Ingénierie des Systèmes d'Information
Vol. 29, No. 6, December, 2024, pp. 2197-2207
Journal homepage: http://iieta.org/journals/isi

Data Deduplication and Integrity Check Scheme Based on Unique Tags in Edge Cloud Computing

Sevuga Pandian Asirvatham[1*], Gomathi Muthusamy[2]

[1] Department of Computer Science, Periyar University, Salem 636 011, Tamil Nadu, India
[2] Government Arts and Science College, Affiliated to Periyar University, Salem 636 011, Tamil Nadu, India

Corresponding Author Email: pandianasir@yahoo.com

https://doi.org/10.18280/isi.290610

Received: 2 March 2024
Revised: 29 September 2024
Accepted: 27 November 2024
Available online: 25 December 2024

Keywords:
cloud computing, edge computing, chunk, index, threshold, similarity measures, data de-duplication, integrity check, security

ABSTRACT

Edge computing is an extension of cloud computing that uses additional middleware, or edge nodes, located closer to clients to speed up request processing for the central cloud. It offers benefits such as reduced latency and bandwidth savings at the edge. Extensive research has focused on decryption technologies for encrypted data to enhance the security and functionality of traditional cloud systems while maintaining privacy. Secure encryption techniques on the server side protect data privacy but can hinder network performance and allow for multiple uploads. On the other hand, a client-side approach improves network performance but is vulnerable to data leakage. To overcome the problem, we proposed a new method based on Checksum Selective Indexing Chunking (CSIC) for improving cloud storage for deduplication files and an error checking that verifies the integrity of the deleted data. Initially, preprocessing can reduce the similarity of text and duplicates between files based on the Threshold Similarity Measures (TSM) and calculate the similarity scores. The second stage is segmentation, based on a Cluster Index-Based Sliding Window Hashing (CISWH) for segmenting the index content using the set. Each data set uses a Hash identifier for chunk contents. Then, the content integrity should be ranked based on the weightage of the file and reduced the empty storage in the cloud by processing a large number of edge data copies and finding multiple degraded edge data files using Prefetching Local Index Caching (PLIC). Then, the Unique Tag Deduplication Integrity Check (UTDIC) is used to download private information in the cloud. A protection analysis demonstrates the accuracy and robustness of the scheme. This method allows you to identify data redundancies at the block level and reduce data redundancies more effectively. The simulation results show that the proposed method is superior in terms of computational efficiency and security level. Thus, the proposed attains high results in terms of precision at 97.9%, recall at 98.8%, and storage optimization performance at 43.2%.

1. INTRODUCTION

Multi-user cloud storage solutions with shared networks are gaining popularity due to the constant increase in data creation on a global scale. However, many users still need to be more hesitant to move data to remote storage due to concerns about data security. Encrypting data before it leaves the owner's property is a standard procedure. While this method is beneficial for safety, it hinders storage efficiency features such as compression and Deduplication. These features allow storage providers to utilize their back-ends more effectively and serve more customers with the same infrastructure.

However, because they no longer own this data locally, service providers find it challenging to maintain complete control over their data housed on dispersed edge servers. In the highly distributed edge computing environment, edge servers should not be taken for granted since hardware and software exceptions might occur accidentally or maliciously. When it comes to edge computing, the issue intensifies considerably. The unique characteristics of edge computing allow us to maintain traditional methods of maintaining data integrity with edge data from cloud service providers.

First, edge servers typically have low processing power, whereas most cloud data integrity techniques assume and leverage the near-limitless computing power of cloud servers. Then, to serve regional customers, service providers typically cache their viral data on edge servers in specific locations. The service provider cannot look at each edge data point separately to find the problematic ones because of excessive processing and bandwidth consumption. One technique now being used to guarantee cloud data security is proven data possession (PDP) and its variations. They are intended to enable users of thin clients to seek data integrity challenges from major cloud storage providers such as Google or Amazon.

Data deduplication is when a storage provider keeps just one duplicate of a file (or portion of a file) that several users hold. Four distinct deduplication methods differ based on whether Deduplication happens at the file or block level and whether it happens on the client or server side. Because client-side data deduplication ensures that many uploads of the same content

only consume a single upload's network bandwidth and storage space, it is preferable to server-side data deduplication.

Although security was not included in the initial deduplication designs, it rapidly became necessary when users sought data protection. Deduplication seems possible with convergent encryption, which uses plaintext as the encryption key. This technology is secure and easy to use. Regretfully, it was shown to be susceptible [1]. Moreover, a general impossibility conclusion contends that basic convergent encryption techniques cannot provide traditional semantic security.

Figure 1 describes Deduplication in edge cloud using the chuck files to store the cloud, first segmenting the file in a cluster set to reduce the similarity hash contents. Reduce the number of computation operations for each hash that computes a part of the hash and minimize duplication. This further accelerates the decoupling process while achieving the same decoupling result. On the other hand, tags assume a user-based response mechanism instead of duplicating files, so cloud users cannot cheat users using duplicate check results when files are uploaded. Complete, and the same checks will be ignored. Therefore, Deduplication is assumed to exist.

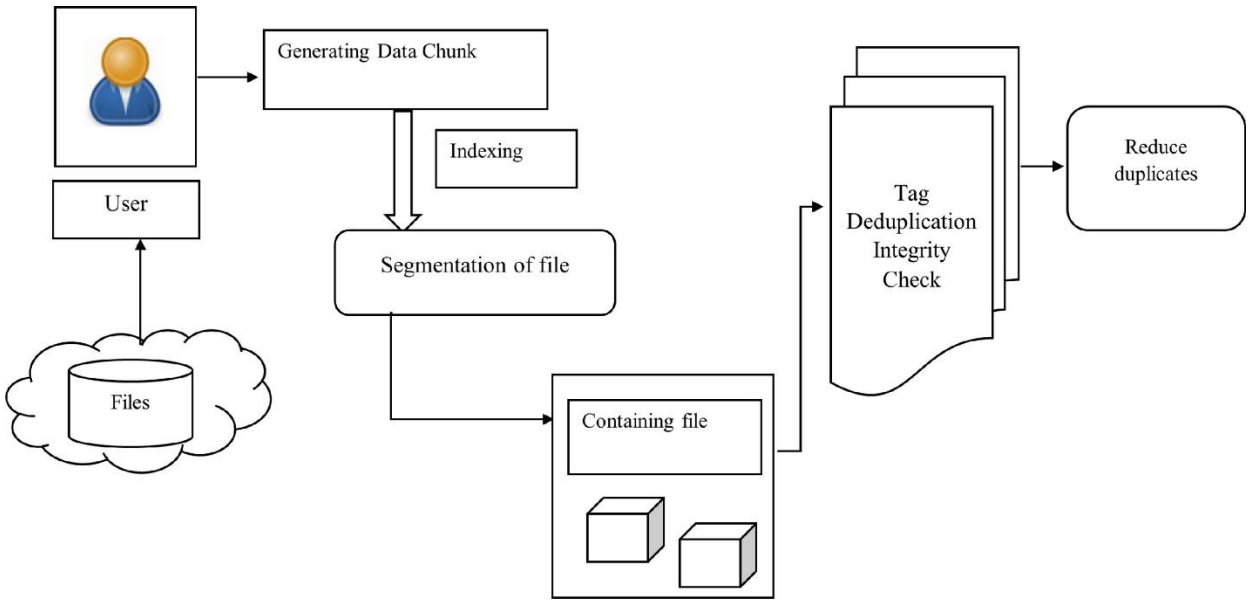

Figure 1. The basic flow of Deduplication in the cloud

2. PRELIMINARIES

Evaluating previous literature can uncover gaps, contradictions, or limits in the current understanding of the issue. This information contributes to the creation of research topics and helps justify the necessity for additional research.

End-to-end encryption is becoming increasingly popular due to recent data breach incidents, as more individuals and businesses are outsourcing their data to cloud storage. Regretfully, semantically secure encryption renders several affordable storage optimization strategies, including data deduplication, useless [1]. Users want to keep encrypted data on cloud servers to safeguard their privacy. Cloud servers reduce the cost of storage and network traffic by removing redundant copies. First, aggregation bias is used to resolve the potential risk of internal data leakage [2].

Customers can access low-latency services using Edge Storage Systems (ESS), which are made up of linked edge servers in a specific location. Conversely, poor edge server storage capabilities result in large data storage overheads, a significant obstacle to maintaining an ESS's application performance [3]. However, in edge-based mobile crowdsensing systems, it could be challenging to carry out safe data deduplication while also shielding participants (like clients and recruited mobile users) and sensing data from different threats (like external and internal assaults) [4]. Data deduplication can improve communication efficiency while saving storage space. However, data deduplication brings new edge computing functionality and security concerns [5].

The interaction above is much reduced by fine-grained admission control using client-level keys and update tools since the cloud server handles the proprietorship change [6]. In any case, no current work can give different deduplication effectiveness and assurance for various information lumps. A security-mindful and compelling information deduplication plot for edge-helped distributed storage frameworks. It works on the proficiency of Deduplication and lessens data spillage brought about by recurrence examination assaults [7]. The smooth execution of a considerable measure of holders simultaneously on the hubs with restricted figuring assets remains a test [8]. It has been shown that the summed-up Deduplication (GD) pressure calculation offers serious pressure proportion, throughput, and arbitrary access properties that empower direct examination of packed information [9].

Even with the widespread use of deduplication techniques by cloud service providers, a substantial amount of overhead bandwidth still needs to be increased. To reduce this extra traffic, a middleware deduplication layer that verifies the legitimacy of files without accessing the cloud is created [10]. Provide CaseDB, a unique key-value store that aggressively isolates keys and bloom filters on the nonvolatile memory express (NVMe) disk and saves the SSD values to overcome the main problems with an LSM tree [11].

Portable edge processing is arising to deal with the sheer volume of delivered information and arrive at the inactivity interest of calculation escalated IoT applications. However, the development of versatile edge figuring on assistance and

idleness has been very much considered; security and proficiency in information utilisation in portable edge processing have yet to be distinguished [12]. Identifying Cloud Security threats and mitigation strategies that detect data leakage, tampering and other vulnerabilities in cloud service deployment are significant [13].

Edge processing answers clients' solicitations with low inactivity by putting away the pertinent documents at the organisation's edge. Different information deduplication innovations are presently utilized at the edge to dispose of repetitive information lumps for space saving. Be that as it may, the query for the worldwide gigantic volume of finger impression lists forced by recognizing redundancies can debase the information handling execution [14]. Previous encrypted data deduplication algorithms were primarily built for standard two-tier cloud storage. It cannot be used with the 3-layer fog assist cloud storage currently under development [15]. Table 1 describes conventional methods for data deduplication.

Table 1. Comparison of existing methods for data deduplication

Reference	Proposed Method	Drawbacks
Ni et al. [16]	A safe data deduplication system helped by fog (Fo-SDD)	Sensed data is often protected, making it challenging to deduplicate
Jiang et al. [17]	An effective PoW procedure combined with secure data	Cannot effectively resolve dynamic ownership changes and specific proof of ownership (PoW)
Li et al. [18]	deduplication strategy for active ownership management Energy Efficient Storage System (EESS)	These storage systems consume much energy to achieve high performance.
Yu et al. [19]	Tag-flexible Integrity Check Protocol with Deduplication Support (TDICP)	Conceive customers into purchasing unnecessary storage for redundant data only kept once.
Yang et al. [20]	User-defined access control-compatible secure deduplication technique that is effective	Eliminate redundant encrypted data to save storage and transmission costs.
He et al. [21]	Routing Strategies in Clustered Deduplication Systems	High deduction rates and low effectiveness rates.
Guo et al. [22]	Double Sliding Window Chunking Algorithm	It is not easy to find the file
Yuan et al. [23]	lightweight rekeying-aware encrypted deduplication scheme (REED)	Difficult to store the data in the cloud
Jin et al. [24]	jump-based chunking (JC) approach	The deduction rate is lower than that of the CDC approach due to the inability to solve the boundary transition problem.
Chen et al. [25]	Dynamic, searchable symmetric encryption (DSSE)	Complex optimization processes and inflexible query methods.
Wang et al. [26]	Lightweight Secure Deduplication Scheme	The complex symmetric key distribution

Edge computing is a crucial part of green computation because it enables real-time applications, reduces the amount of computing and network resources needed to transmit data to the cloud for processing, and locates cloud resources closer to the Edge [27]. A model for shifting processing from mobile to wired networks is called share-based edge computing, or SEC (M2W). Tasks from mobile devices can be divided among wired devices for processing under the oversight of SEC servers [28].

It is a significant problem for service providers, but more needs to be discussed. Therefore, in the context of edge computing, accurately and successfully verifying the integrity of edge data is a critical security concern [29]. App providers can host services for local consumers on the Edge Server Network (ESN), composed of edge servers in a particular region and the connections that link them. Several recent studies have demonstrated that large ESN densities offer excellent service performance because edge servers can efficiently share resources and interact throughout the ESN [30, 31]. Fifth-generation (5G) networks can benefit significantly from the improved spectral efficiency, higher quality of service, and reduced latency that NOMA and Mobile Edge Computing (MEC) can provide.

This is a big concern for service providers, yet little has been done about it. Therefore, in edge computing [32], precisely and successfully confirming the integrity of edge data presents a significant security problem. App providers can host their services for local clients on the Edge Server Network (ESN), which comprises edge servers in a particular region and the connections that connect them. Several recent studies have shown a high ESN density to enhance service performance since edge servers can effectively communicate and share resources across the ESN. Fifth-generation (5G) networks are finding that NOMA [33] and Mobile Edge Computing (MEC) are crucial enablers due to their improved quality of service, reduced latency, and excellent spectrum efficiency [34].

Because of high latency and a rise in task failures in a dynamic environment with frequent changes and unexpected end-user demand, executing offloaded activities on a subpar server may lower the quality of service [35, 36]. The resource management issue at the edge server is addressed using a Reinforcement-Learning-Based State-Action-Reward-State-Action (RL-SARSA) technique. Additionally, it chooses the optimal offloading strategy to lower system expenses like compute latency and energy consumption [37]. Cloud storage raises significant security issues. Data breaches can lead to severe violations of privacy and data integrity, highlighting the need for robust security measures [38].

3. PROPOSED SCHEME

A lot of storage space is one of the many resources a cloud service provider offers its clients. Managing the constantly expanding volumes of data in the cloud is one of the main problems. In cloud edge computing, the DD approach improves data management's scalability. However, the primary issue with data deduplication is security. To overcome

this issue, this study provides a safe data deduplication system via an integrated cloud-edge environment using convergent and Unique Tag Deduplication Integrity Check (UTDIC) methods. This algorithm classifies people according to their traits. Depending on whether the number of owners of a collection of data matches the degree of popularity, the data set may be considered duplicate.

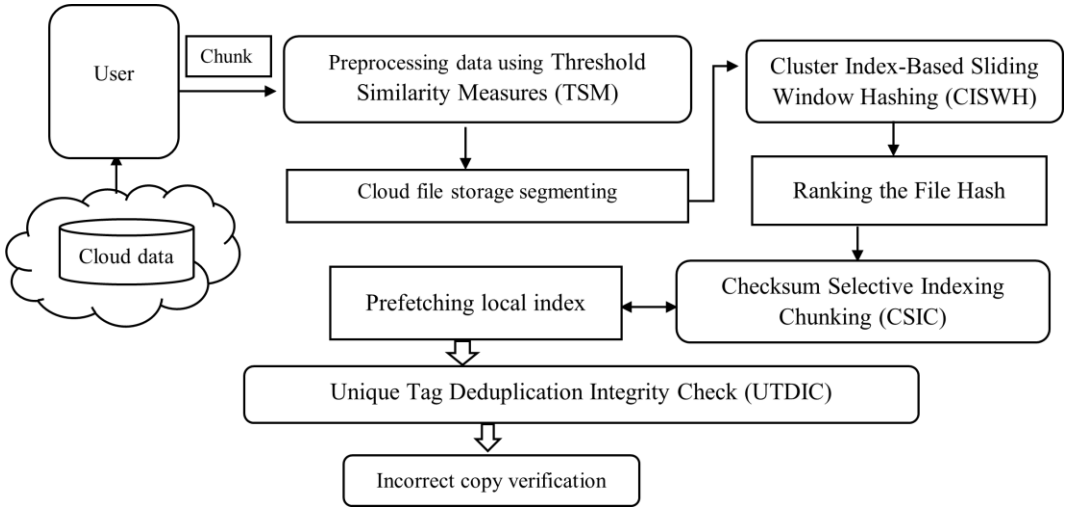

Figure 2. Proposed system

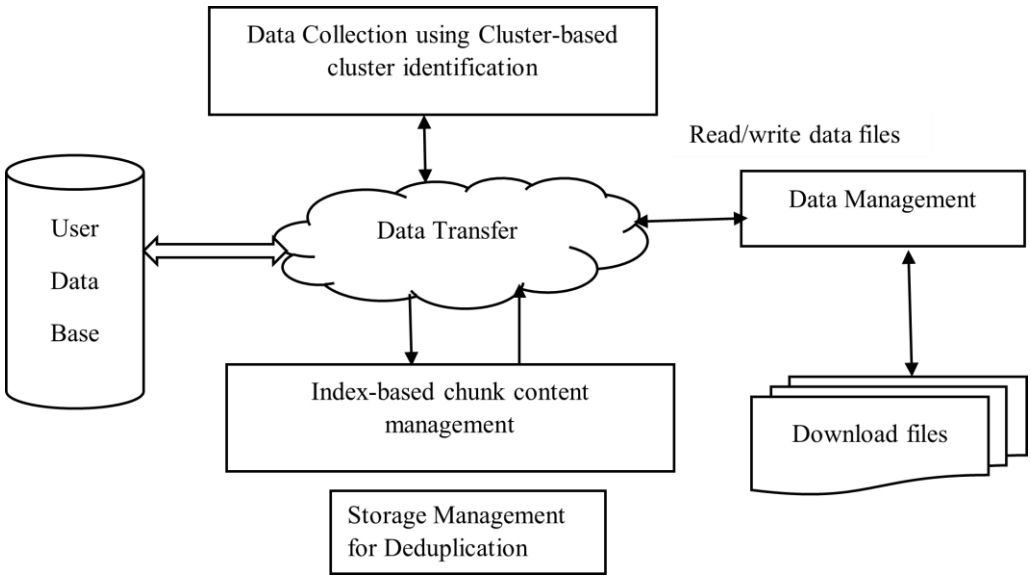

Figure 3. Diagram for edge could deduplication

Figure 2 provides a proposed system for Deduplication using in the cloud and then generating the chunk file to reduce the duplicates, initially preprocessing using Threshold Similarity Measures (TSM) to estimate the text and copies are removed, then segmenting the cloud folder using CISWH for storing the file in cluster format and ranking the order based on the cloud for hashing identification. Checksum Selective Indexing Chunking (CSIC) is finding the selective index range for chunking content using Verify the integrity of deduplicated data by integrating a checksum or error-checking appliance. Prefetching the local index means allocating the store's frequently accessed data blocks to the local cache. This reduces redundant operations on processed chunks and improves overall deduplication speed. Finally, the Unique Tag Deduplication Integrity Check (UTDIC) is a validation mechanism to verify the uniqueness of the tag when creating or assigning a new tag. This includes checking the repository to make sure the suggested tag is correct.

3.1 Edge cloud deduplication model

Only on centralised cloud servers does edge computing allow data to be processed closer to the point of generation. The process of eliminating multiple copies of repeated content is known as Deduplication. It is frequently used in storage systems to boost data speed and reduce the amount of storage space needed.

Figure 3 shows a balanced design for end users and end service providers. This edge computing solution consists of user, storage, and management nodes for error file reduction and Deduplication.

3.2 Preprocessing data using Threshold Similarity Measures (TSM)

Preprocessing uses a word-based similarity metric to discover inputs that surpass a predefined similarity threshold.

Sets the point at which inputs are deemed identical. Using a similarity metric based on threshold points can help to decrease duplicate files in cloud storage. To avoid duplicate data sets, use thresholding similarity algorithms to locate related texts and delete unnecessary information.

$$Ther(text) = 1 - \frac{\sum_{x=1}^{N} Sc_x(txt)}{N} \quad (1)$$

where, $Thre(text)$ is the threshold of a particular text, and the N number of records in cloud data, index (x) SCx (text) is a similarity finding, and SCx (text) =1.

If the x^{th} values of a particular similarity score is between 0 and 1, cloud records X1 and X2 of records calculating similarity score values of texts. If the similarity score range between two values of a field is greater than or equal to that field's range, then the Similarity Score (SS) is defined as:

$$Sc(x1, x2) = t_1 * V_1(x1, x2) + t_2 * V_2(x1, x2)$$

Similarity=0;
For n=1 to X-1 do
$Thre(n) = V(n) - V(n+1)$;
While $Max\ (Sc) > 0$
X=finsTh.sc (max)
Swap $(V(n) - V(n+1))$;
Remove similarity;
End while

X 1 and X2 are input strings or records to be compared field by field. T1 and T2 are weighted by the similarity and string similarity algorithms, respectively, and are equal (t1=t2=0.5). V1 represents the similarity score, and V2 represents the string similarity score. This code shows the algorithm for obtaining the similarity class. It is an iterative process that extracts the outermost node from the entire set.

3.3 Cluster Index-Based Sliding Window Hashing (CISWH)

Compared to reduced window or landmark models, very few research has concentrated on clustering algorithms that use sliding windows. A CISWH algorithm that is effective. The technique seeks to deliver quick, excellent clustering outcomes. Compared to previous methods, our method's unique data structure and process can perform full-range clustering on tuples and reduce the computational cost of operations such as insert, delete, and search. Sliding window aggregation divides the window into blocks and aggregates the summaries of these blocks to create a window.

Our method uses hashing, whereas the other uses a tree structure to find the nearest CF. The ' current state is S1=c1, c2, S2=c3, S3=x4, and the input variables are c1, c2, ... (ci+1 is fresher than ci). The current loads are combined when a new tuple, c5, appears, and a new bucket with the new tuple is created, with S1=c1, c2, S2=c3, c4, and S3=c5. A sliding window advances and deletes loads that have expired timestamps. Nonetheless, the timestamps within a container could differ. If the sliding window in the example has a size of 4, it should remove the tuple x1.

3.3.1 Sliding window

Only arrays with timestamps from the window's start to the current timestamp are included in sliding windows. In particular, the timestamp of the tuple x_i and the current timestamp t can define the window as a weight function.

$$\boldsymbol{s}w(x - x_i) = \begin{cases} 1, if\ x - x_i \le V \\ 0, ifx - x_i > V \end{cases} \quad (2)$$

where, V is the sliding window's temporal range, the window removes the Cluster's loads based on the vectors. When another 100 tuples come, the previous 100 are removed from the window, and the new 100 are attached. The terms "a window slides" and "a window moves" indicate that when set SLIDE to V, the window's oldest V tuples are eliminated, and fresh V tuples are inserted.

Sum←0;
For x←1 to V do
$Sum \leftarrow Sum + X(i)$;
End
Result←sum;
For x← v+1 to n do
$Sum \leftarrow sum + x(i) - x(x - x_i)$;
Result← Max (Res, sum)
End
Return

First, all possible starts of the range are repeated. Each range's elements are iterated from V to V+n-1, and their sum is calculated.

Depending on the sliding condition and time unit, sliding windows can be classified as either time-based or triple-based. An index-based window for clarity, while a time-based window may also be examined using the same techniques.

3.3.2 Cluster index

A clustered index is constructed using a column or group of columns that uniquely identify each entry. This makes index maintenance easier.

Assuming the incoming data streams $x = hx1, t1i, hx2, t2i, ...$, I cannot fulfill that request for the sliding interval L=3, with 3 clusters (c=3) and a window size R=9. |C|=5.

The number of tuple structures, their square sum, and their linear sum are all maintained by the Clustering Feature (CF). If the radius of a new tuple is less than a predetermined threshold, the nearest CF absorbs it.

$I \le C^I$ is the clustering index range to be calculated based on the data distance evaluation using weights w_1, w_2,

$$(w_1, w_2) = ||w_1 - w_2||^2 = \sqrt{\sum_{x-1}^{w} (|w_1 x - w_2 x||^2)} \quad (3)$$

$$Const\ (w, v) = \sum_{x \in V} range^2\ (w, v) \quad (4)$$

$$Const_{\ w}(w, v) = \sum_{x \in S} w(x).weight^2(w, v) \quad (5)$$

Aggregation aims to reduce the total squared distances between each tuple in S and the nearest range. The clusters have a weight function value with a sliding window. Figure 4 illustrates the cluster indexing grouping.

$$Const\ (w, v) = \sum_{x \in V} range^2\ (w, v) * (t - t_w) \quad (6)$$

$$Cont^{n}_{Opt}(w) = \min_{C \in range^2, ||C||=n} const(w, v) \quad (7)$$

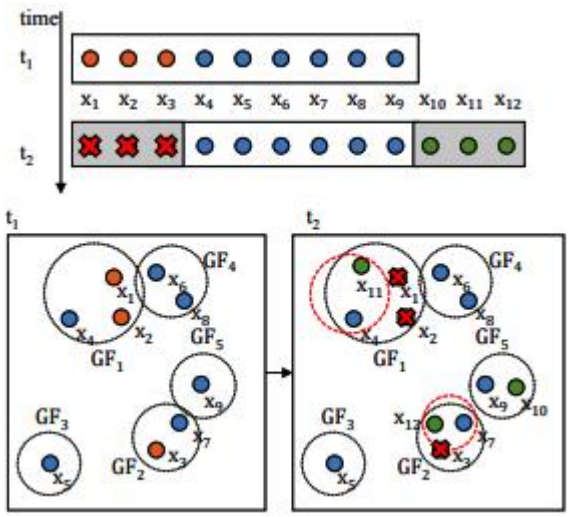

Figure 4. Cluster index grouping

The clustering sliding window is processed in the cluster center, and the indexing range follows each Cluster; the quality of the clusters depends on the index cluster centers.

3.3.3 Cluster index hashing with Sliding window

$$x_1 = 0, x_2 = 0; cont = 0 \quad (8)$$

Sliding window cluster index (data)
While x_2 is less, the counting of the clusters

$$x_2 = x + 1 \quad (9)$$

Error rate Index. Cluster (Data, x_1, x_2)
If index err is generating the Hash

$$x_1 = x_2$$
$$Cx_1(Con) = x_2$$

End if
Hash Index (x_1, x_2))
If H←Hash (X. Get cluster Number (Chunk))
SW←H.get (H)
Delete By Key (H Chunk)
It is Joint (x, y) then
Insert ((x, y))
Else
Data_ Insert (H)
Count the number of Chunks
Return
End if
End while

In addition, blocks often contain features much deeper than the average sort depth, which increases the RAM required to store hash tables in a particular location. Instead of a hash table with conflict resolution, use a hash table H with fixed size h without conflict resolution. Differently named alignments are assigned the same hash value.

3.4 Checksum Selective Indexing Chunking (CSIC)

A checksum is a value calculated by summing the bits in a data collection. It is used to identify mistakes in data transfer or storage. Selecting specific data or material for inclusion in an index is called selective indexing. In databases or search engines, it entails choosing which qualities or fields to include in an index to permit faster and more efficient search operations. Indexing Chunking is the technique of dividing vast information into smaller, more manageable "chunks." It is frequently used for memory and learning.

Input: Default values, Length of sliding window, W
Output: Index selection
Function Checksum Chunking (File, values)

X=1;
Indexing Checksum=0;
While (Checksum=read (file))
Array [index%W+1]=checksum length
If selecting. Length>=V then
If the hash value (Checksum, index,v)==Checksum values, then
Return x
End if
Else
Continue
End if
X=x+1
End while
End

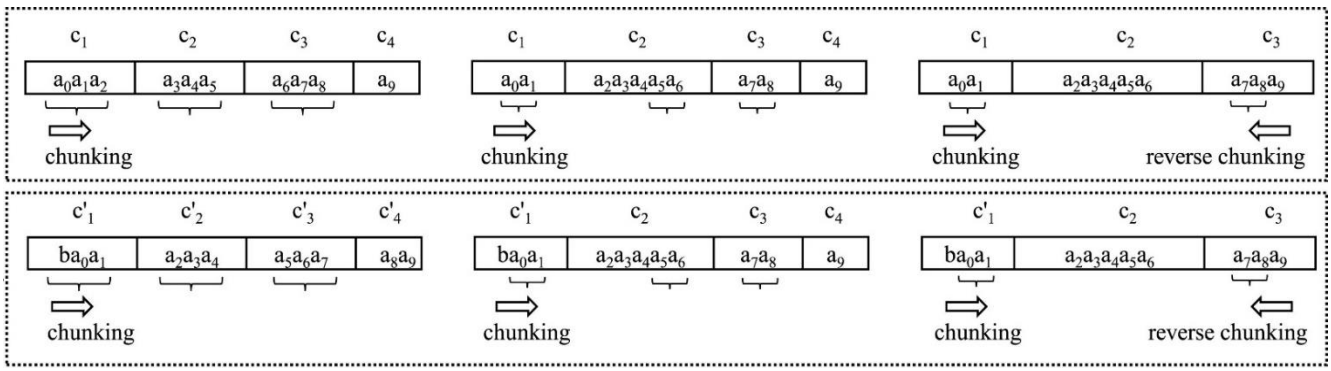

Figure 5. Chunking process

Figure 5 shows a comparison of the Chunking Method. (A) While fixed-size chunking is fast, it has a boundary-shift problem. (a) The issue can be resolved using variable-size chunking, but each byte requires some processing. (c) This problem can be improved with the time complexity of O(1) by using the constant-time chunking that this study suggests; variable-size and reverse chunking, on the other hand, only identify the first and final chunks, leaving a sizeable middle chunk c.

File types and sizes are represented as an array of bytes in the input.

Different-sized File Chunks, Max Th and Min Th, are generated.

Step 1: Configure the File Length (f).

Find the Length of the L.

Set the Endpoint to 0 and the Startpoint to 0.

Step 2: Though Not the End of the DocumentIf Len is equal to zero, then move on to Step 5. Should Len match Min Th

Proceed to Step 3 after setting the File Chunk Bytes from Startpoint to (Startpoint+Len).

Else If (Min Th.+L)=Len, Then

Proceed to Step 3 after setting the File Chunk Bytes from Startpoint to (Startpoint+Len+L).

Endpoint

Bytes of File from Endpoint+L should be selected as the window size.

Step 3: End

Additionally, chunking files with minimum and maximum values are sometimes used as a hard barrier to prevent massive storage of chunk size variation factors.

3.5 Prefetching Local Index Caching (PLIC)

Prefetching Local Index Caching is an approach used in computing to increase system speed by retrieving data or instructions on a local system or device before generating an index rather than depending on a centralized index. Reducing the need to query a central index can improve access times for frequently used data. Caching is keeping copies of frequently requested data at a location that enables quicker retrieval. Local index caching may refer to retaining a cache of commonly used index data on a local system to speed up search or retrieval processes.

$$C_n = (f_i v : v \in W, w \in |w| = t, x \in |N|) \tag{10}$$

$$c_d = \left(\oplus \in W; \frac{f_i v}{[s]} : S \subseteq |N|, |s| = t+1\right) \tag{11}$$

Which requires broadcasting $v \in W : S$ bits. Note that user k∈S, for S), wants $\frac{f_i v}{[s]} : S$ and has cached $t \in W$ for all s∈S: s 6=k so that it can recover $\frac{f_i v}{[s]} : S$ rom Xt and the cache content.

$$w_{x,y} Max\,(t) = \frac{\binom{N}{T+1} - \binom{N-\min(n,m)}{t+1}}{\binom{N}{t}} \geq V^x \tag{12}$$

$\binom{N}{T+1}$ With $t \in v$ the sub set t- an integer, one takes the lower convex envelope of set of points (x,v)=$\binom{N-\min(n,m)}{t+1}$ for t∈[0: n].

$$\in_n := \max_{x \in [n]} Pre\{w_x(x^n, (N : x \in i)) \neq D_v\} \tag{13}$$

$$R \in \bigcap_{x \in |n|} \bigcup_{Xv' Ci \in V} R\left(\frac{N}{P} + Ci\right) \tag{14}$$

$$\bigcup_{Xv' Ci \in V} R\left(\frac{N}{P} \middle| Ci\right) = R(P+1) \tag{15}$$

$$\sum_{X : X \in |N|} Ux < 1 \tag{16}$$

The attainable step technique for composite (index) coding is based on prefetching decoding. If the cache placement phase is coded in caching, the delivery phase is a well-specified message and demand set.

3.5 Unique Tag Deduplication Integrity Check (UTDIC)

Users can use UTDIC to create unique validation tags to ensure data integrity across CSPs and enable Deduplication. At the same time, to prevent the results of CSP duplicate checking during the file upload process from misleading users, UTDIC is based on the private set intersection, which allows users to determine whether files are duplicates before CSP.

UTDIC requires challenger A to reply with (R) valid note sets to pass the integrity check.

$$\delta = \sum_{i=1}^{\gamma} P^A_{(Sucess,i)} = (1-\rho_{adv})^{\gamma} + \underbrace{\frac{\gamma \rho_{adv}(1-\rho_{adv})^{\gamma-1}}{2^{ns}}}_{\xi} + \underline{o(\frac{1}{2^{ns}})} \tag{17}$$

UTDIC can recover rD ¼ d if there are more than d 2 corrupted errors in a block, our protocol cannot recover the blocks.

$$P^{\sigma}_{(Fail,i)} \leq exp(-\frac{\rho_{adv} D}{3}(1-\frac{\rho}{\rho_{adv}})^z) \tag{18}$$

$$(1-\frac{\rho}{\rho_{neg}})^2 \rho_{neg} = \frac{3\ln(2)\tau}{D} and \rho_{neg} < \rho \tag{19}$$

Next, choose a threshold (T) for the query time g so our protocol will discover a chunk reduction. A with an overwhelming likelihood if it corrupts more than γ neg proportion of the blocks. $if\ g >= neg\ and\ radv > \eta neg. Next$

$$\begin{aligned} Resp * b^{-1} modm &= (v \times D) * b^{-1} modm \\ &= (be \times D) * b^{-1} modm \\ &= e \times D modm \end{aligned} \tag{20}$$

$$D = \begin{pmatrix} d_{11} & \cdots & d_{1y} \\ \vdots & \ddots & \vdots \\ d_{x1} & \cdots & d_{xy} \end{pmatrix} \tag{21}$$

Then

$$\begin{aligned} e \times D \mathrm{mod} m &= \left(\sum_{i=1}^{x} e_i d_{i1}, \sum_{i=1}^{x} e_i d_{i2}, \ldots, \sum_{i=1}^{x} e_i d_{iy}\right) \mathrm{mod} m \\ &= \left(\sum_{i=1}^{x} e_i d_{i1}, \sum_{i=1}^{x} e_i d_{i2}, \ldots, \sum_{i=1}^{x} e_i d_{iy}\right) \end{aligned} \tag{22}$$

Demonstrate the efficacy of utilizing the UTDIC approach to retrieve the query column containing the notes, enabling many users to generate their verification tags while maintaining support for tag deduplication at the CSP. Additionally, try your hardest to guarantee that the data duplication check is accurate.

3.5.1 Deduplication file

The proposed approach of deduplication functions at the file level. Only text files (.txt) are allowed. The user and his cloud environment are the two stakeholders. Google Drive is the cloud environment in this case. The files to be uploaded are selected by the user using this method. The UTDIC technique is used to produce the file's hash. The hash value of the cloud and uploaded files are compared throughout the upload process. Uploading of the file will be blocked if a match is discovered. The chosen file will be uploaded successfully, even without a support file.

The new file's bin information and the segment numbers that were generated for it are added to the file-to-segment metadata. The segments that are duplicates have the same segment number. The quantity of file-generated unique Tag segments is updated in the bin structure configuration file. In essence, the number of segments beneath each bin is counted in this file. This module uses gzip to compress and decompress files, saving additional storage space.

```
Begin Deduplicated
        Index_file [] cluster chunk=
        Hash_cluster_Identifier ()
                ForEach(Seg in clusters[])do
                        Index_clusters←     calculating
hash(seg)
                        If (Index_file==Hash[]) then
                                Downloading        file
authenticated user
                        End
                        Else
                                Error Message
                        End
    End
    End
```

A list search has already produced the computed hash value. The hash () function generates this list lookup. Specific segments are not preserved if the hash value is present. If not, the section is stored as indicated and then downloaded afterward.

4. RESULT AND DISCUSSION

The outcomes are verified using an EC2 server instance, a cloud environment powered by Amazon Web Services (AWS), and EBS storage; a duplicate dataset is created by combining the gathered content files to provide deduplication redundancy storage. The UTDIC implementation uses lookup indexing tables and indexed hash table files to perform block-based comparisons. This technique yields better results than others when tested using a confusion matrix to gauge efficiency in terms of precision, recall, false rate, and accuracy of storage optimization. Table 2 provides a clear illustration of simulation processing.

The introduced technique provides better performance under different levels of testing by ensuring accuracy of collection, time complexity, and recall.

Table 2. Simulation processing

Simulation Parameters	Values
Environment of Cloud	AWS cloud storage
Size of data	20Gb
Tool	Visual Studio/ c#

Table 3. Analysis of precision rate

Performance of Precision Rate in %					
Storage /methods	REED	DSSE	TDICP	RSLI-SCCC	UTDIC
10GB	74.3	87.3	89.1	92.4	94.8
15GB	76.8	84.6	85.4	94.9	95.4
20GB	77.2	85.5	87.8	96.3	97.9

Table 4. Analysis of false rate

Analysis of False Rate %					
Storage /methods	REED	DSSE	TDICP	RSLI-SCCC	UTDIC
10GB	6.5	5.2	4.9	4.6	3.8
15GB	7.8	7.4	6.2	5.6	3.5
20GB	10.3	9.6	7.3	4.4	2.3

Table 5. Performance of storage

Performance of Storage Rate in %					
Storage /methods	REED	DSSE	TDICP	RSLI-SCCC	UTDIC
10GB	6.8	10.4	15.6	20.5	28.2
15 GB	7.3	14.9	18.1	23.4	33.6
20 GB	15.7	23.3	26.5	34.7	43.2

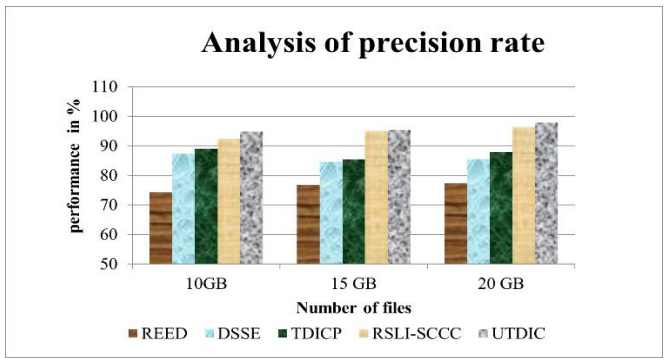

Figure 6. Performance of precision rate

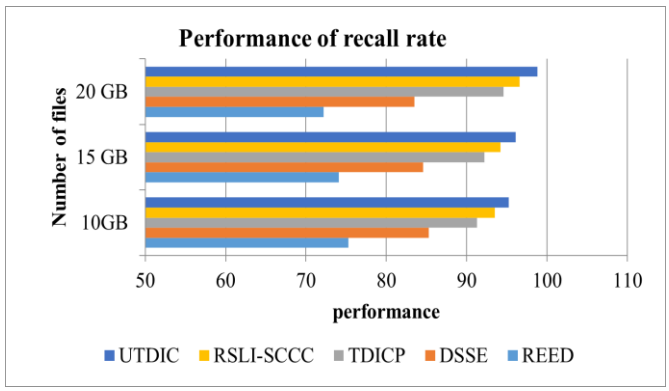

Figure 7. Performance of recall rate

Table 3 shows the performance comparison results of invention accuracy by different methods. Compared to other

approaches, the proposed UTDIC can provide improved accuracy in terms of precision ratio.

Figure 6 clearly illustrates different methods for diverse data sizes to observe the performance of the tested accuracy. The proposed UTDIC performs at 97.9% and can be associated with the tested pipeline in 10GB size. While the RSLI-SCCC method attained 96.3%, the existing REED, DSSE, and TDICP methods are 77.2%, 85.5%, and 87.8% of precision performance.

Recall performance can be measured using various methods, as illustrated in Figure 7. The proposed method shows 98.8%, compared with the tested methods on 10GB data. Also, the proposed UTDIC algorithm can achieve higher memory efficiency than other methods. While the RSLI-SCCC method attained 96.6%, the existing REED, DSSE, and TDICP methods are 72.2%, 83.5%, and 94.6% of recall performance, respectively. The previous methods obtained less recall rate performance.

The results show that the proposed UTDIC algorithm's classification rate is lower than that of other methods. Furthermore, Table 4 lists the error rate production rates measured by the different methods.

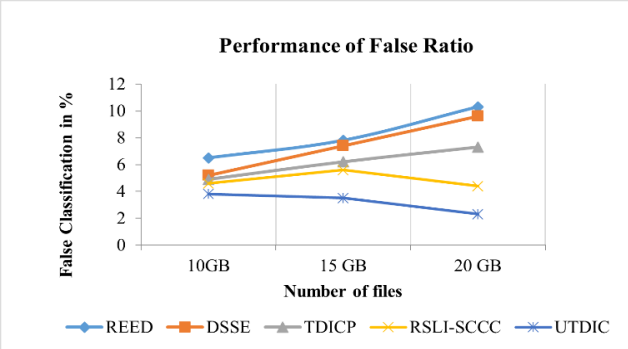

Figure 8. False classification rate

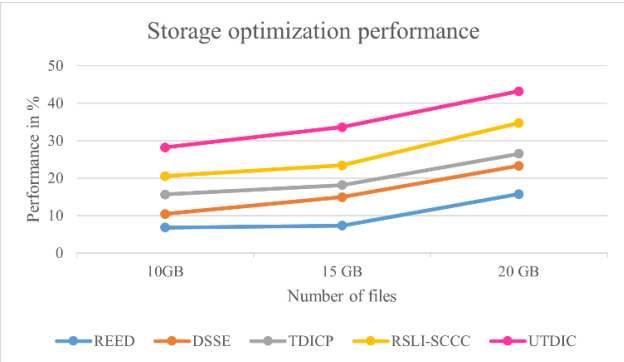

Figure 9. Comparison of storage optimization performance rate

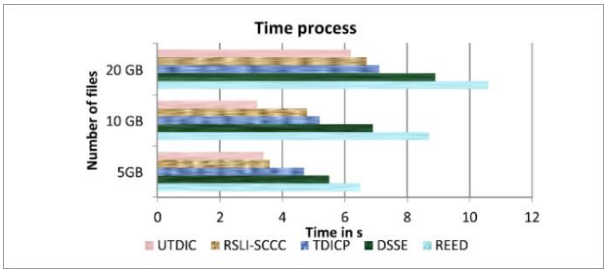

Figure 10. Time process

Figure 8 illustrates the many approaches used to quantify the productivity of the error redundancy rate. The suggested UTDIC is 2.3% compared to the techniques evaluated on 10GB of data. According to the data, the proposed UTDIC process has a lower organization ratio than alternative techniques. While the RSLI-SCCC method attained 4.4%, the existing REED, DSSE, and TDICP methods are 10.3%, 9.6%, and 7.3% of false rate performance. The previous methods obtained high false rate performance.

Performance analysis Figure 9 shows the results of the storage enactment ratios of the diverse methods in memory. Compared to the methods tested on 10GB of data, the offered UTDIC gives the best performance, up to 43.2%, reduces data duplication, and minimizes memory consumption. However, the traditional methods still need to achieve efficient storage optimization performance.

Table 5 analyzes the storage performance rate performance measure and provides a variance estimate for the compared datasets. The comparison results show that the suggested UTDIC system uses less storage than the other approaches.

Figure 10 above shows the production measurement time flow for different methods. The proposed UTDIC algorithm reduces the time compared to other methods. Compared to the methods tested with 10GB data, the proposed UTDIC yields the best performance up to 6.2 (s), DSSE yields 8.9 (s), and RSLI-SCCC delivers 6.7 (s).

4.1 Discussion

The experimental results show that the proposed method attains proficient precision, recall, false rate, storage optimization, and time process performance. The proposed method reaches 94.8%, 95.4%, and 97.9% for 10GB, 15GB, and 20GB, respectively. Similarly, the existing method individually obtained 77.2%, 85.5%, 87.8%, and 96.3% for REED, DSSE, TDICP, and RSLI-SCCC.

Another parameter is recalling comparison performance. The proposed method attained a recall result of 98.8%; similarly, the traditional methods like the RSLI-SCCC method attained 96.6%, the existing REED, DSSE, and TDICP methods are 72.2%, 83.5%, and 94.6% of recall performance, correspondingly.

Also, storage optimization performance outcomes in the cloud environment. The proposed UTDIC method attains 28.2%, 33.6%, and 43.2%. Likewise, the time process and false rate performance achieved fewer outcomes by the proposed method. However, the traditional method gained a higher time process and false rate performance.

5. CONCLUSION

In conclusion, edge cloud deduplication is now a vital remedy in distributed computing. This technique enhances the overall performance of edge devices in addition to optimizing bandwidth utilization and storage economy. Duplication minimizes latency, speeds up data transfers, and eases the load on network resources by removing redundant data at the edge. With edge computing still a significant player in many sectors, it is increasingly dedicated to efficient operation data management. Deduplication is the most well-known method of data compression. Many existing approaches presented various deduplication algorithms but were insecure in a cloud-based secure deduplication solution based on convergent and

UTDIC algorithms. Using file sizes ranging from 10GB to 20 GB, with a 10GB increment in each iteration, the suggested system's performance was assessed. According to the performance research, the proposed method has a 43.2% storage optimization performance, which is higher than the other existing approaches and has a promising outcome. To guarantee the correctness of the duplication check, however, we employed a novel challenge and answer approach in the duplication check UTDIC, allowing the data deliverer, rather than the CSP, to ascertain if a file is duplicated first. According to security and performance, UTDIC is safe and effective within the specified security model. In the future, we can improve de-duplication performance using an indexing approach.

REFERENCES

[1] Stanek, J., Kencl, L. (2016). Enhanced secure thresholded data deduplication scheme for cloud storage. IEEE Transactions on Dependable and Secure Computing, 15(4): 694-707. https://doi.org/10.1109/TDSC.2016.2603501

[2] Teng, Y., Xian, H., Lu, Q., Guo, F. (2022). A data deduplication scheme based on DBSCAN with tolerable clustering deviation. IEEE Access, 11: 9742-9750. https://doi.org/10.1109/ACCESS.2022.3231604

[3] Luo, R., Jin, H., He, Q., Wu, S., Xia, X. (2023). Enabling balanced data deduplication in mobile edge computing. IEEE Transactions on Parallel and Distributed Systems, 34(5): 1420-1431. https://doi.org/10.1109/TPDS.2023.3247061

[4] Li, J., Su, Z., Guo, D., Choo, K.K.R., Ji, Y., Pu, H. (2020). Secure data deduplication protocol for edge-assisted mobile crowdsensing services. IEEE Transactions on Vehicular Technology, 70(1): 742-753. https://doi.org/10.1109/TVT.2020.3035588

[5] Ming, Y., Wang, C., Liu, H., Zhao, Y., Feng, J., Zhang, N., Shi, W. (2022). Blockchain-enabled efficient dynamic cross-domain deduplication in edge computing. IEEE Internet of Things Journal, 9(17): 15639-15656. https://doi.org/10.1109/JIOT.2022.3150042

[6] Lang, W., Ma, W., Zhang, Y., Wei, S., Zhang, H. (2020). EdgeDeup: An edge-IoT data deduplication scheme with dynamic ownership management and privacy-preserving. In 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chongqing, China, IEEE, 1: 788-793. https://doi.org/10.1109/ITNEC48623.2020.9085119

[7] Xie, Q., Zhang, C., Jia, X. (2022). Security-aware and efficient data deduplication for edge-assisted cloud storage systems. IEEE Transactions on Services Computing, 16(3): 2191-2202. https://doi.org/10.1109/TSC.2022.3195318

[8] Shen, G.L., Lee, C.R. (2022). FLOMD: Fast and low overhead memory deduplication for edge nodes. In 2022 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Bangkok, Thailand, pp. 83-90. https://doi.org/10.1109/CloudCom55334.2022.00022

[9] Hurst, A., Lucani, D.E., Assent, I., Zhang, Q. (2022). GLEAN: Generalized-deduplication-enabled approximate edge analytics. IEEE Internet of Things Journal, 10(5): 4006-4020. https://doi.org/10.1109/JIOT.2022.3166455

[10] Aparna, R., Bandopadhyay, S., Pandey, S. (2021). Blockdrive: A deduplication framework for cloud using edge-level blockchain. In 2021 International Conference on Communication Information and Computing Technology (ICCICT), Mumbai, India, pp. 1-6. https://doi.org/10.1109/ICCICT50803.2021.9510039

[11] Tulkinbekov, K., Kim, D.H. (2020). CaseDB: Lightweight key-value store for edge computing environment. IEEE Access, 8: 149775-149786. https://doi.org/10.1109/ACCESS.2020.3016680

[12] Ni, J., Lin, X., Shen, X.S. (2019). Toward edge-assisted Internet of Things: From security and efficiency perspectives. IEEE Network, 33(2): 50-57. https://doi.org/10.1109/MNET.2019.1800229

[13] Gadde, S., Rao, G.S., Veesam, V.S., Yarlagadda, M., Patibandla, R.S.M.L. (2023). Secure data sharing in cloud computing: A comprehensive survey of two-factor authentication and cryptographic solutions. Ingénierie des Systèmes d'Information, 28(6): 1467-1477. https://doi.org/10.18280/isi.280604

[14] Cheng, G., Guo, D., Luo, L., Xia, J., Gu, S. (2021). LOFS: A lightweight online file storage strategy for effective data deduplication at network edge. IEEE Transactions on Parallel and Distributed Systems, 33(10): 2263-2276. https://doi.org/10.1109/TPDS.2021.3133098

[15] Song, M., Hua, Z., Zheng, Y., Xiang, T., Jia, X. (2023). FCDedup: A two-level deduplication system for encrypted data in fog computing. IEEE Transactions on Parallel and Distributed Systems, 34(10): 2642-2656. https://doi.org/10.1109/TPDS.2023.3298684

[16] Ni, J., Zhang, K., Yu, Y., Lin, X., Shen, X.S. (2018). Providing task allocation and secure deduplication for mobile crowdsensing via fog computing. IEEE Transactions on Dependable and Secure Computing, 17(3): 581-594. https://doi.org/10.1109/TDSC.2018.2791432

[17] Jiang, S., Jiang, T., Wang, L. (2017). Secure and efficient cloud data deduplication with ownership management. IEEE Transactions on Services Computing, 13(6): 1152-1165. https://doi.org/10.1109/TSC.2017.2771280

[18] Li, H., Dong, M., Liao, X., Jin, H. (2015). Deduplication-based energy efficient storage system in cloud environment. The Computer Journal, 58(6): 1373-1383. https://doi.org/10.1093/comjnl/bxu122

[19] Yu, X., Bai, H., Yan, Z., Zhang, R. (2022). Veridedup: A verifiable cloud data deduplication scheme with integrity and duplication proof. IEEE Transactions on Dependable and Secure Computing, 20(1): 680-694. https://doi.org/10.1109/TDSC.2022.3141521

[20] Yang, X., Lu, R., Shao, J., Tang, X., Ghorbani, A.A. (2020). Achieving efficient secure deduplication with user-defined access control in cloud. IEEE Transactions on Dependable and Secure Computing, 19(1): 591-606. https://doi.org/10.1109/TDSC.2020.2987793

[21] He, Q., Bian, G., Zhang, W., Zhang, F., Duan, S., Wu, F. (2021). Research on routing strategy in cluster deduplication system. IEEE Access, 9: 135485-135495. https://doi.org/10.1109/ACCESS.2021.3116270

[22] Guo, S., Mao, X., Sun, M., Wang, S. (2023). Double sliding window chunking algorithm for data deduplication in ocean observation. IEEE Access, 11:

70470-70481. https://doi.org/10.1109/ACCESS.2023.3276785
[23] Yuan, H., Chen, X., Li, J., Jiang, T., Wang, J., Deng, R.H. (2019). Secure cloud data deduplication with efficient re-encryption. IEEE Transactions on Services Computing, 15(1): 442-456. https://doi.org/10.1109/TSC.2019.2948007
[24] Jin, X., Liu, H., Ye, C., Liao, X., Jin, H., Zhang, Y. (2023). Accelerating content-defined chunking for data deduplication based on speculative jump. IEEE Transactions on Parallel and Distributed Systems, 34(9): 2568-2579. https://doi.org/10.1109/TPDS.2023.3290770
[25] Chen, L., Li, J., Li, J. (2023). Toward forward and backward private dynamic searchable symmetric encryption supporting data deduplication and conjunctive queries. IEEE Internet of Things Journal, 10(19): 17408-17423. https://doi.org/10.1109/JIOT.2023.3274390
[26] Wang, Z., Gao, W., Yu, J., Shen, W., Hao, R. (2023). Lightweight secure deduplication based on data popularity. IEEE Systems Journal, 17(4): 5531-5542. https://doi.org/10.1109/JSYST.2023.3307883
[27] Yao, W., Hao, M., Hou, Y., Li, X. (2022). FASR: An efficient feature-aware deduplication method in distributed storage systems. IEEE Access, 10: 15311-15321. https://doi.org/10.1109/ACCESS.2022.3147545
[28] Wu, J., Hua, Y., Zuo, P., Sun, Y. (2018). Improving restore performance in deduplication systems via a cost-efficient rewriting scheme. IEEE Transactions on Parallel and Distributed Systems, 30(1): 119-132. https://doi.org/10.1109/TPDS.2018.2852642
[29] Guim, F., Metsch, T., Moustafa, H., Verrall, T., Carrera, D., Cadenelli, N., Chen, J., Doria, D., Ghadie, C., Prats, R.G. (2021). Autonomous lifecycle management for resource-efficient workload orchestration for green edge computing. IEEE Transactions on Green Communications and Networking, 6(1): 571-582. https://doi.org/10.1109/TGCN.2021.3127531
[30] Shi, W., Zhang, J., Zhang, R. (2019). Share-based edge computing paradigm with mobile-to-wired offloading computing. IEEE Communications Letters, 23(11): 1953-1957. https://doi.org/10.1109/LCOMM.2019.2934411
[31] Cui, G., He, Q., Li, B., Xia, X., Chen, F., Jin, H., Xiang, Y., Yang, Y. (2021). Efficient verification of edge data integrity in edge computing environment. IEEE Transactions on Services Computing, 15(6): 3233-3244. https://doi.org/10.1109/TSC.2021.3090173
[32] Luo, R., Jin, H., He, Q., Wu, S., Xia, X. (2022). Cost-effective edge server network design in mobile edge computing environment. IEEE Transactions on Sustainable Computing, 7(4): 839-850. https://doi.org/10.1109/TSUSC.2022.3178661
[33] Kiani, A.Y., Hassan, S.A., Su, B., Pervaiz, H., Ni, Q. (2020). Minimizing the transaction time difference for NOMA-based mobile edge computing. IEEE Communications Letters, 24(4): 853-857. https://doi.org/10.1109/LCOMM.2020.2966442
[34] Douch, S., Abid, M.R., Zine-Dine, K., Bouzidi, D., Benhaddou, D. (2022). Edge computing technology enablers: A systematic lecture study. IEEE Access, 10: 69264-69302. https://doi.org/10.1109/ACCESS.2022.3183634
[35] Valadares, D.C.G., De Oliveira Filho, T.B., Meneses, T.F., Santos, D.F., Perkusich, A. (2022). Automating the deployment of artificial intelligence services in multiaccess edge computing scenarios. IEEE Access, 10: 100736-100745. https://doi.org/10.1109/ACCESS.2022.3208118
[36] Jamil, M.N., Hossain, M.S., Islam, R.U., Andersson, K. (2023). Workload orchestration in multi-access edge computing using belief rule-based approach. IEEE Access, 11: 118002-118023. https://doi.org/10.1109/ACCESS.2023.3326244
[37] Alfakih, T., Hassan, M.M., Gumaei, A., Savaglio, C., Fortino, G. (2020). Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA. IEEE Access, 8: 54074-54084. https://doi.org/10.1109/ACCESS.2020.2981434
[38] Abdalhameed, A.A., Kadhim, A.I. (2024). Data recovery in cloud data storage. Ingénierie des Systèmes d'Information, 29(5): 1959-1966. https://doi.org/10.18280/isi.290527