Byte-Level Deduplication

Suvarna L. Ghogare
Pravara Sir Visvesvaraya institute of Technology,Nashik

***Abstract*—Byte-level deduplication is an advanced data optimization technique used to eliminate redundant data at the finest granularity by identifying and removing duplicate byte sequences. Unlike file-level or block-level deduplication, this method analyzes data at the byte level, enabling higher storage efficiency by detecting even the smallest repetitions within and across files. The process typically involves breaking data into variable-length segments, generating unique fingerprints using hashing algorithms, and storing only unique byte patterns while maintaining references to duplicates.This approach significantly reduces storage requirements, improves bandwidth utilization during data transfer, and enhances backup and archival performance. Byte-level deduplication is widely applied in cloud storage systems, distributed file systems, and backup solutions where large volumes of repetitive data are common. However, the technique introduces computational overhead and complexity in indexing and retrieval due to its fine granularity. Despite these challenges, byte-level deduplication remains a powerful solution for efficient data management in modern large-scale storage environments.**

***Index Terms*—Byte-Level Deduplication, Data Deduplication, Data Redundancy, Variable-Length Segmentation, Data Fingerprinting, Hashing Algorithms, Storage Optimization, Cloud Storage, Distributed File Systems, Backup and Archival Systems, Bandwidth Optimization, Storage Efficiency.**

I. INTRODUCTION

In today's digital era, the volume of data generated and stored is increasing at an exponential rate due to applications such as cloud computing, multimedia storage, and big data analytics. This rapid growth creates challenges in terms of storage capacity, cost, and data management. A significant portion of stored data is often redundant, as identical or similar data appears multiple times across files and systems.To address this issue, data deduplication techniques are used to eliminate duplicate data and improve storage efficiency. Among these techniques, byte-level deduplication is the most fine-grained approach. It operates at the smallest unit of data —the byte— allowing the system to identify even the smallest repetitions within and across files.byte-level deduplication provides higher accuracy and better storage optimization by detecting partial similarities between data segments. This makes it especially useful in environments where data changes frequently but only in small portions, such as backup systems, cloud storage, and distributed file systems.Although byte-level deduplication offers significant advantages in reducing storage space and improving efficiency, it also introduces challenges such as increased computational complexity and the need for efficient indexing mechanisms. Despite these challenges, it remains a powerful technique for modern data storage systems.

II. RELATED WORK

Recent studies have explored deduplication in cloud computing environments, where distributed and scalable deduplication is necessary.Data deduplication has been extensively researched as and efficient technique to reduce storage overhead and optimize data transfer, particularly in cloud storage systems, backup services, and enterprise data centers.The concept of eliminating redundant data blocks has evolved over the years from file-level deduplication to the more efficient block-level and content-based chunking approaches.Review of literature focused on some previously exist secure data auditing and de-duplication techniques by considering their advantages, disadvantages and features.First we focused on the techniques having details about data deduplicationtechnique.[1]S.Ghogare1 Prof.S.k.sonkar,"secure data auditing and deduplicating data with multiuser cloud environment",vol-3 issue-4 2017.In this two secure approaches namely, SecCloud

and SecCloud+ for secured auditing and de-duplication in cloud server.On loud server, there may have chances of duplicate data which affects on the management of data. The proposed system check for duplicate files on cloud server before uploading file on cloud by using file tag generation and file block uploading approaches. For verification of data `auditor' is introduced which verifies the data integrity. There are several features introduced in proposed system such as, data encryption, data duplication check and periodic verification of data. For better efficiency map-reduce functionality is used. As a part of contribution,system is design and developed with multiuser environment. User can share files with the other users which is uploaded by them to cloud server. With an experimental set up proposed system proves an efficiency of system in terms of Data uploading, tag generation and deduplication check time of execution.[2] Suvarna Ghogare.Information Technology Department, Sir Visvesvaraya Institute of Technology, Nashik."FILE LEVEL DEDUPLICATION USING fdupes".

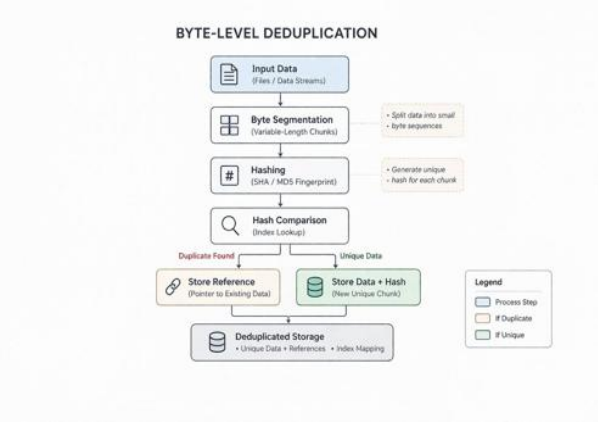
This proposed secure approaches file level de-duplication for secured de-duplication at storage. There may have chances of duplicate data which effects on the memory requirement, management of data. The proposed system checks for duplicate files at storage by using approaches file level deduplication. There are several features introduced in proposed system such as, data duplication check by fdupes. For better efficiency Hash based algorithm is used. File-level deduplication reduce redundancy. It works by comparing each new file against a database of existing files and only storing a single instance of a unique file, updating the index accordingly.[3]fdupes is a program written by Adrián López to scan directories for duplicate files, with options to list, delete or replace the files with hardlinks pointing to the duplicate. It first compares file sizes,partial MD5 signatures, full MD5 signatures, and then performs a byte-by- byte comparison for verification.fdupes is written in C and is released under the MIT License.J. Li, X. Chen, M. Li, et al [5], proposed Dekey. It is an efficient and reliable convergent key management scheme for secure deduplication. It applies deduplication among convergent keys and distributes convergent key shares across multiple key servers, while preserving semantic security of convergent keys and confidentiality of outsourced data. They implemented Dekey using the Ramp secret sharing scheme and demonstrate that it incurs small encoding/decoding overhead compared to the network transmission overhead in the regular upload/download operations.S. Halevi, et al [6], put forward the notion of proof-of-ownership, by which a client can prove to a server that it has a copy of a file without actually sending it. This allows to counter attacks on file-deduplication systems where the attacker obtains a "short summary" of the file and uses it to fool the server into thinking that the attacker owns the entire file. We gave three definitions for security in this setting and three matching protocols, the last of which is very practical.R. Burns, Ateniese, et al. [7]discussed PDP solution as far as data integrity checking by auditor to the cloud in concern.Uploaded file having n-blocks are verified.Packet Data Protocol is the protocol that verifies that the cloud storage return a file consisting „n"-blocks.In [7], author Burnus and Ateniese uses Packet Data Protocol model for

distant data checking. End user.

III. PROBLEM DEFINITION

Byte-level deduplication identifying and removing duplicate data at the smallest unit.

IV. PROPOSED SYSTEM



Working of Each Block

1. Input Data

- The system receives data in the form of files or data streams.
- This data may contain duplicate or similar content.

2. Byte Segmentation
- The input data is divided into small byte-level chunks.
- Usually uses variable-length chunking to detect even small changes.
- Helps in identifying fine-grained similarities.

3. Hashing(Fingerprint Generation)
- Each chunk is processed using a hash function (e.g., MD5, SHA).
- A unique hash value (fingerprint) is generated for every chunk.
- This fingerprint represents the content of that chunk.

4. Hash Comparison (Index Lookup)
- The generated hash is compared with stored hashes in an index table.
- If the hash already exists → duplicate data detected.
- If not → data is unique.

5. Duplicate Found → Store Reference
- Instead of storing the duplicate data again,
- Only a reference (pointer) to the existing data is stored.
- Saves storage space.

6. Unique Data → Store Data + Hash
- If the data is new,
- The system stores:
- Actual data chunk
- Its corresponding hash value
- Updates the index.

7. Deduplicated Storage
- Final storage contains:
- Only unique data chunks
- References to duplicates
- Maintains mapping: Hash → Data Location
- Ensures efficient storage and retrieval.

V. METHODOLOGY

The proposed Byte-Level Deduplication system follows a systematic approach to identify and eliminate redundant data at the finest granularity. The methodology consists of the following steps:

1. Data Collection
- Input data is collected in the form of files or data streams.
- This data may contain duplicate or partially similar content.

2. Data Preprocessing
- The input data is prepared for processing.
- Noise or unnecessary formatting (if any) is handled.

3. Byte-Level Segmentation
- The data is divided into small chunks at the byte level.
- Variable-length chunking is used to improve detection of small changes.

4. Hash Generation
- Each chunk is processed using a hash function (such as MD5 or SHA).
- A unique fingerprint is generated for every chunk.

5. Indexing and Comparison
- Generated hash values are compared with an existing index table.
- The index maintains mapping between hash values and stored data.

6. Deduplication Process
- If hash matches an existing entry → duplicate detected
- → Store only a reference (pointer)
- If no match → unique data
- → Store chunk and update index

7. Storage Management
- Only unique data chunks are stored in memory/storage.
- Duplicate chunks are represented using references.

8. Data Retrieval
- When required, original data is reconstructed using stored chunks and references.
- Ensures data integrity and correctness.

9. Performance Evaluation

- Evaluate system based on:
- Storage efficiency
- Deduplication ratio
- Processing time

VI. ALGORITHM

Algorithm: Byte-Level Deduplication Input: Data file / data stream

Output: Deduplicated storage with unique data and references

Steps:

1. Start
2. Read Input Data
- Accept file or data stream.
3. Perform Byte Segmentation
- Divide data into small chunks (variable-length).
4. For each chunk, do:

a. Generate hash value using hash function (MD5/SHA)

b. Search hash in index table

5. If hash exists (Duplicate Found):
- Store only reference (pointer) to existing chunk
6. Else (Unique Data):
- Store chunk in storage
- Store hash in index table
7. Repeat steps for all chunks
8. Maintain Mapping:
- Hash → Storage Location
9. End

Step	Algorithm Operation	Description
1	Input Data	Read the input file/data stream as a sequence of bytes.
2	Initialize Parameters	Set minimum chunk size, maximum chunk size, and rolling-hash parameters.
3	Byte Segmentation	Scan the byte stream and determine variable-length chunk boundaries using a rolling hash.
4	Generate Chunk	Extract the bytes between two consecutive boundaries as a chunk.
5	Calculate Hash	Calculate SHA-256 for each generated chunk.
6	Hash Index Lookup	Search the SHA-256 value in the deduplication index.
7	Duplicate Check	If the hash exists, treat the chunk as a duplicate.
8	Store Reference	Instead of storing the duplicate chunk, store a reference/pointer to the existing chunk.
9	Unique Chunk	If the hash does not exist, store the chunk and its SHA-256 hash.
10	Update Index	Add the new hash and its storage location to the index.
11	Continue	Repeat the process until the complete input file/data stream is processed.
12	Output	Produce deduplicated storage containing unique chunks, references, and index mappings.

VII. MATHEMATICAL MODEL

1. System Definition

Let:

$D=\{d1,d2,d3,...,dn\}$ → Set of input data files

$C=\{c1,c2,c3,...,cm\}$ → Set of byte-level chunks

$H(ci)$ → Hash function applied to chunk ci

S → Storage space required

2. Input

• Data set D containing files with possible redundant data.

3. Process

Step 1: Chunking

Each file is divided into chunks:

$di \rightarrow \{c1,c2,...,ck\}$

Step 2: Hashing

Each chunk is converted into a hash value:

$hi=H(ci)$

Step 3: Deduplication Condition

• If:

$hi=hj(i \square =j)$

Then:

chunk)ci=cj(duplicate chunk)

Else:

chunk)ci $\square$ =cj (unique chunk)

4. Storage Model
Total storage without deduplication:
Soriginal=i=1∑msize(ci)
Storage with deduplication:
sizeSdedup=i ∈ unique∑ size(ci)+reference size

5. Deduplication Ratio
RatioDeduplication Ratio=SdedupSoriginal
6 Output
• Unique chunks stored
• Duplicate chunks replaced by references
7. Objective Function
Minimize storage:
min(Sdedup)

VIII. EXPERIMENTAL SETUP

1. Hardware Requirements
Processor: Minimum Intel i3 / equivalent
RAM: 4 GB or higher
Storage: 500 GB HDD / SSD
System: Personal Computer / Laptop

2. Software Requirements
• Operating System: Windows / Linux / Ubuntu
• Programming Language: Python / Java
• Tools: IDE (VS Code / Eclipse / PyCharm)
• Libraries: Hashing libraries (MD5, SHA)

3. Dataset Description
• Input data consists of:
• Text files
• Documents
• Backup data with repetitive content
• Dataset size varies (small to large files) to test performance.

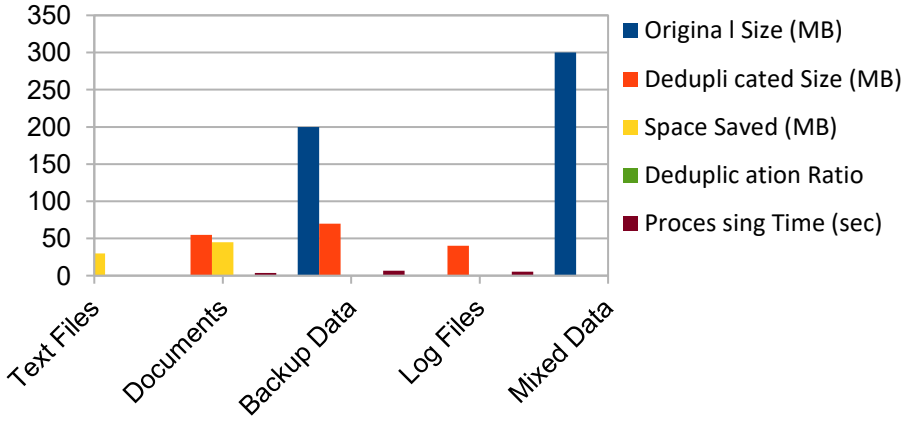
4. Implementation Setup
• Data is divided into byte-level chunks.
• Hash function is applied to each chunk.
• Hash values are stored in an index table.
• Duplicate chunks are replaced with references.

5. Evaluation Parameters
The system is evaluated based on:
Storage Reduction
Deduplication Ratio
Processing Time
Memory Usage

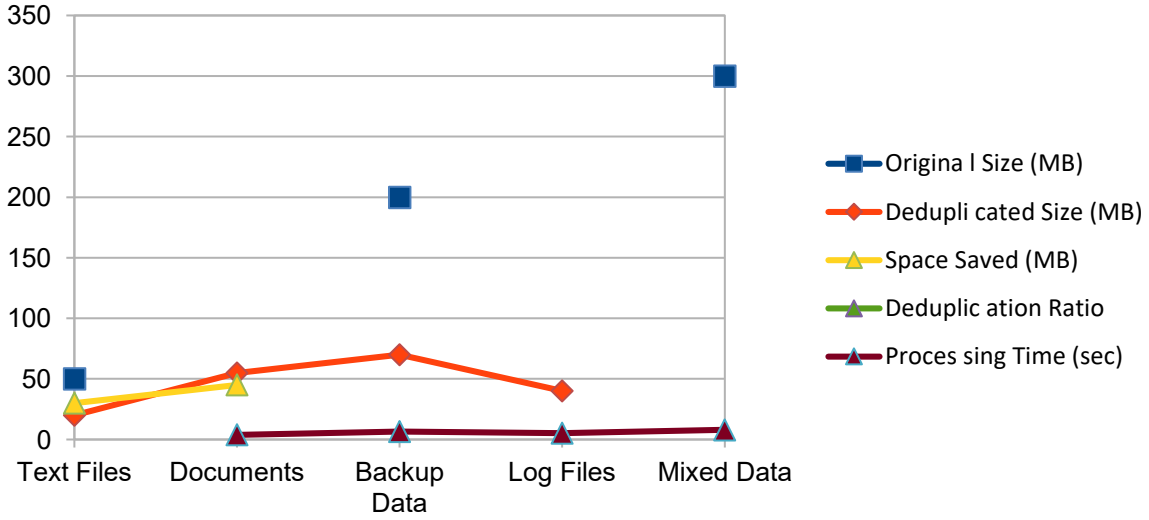
6. Experimental Procedure
1. Load input dataset
2. Apply byte-level segmentation
3.Generate hash values
4.Compare with index table
5.Store unique data and references
6.Measure performance metrics
7. Tools for Measurement
• Time calculation (execution time)
• Memory usage monitoring tools
• File size comparison (before vs after deduplication)

IX. RESULT TABLES AND DISCUSSION

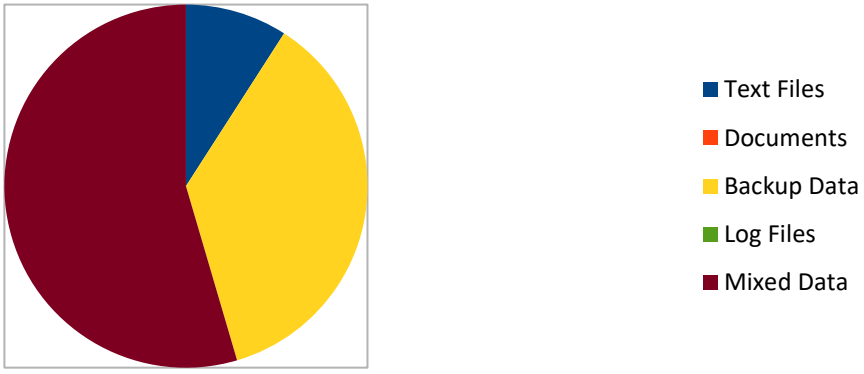
Dataset	Original Size (MB)	Deduplicated Size (MB)	Space Saved (MB)	Deduplication Ratio	Processing Time (sec)
Text Files	50	20	30	2.5 : 1	2.1
Documents	100	55	45	1.82 : 1	3.8
Backup Data	200	70	130	2.85 : 1	6.5
Log Files	150	40	110	3.75 : 1	5.2
Mixed Data	300	120	180	2.5 : 1	8.0



column chart: deduplication at byte level on basis of size and duplicate data present



line chart: deduplication at byte level on basis of size and duplicate data present



pie chart: deduplication at byte level on basis of size and duplicate data present

screeenshot of reading:

X. CONCLUSIONS

The Byte-Level Deduplication system successfully demonstrates an effective approach to reducing data redundancy by identifying duplicate data at the smallest unit level. By using byte-level segmentation and hashing techniques, the system ensures that only unique data is stored while duplicates are replaced with references, leading to significant storage savings.The experimental results show that the system achieves a high deduplication ratio, especially for datasets with repetitive content such as logs and backup files. Although the method introduces additional computational overhead due to fine-grained processing, the overall benefits in storage optimization and bandwidth efficiency outweigh the limitations.In conclusion, byte-level deduplication proves to be a powerful and efficient technique for modern data storage systems, particularly in cloud computing and backup environments where data redundancy is high.

XI. ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have supported and guided me in the successful completion of this project on Byte-Level Deduplication.I am especially thankful to my co-worker for their valuable guidance, continuous support, and encouragement throughout the development of this project. Their insights and suggestions greatly contributed to improving the quality of this work.I would also like to thank the faculty members of the department for providing the necessary resources and knowledge required for this project.Finally, I express my heartfelt thanks to my friends and family for their constant motivation and support, which helped me complete this project successfully.

REFERENCES

[1] S. Ghogare1, Prof.S.k.sonkar ," secure data auditing and deduplicating data with multiuser

cloud environment", vol-3 issue-4 2017.

[2] Suvarna Ghogare. Information Technology Department,Sir Visvesvaraya Institute of Technology, Nashik." FILE LEVEL DEDUPLICATION USING fdupes"

[3] fdupes is a program written by Adrián López to scan directories for duplicate files, with options to list, delete or replace the files with hardlinks pointing to the duplicate. It first compares file sizes, partial MD5 signatures,full MD5 signatures, and then performs a byte-by- byte comparison for verification. fdupes is written in C and is released under the MIT License.

[4] Jingwei Li, Jin Li, Dongqing Xie, and Zhang Cai "Secure Auditing and Deduplicating Data in Cloud" VOL.65, NO. 8, AUGUST 2016

[5] J. Li, X. Chen, M. Li, J. Li, P. Lee, and W. Lou,"Secure deduplication with efficient and reliable convergent key management," IEEE Trans.Parallel Distrib.Syst., vol. 25, no. 6, pp. 1615–1625, Jun. 2014.

[6] S. Halevi, D. Harnik, B. Pinkas, and A.Shulman-Peleg,"Proofs of ownership in remote storage systems," in Proc.18th ACM Conf.Comput. Commun. Secur., 2011, pp. 491–500

[7] G. Ateniese, R. Burns, R. Curtmola, J. Herring,O. Khan, L.Kissner, Z. Peterson,and D. Song,"Remote data checking using provable data possession," ACM Trans. Inform. Syst. Secur.,vol. 14, no. 1, pp. 1–34, 2011.

[8] G. Ateniese, R. Di Pietro, L. V. Mancini, and G.Tsudik,"Scalable and efficient provable data possession," in Proc. 4th Int. Conf. Secur.Privacy Commun.Netow., 2008, pp. 1–10.

[9] Erway, A. Kupc € u, C. Papamanthou, and R.Tamassia, € "Dynamic provable data possession," in Proc. 16th ACM Conf. Comput.Commun. Secur., 2009, pp. 213–222.

[10] H. Wang, "Proxy provable data possession in public clouds," IEEE Trans. Serv. Comput., vol. 6, no. 4, pp. 551– 559, Oct.-Dec. 2013