithm in data Deduplication," in *Proceedings of the International Conference on Soft Computing Systems (ICSCS)*, vol. 2, pp. 13–20, 2016.

[22] W. Dong, D. Fred, L. Kai, H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters," in *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pp. 15–29, USENIX, Berkeley, CA, USA, 2011.

[23] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, pp. 174–187, New York, USA, July 2001.

[24] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *Journal of Computer and System Sciences*, vol. 79, no. 8, pp. 1230–1242, 2013.

[25] C. Cornel, G. Joseph, and D. Chambliss, "Mixing deduplication and compression on active data sets," in *Proceedings of the Data Compression Conference (DCC), IEEE Computer Society*, pp. 393–402, Washington, DC, USA, March, 2011.

[26] A. Huang, "Similarity measures for text document clustering," in *Proceedings of the Sixth New Zealand Computer Science Research Student Conference (NZCSRSC)*, vol. 4, pp. 49–56, Christchurch, New Zealand, January, 2008.

[27] M. E. S. Mendes Rodrigues and L. Sacks, "A scalable hierarchical fuzzy clustering algorithm for text mining," in

Proceedings of the 5th International Conference on Recent Advances in Soft Computing, January, 2004.

[28] K. Sathiyakumari, V. Preamsudha, and G. Manimekalai, "Unsupervised approach for document clustering using modified fuzzy C mean Algorithm," *International Journal of Clinical Oncology*, vol. 1, no. 3, pp. 12–17, 2011.

[29] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Incremental singular value decomposition algorithms for highly scalable recommender systems," in *Proceedings of the 5th International Conference in Computers and Information Technology*, Washington, DC, USA, September, 2002.

[30] A. N. Ghosh, N. S. Mishra, and S. Ghosh, "Fuzzy clustering algorithms for unsupervised change detection in remote sensing images," *Information Sciences*, vol. 181, no. 4, pp. 699–715, 2011.

[31] C. R. Ramesh, K. Rao, and G. Jena, "Fuzzy clustering algorithm efficient implementation using centre of centres," *International Journal of Intelligent Engineering and Systems*, vol. 11, no. 5, pp. 1–10, 2018.

[32] L. Wang, Z. Zhu, X. Zhang, X. Dong, and Y. Wang, "DOMe: a deduplication optimization method for the newsql database backups," *PLOS ONE Research Article*, vol. 12, no. 10, Article ID 01851899, 2017.

[33] T. Reddy, S. Chandra, and M. V. P. Rao, "Performance evaluation of various data deduplication schemes in cloud storage," *International Journal of Pure and Applied Mathematics*, vol. 116, no. 5, pp. 175–180, 2018.

[34] D. Shen, J. T. Sun, H. Li, Q. Yang, and Z. Chen, "Document summarization using conditional random fields," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pp. 2862–2867, Hyderabad, India, January, 2007.

[35] C. Y. Lin, "ROUGE: a package for automatic evaluation of summaries," in *Text Summarization Branches*, pp. 74–81, Association for Computational Linguistics, 2004.

[36] Y. Zhang, D. Li, Y. Wang, Y. Fang, and W. Xiao, "Abstract text summarization with a convolutional Seq2seq model," *Applied Sciences*, vol. 9, no. 8, p. 1665, 2019.

[37] J. N. Madhuri and R. Ganesh Kumar, "Extractive text summarization using sentence ranking," in *Proceedings of the 2019 International Conference on Data Science and Communication (IconDSC)*, pp. 1–3, Bangalore, India, March, 2019.