Ripple: Localized Content-Centric Deduplication at the Edge

Mamidi Chinmayi, department of CSE, GNITC-HYD, 22-5H3, 22wj1a05h3@gniindia.org
Madupalli Venu, department of CSE, GNITC-HYD, 22-5H0, 22wj1a05h0@gniindia.org
Mohammad Azhar, department of CSE, GNITC-HYD, 23-521, 23wj5a0521@gniindia.org
Karinga Shankar, Assistant Professor, department of CSE, GNITC-HYD, shankar.csegnitc@gniindia.org

Abstract – Edge computing has become an important foundation for many latency-sensitive applications because it enables faster data access and reduces the amount of traffic sent to distant cloud servers. The rapid growth of data generated at the edge places significant pressure on the limited storage capacity of edge servers, making efficient resource management a major challenge. While existing solutions attempt to improve storage efficiency through techniques such as optimized data placement, partitioning, and data sharing, they often overlook the issue of data redundancy. Multiple copies of similar data may be stored across edge nodes, leading to unnecessary storage consumption and increased network overhead. Data deduplication at the edge can help address this problem by ensuring that only unique data is stored. Many current deduplication approaches rely on centralized coordination, which is not always suitable for real-world edge environments where a central controller may be unavailable or may introduce a single point of failure.

To overcome these limitations, this work proposes Ripple, a decentralized edge-based data deduplication framework that allows each edge server to perform deduplication independently. Ripple maintains a local data index on every edge node, enabling servers to detect duplicate data chunks, eliminate redundant copies while maintaining low latency, and preserve data availability through controlled replication strategies. The system is implemented using a full-stack Java architecture that integrates a RESTful service layer, a distributed storage layer, and a lightweight coordination protocol for exchanging deduplication metadata among edge nodes. Experimental evaluations using trace-driven workloads on a real-world edge testbed demonstrate the effectiveness of the proposed approach. The results show that Ripple reduces average data retrieval latency by approximately 60% and improves the deduplication ratio by up to 17% compared with existing edge and edge-assisted cloud deduplication techniques. These findings indicate that decentralized deduplication can significantly enhance storage efficiency while maintaining system reliability and quality of service in next-generation edge computing environments.

Key Words: Edge computing, data redundancy, data retrieval latency, data deduplication, data index.

1.INTRODUCTION

Edge computing brings computation and storage resources closer to end users, enabling faster data processing and reducing the amount of traffic that needs to be transmitted to distant cloud data centres. This is particularly important for modern applications such as the Internet of Things (IoT), video analytics, and smart city services, which generate large volumes of data that must be processed with minimal delay. As the amount of data produced at the edge continues to grow, the limited storage capacity of edge servers becomes a major challenge. Managing storage resources efficiently is therefore essential to ensure smooth system performance and reliable service delivery. Many existing techniques attempt to improve storage utilization through methods such as data placement optimization, partitioning, and data sharing among edge nodes. Although these approaches help organize how data is stored, they often overlook the problem of data redundancy. In many edge environments, the same or very similar data may be stored on multiple servers, resulting in unnecessary storage consumption and additional network overhead during data transmission and retrieval. Data deduplication, which eliminates duplicate copies and keeps only a single instance of repeated data, has been widely used in cloud and large-scale storage systems to address this issue. And, most traditional deduplication solutions rely on centralized controllers or powerful data centres to coordinate the process. Such centralized designs can create bottlenecks and introduce a single point of failure, making them less suitable for dynamic and distributed edge environments.

This study introduces Ripple, a decentralized framework designed for data deduplication in edge computing systems. In Ripple, each edge server maintains its own local index that helps identify duplicate data chunks stored across neighbouring nodes. Instead of depending on a central controller, edge servers perform deduplication independently and communicate with other nodes only when necessary. This approach reduces coordination overhead while maintaining efficient storage usage. The design focuses on three main objectives: reducing storage consumption and network overhead through effective deduplication, preserving low-latency data access by considering retrieval constraints, and maintaining data availability and reliability after redundant data copies are removed. Experimental evaluations indicate that decentralized edge deduplication can significantly improve the deduplication ratio and reduce data retrieval latency, highlighting its potential as an effective solution for modern edge computing systems.

2. LITERATURE REVIEW

Edge data deduplication has become an important technique for addressing the growing challenge of large-scale data generation and limited storage resources in mobile edge computing (MEC) environments. As edge devices and applications continue to produce massive volumes of data, efficient storage management and bandwidth utilization have become critical concerns. Earlier research introduced Edge Data Deduplication (EDDE) as an optimization problem aimed at maximizing the deduplication ratio while maintaining strict latency requirements. These studies demonstrated that the problem is NP-hard and proposed both optimal (EDDE-O) and approximation-based (EDDE-A) algorithms that enable collaboration among edge servers to achieve effective redundancy removal.

Later studies expanded this idea by introducing balanced deduplication approaches that consider not only the deduplication rate but also storage benefits and load balancing across edge storage systems. The goal of these methods is to prevent certain servers from becoming overloaded during the deduplication process. Similar to earlier models, these approaches also treat the problem as NP-hard and provide scalable algorithms that have been validated using real-world trace data. In addition, researchers have explored robust optimization techniques designed to handle dynamic edge environments. These approaches incorporate uncertainties such as user mobility, fluctuating data demand, and potential edge server failures. Algorithms such as uEDDE-C and uEDDE-A have shown promising results by reducing storage costs and improving data retrieval latency while maintaining system reliability.

Beyond centralized or cluster-based solutions, decentralized deduplication frameworks have also been proposed. In such systems, each edge server maintains a local index that allows it to detect duplicate data independently. Ripple is one example of this approach, where servers coordinate only when necessary, reducing reliance on centralized control while maintaining efficient deduplication and data availability. Experimental results from such systems demonstrate improved deduplication ratios and lower data access latency compared to many traditional centralized techniques.

Other studies have focused on collaborative and lightweight indexing mechanisms to further improve efficiency. EF-Dedup groups edge nodes into clusters and uses a distributed key–value index to optimize overall storage and network costs, achieving higher throughput compared to cloud-only deduplication approaches. Similarly, the LOFS system applies Bloom filters and locality-sensitive hashing to approximate global deduplication with fast lookup times and balanced workloads across nodes.

Performance-oriented solutions is a significant amount of research has focused on secure data deduplication for edge and IoT environments. These studies adapt cryptographic techniques originally designed for cloud storage, including hybrid client–server deduplication schemes based on homomorphic encryption. Security-aware multi-level encryption techniques have also been proposed to balance privacy protection with deduplication efficiency. And, several MEC-specific protocols support secure storage, data sharing, and deduplication across centralized, semi-distributed, and fully distributed edge architectures.

3. RELATED WORK

Edge data deduplication has progressed significantly over the years, evolving from simple redundancy removal techniques to advanced systems that incorporate optimization strategies and security mechanisms. These developments are particularly important in environments such as Mobile Edge Computing (MEC), Internet of Things (IoT) networks, and integrated cloud–edge infrastructures, where large volumes of data must be processed and stored efficiently.

Early research in this area focused on developing formal models for deduplication in edge environments. One of the foundational approaches introduced Edge Data Deduplication (EDDE) as an NP-hard optimization problem aimed at maximizing the deduplication ratio while satisfying strict latency requirements. To address this problem, researchers proposed both an exact integer programming-based solution (EDDE-O) and a logarithmic approximation algorithm (EDDE-A) suitable for large-scale systems. Subsequent work extended these ideas by introducing Balanced Edge Data Deduplication (BEDD), which considers not only deduplication efficiency but also storage benefits and load balancing among edge servers. These methods demonstrated improved system performance when evaluated using real-world trace data. More recent studies have explored robust optimization techniques that account for uncertainties such as fluctuating user demand and potential edge server failures. Algorithms such as uEDDE-C and uEDDE-A have shown the ability to reduce storage costs and data retrieval latency in highly dynamic MEC environments. In addition, reinforcement learning methods have been investigated to support adaptive and online deduplication decisions in edge storage systems.

Another important direction in this field involves collaborative and decentralized deduplication strategies. Collaborative approaches such as EF-Dedup organize edge nodes into clusters and maintain a distributed key–value index to jointly optimize storage and network costs. This method has demonstrated higher throughput and lower operational costs compared to traditional cloud-based deduplication solutions. In contrast, fully decentralized frameworks aim to remove reliance on centralized coordination. For example, Ripple introduces a design in which each edge server maintains a local index to detect duplicate data independently. By allowing nodes to communicate only, when necessary, this approach reduces coordination overhead while maintaining system reliability and low latency. Experimental studies report improvements of up to 16.8% in deduplication ratio and approximately 60% reduction in latency compared to existing techniques. Other systems, such as MEAN, focus on improving reliability in environments with unstable edge resources by combining similarity-aware clustering with replication mechanisms. Additionally, generalized deduplication techniques have been applied to approximate edge analytics through systems like GLEAN, which achieve faster processing and storage efficiency while maintaining acceptable accuracy levels.

Security and privacy have also become important considerations in edge-based deduplication. Secure deduplication approaches aim to protect sensitive data while still enabling redundancy elimination. Hybrid frameworks combine client-side and server-side deduplication techniques using cryptographic methods such as homomorphic encryption to balance privacy protection with bandwidth efficiency. Multi-level security-aware deduplication schemes further allow flexible trade-offs between deduplication performance and resistance to attacks such as frequency analysis. For applications such as MEC content delivery and crowdsensing, specialized protocols have been designed to support secure storage, key management, and duplicate detection across centralized, semi-distributed, and distributed edge systems. Some IoT-oriented frameworks integrate blockchain technology with edge computing to enhance trust and traceability. Mechanisms such as RARE help defend against side-channel attacks, while structures like label trees and smart contracts support efficient one-to-many data sharing and resistance to collusion. And, cryptographic techniques based on convergent encryption and elliptic curve cryptography have been applied in integrated cloud–fog–edge environments to support secure block-level deduplication.

Several survey papers have also contributed to understanding the broader landscape of data deduplication. Comprehensive reviews published in IEEE and other venues categorize deduplication systems based on their workflow stages, including chunking, fingerprinting, indexing, garbage collection, and data restoration. These studies highlight key challenges such as index scalability, security risks, and performance optimization. More recent surveys explore distributed deduplication in big data environments, focusing on aspects such as data routing, multi-node index management, system reliability, and security mechanisms. And, systematic reviews examine deduplication in cloud storage and multimedia data, as well as redundancy elimination techniques for IoT-generated big data, with particular emphasis on encrypted deduplication and blockchain-enabled solutions.

4. PROPOSED METHODOLOGY

To improve storage efficiency and reduce data retrieval latency in edge computing environments. Unlike traditional deduplication systems that rely on centralized coordination, the Ripple framework adopts a decentralized approach in which each edge server independently performs deduplication. In this design, every edge node maintains a local data index and communicates with neighbouring nodes only, when necessary, through lightweight coordination mechanisms.

In the proposed framework, data generated by users or IoT devices is first directed to the nearest edge server for processing. Before storing the data, the system applies content-defined chunking and hashing techniques to divide the data into smaller segments and generate unique fingerprints for each segment. These fingerprints are then compared with entries in the local index to determine whether the data segment already exists. If a potential duplicate is identified, the edge server consults its local index and may exchange a small amount of metadata with nearby nodes to confirm duplication without transferring the full data content. This process minimizes unnecessary network communication while ensuring accurate duplicate detection.

The decentralized nature of Ripple reduces the dependence on a central controller, thereby improving system scalability and fault tolerance. At the same time, the framework is designed to preserve low-latency data access by ensuring that frequently requested or unique data remains available at the edge. To maintain reliability and balanced resource usage, the system also employs selective replication strategies, where important or popular data may be stored on multiple edge nodes.

Ripple establishes a local indexing mechanism that allows edge servers to independently identify and eliminate redundant data while still cooperating with other nodes when required. The framework therefore supports efficient duplicate detection, maintains low data retrieval latency, and ensures data availability even after redundant copies are removed.

The effectiveness of the proposed approach is validated through trace-driven experiments conducted on a real-world edge testbed. The experimental results show that Ripple significantly improves system performance compared with existing techniques. In particular, the framework achieves approximately a 60.42% reduction in average data retrieval latency and improves the deduplication ratio by up to 16.79%. These demonstrate that the proposed decentralized approach provides a practical and scalable solution for efficient data management in modern edge computing environment**s.**

5. DRDDEA ALGORITHM

We introduce the DRDDEA algorithm to eliminate redundant data across edge nodes.

The Decentralized Redundant Data Detection and Elimination Algorithm (DRDDEA) is designed to optimize data management in distributed and decentralized environments such as wireless sensor networks (WSNs), Internet of Things (IoT) systems, and blockchain-based storage networks. In these environments, multiple nodes often collect and transmit similar or identical data due to overlapping sensing areas or repeated operations. Sending all redundant data across the network can lead to unnecessary bandwidth consumption, higher latency, and increased energy usage, particularly in resource-constrained systems. DRDDEA addresses this issue by identifying and eliminating redundant data in a decentralized manner, thereby improving overall system efficiency.

Unlike traditional centralized projects that depend on a single server to process and manage data, DRDDEA allows individual nodes or groups of nodes to analyse their locally generated data and compare it with data from neighbouring nodes. This decentralized processing distributes the computational workload across the network, reducing reliance on a single point of control and improving system scalability and fault tolerance. To detect redundant information, nodes apply lightweight comparison techniques based on data signatures such as hash values, timestamps, or content-based identifiers. When duplicate or highly similar data is detected, the algorithm ensures that only the unique data is transmitted or stored, while redundant copies are either discarded or compressed locally.

Maintaining consistency and accuracy in a decentralized environment presents several challenges, especially when there is no centralized authority to verify data decisions. To address this issue, DRDDEA can incorporate collaborative verification mechanisms or lightweight consensus protocols that allow nodes to confirm redundancy detection without compromising data integrity.

The algorithm is also designed to adapt to dynamic network conditions, including node failures, device mobility, and fluctuations in data generation rates.

DRDDEA contributes to improved network performance by reducing unnecessary data transmissions, conserving energy resources, and enhancing storage utilization. These characteristics make the algorithm particularly suitable for large-scale decentralized systems where efficient data handling is essential and centralized coordination may be impractical or undesirable.

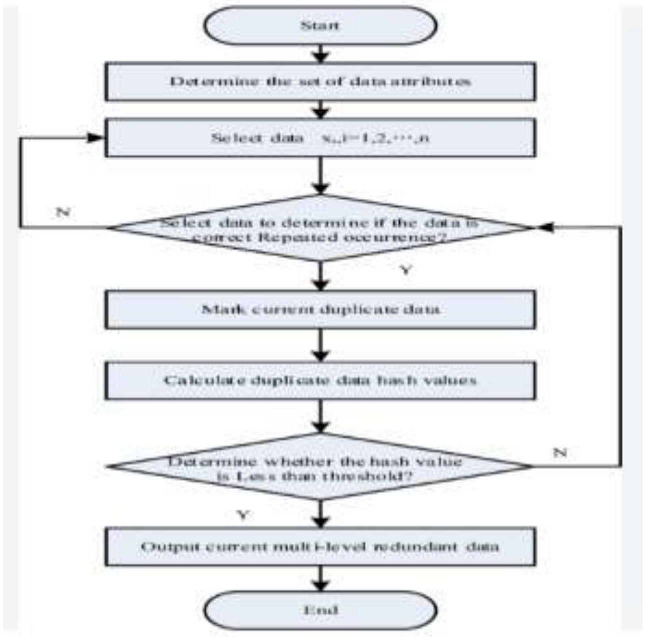
ALGORITHM PSUEDOCODE:

Algorithm 1: DRDDEA Algorithm

Input: Data chunks from edge nodes
Output: Unique data storage

```
1. Receive data chunk from node
2. Generate hash fingerprint
3. Compare fingerprint with local index
4. If duplicate detected:
       discard redundant chunk
   Else
```

store chunk in storage
5. Update index table
6. Broadcast metadata to neighboring nodes


Start
Determine the set of data attributes
Select data $x_i, i=1,2,\cdots,n$
Select data to determine if the data is correct Repeated occurrence?
N
Y
Mark current duplicate data
Calculate duplicate data hash values
Determine whether the hash value is Less than threshold?
N
Y
Output current multi-level redundant data
End

6. RESULTS AND DISCUSSION

We design a decentralized deduplication framework called Ripple for edge computing environments.
From the conducted experiments, we observed that the proposed Ripple framework performs efficiently in reducing redundant data and improving retrieval latency. The results indicate that decentralized deduplication at the edge can significantly enhance system performance when compared with traditional centralized approaches.
One of the key observations from the experiments is the improvement in the deduplication ratio achieved by Ripple. By allowing edge servers to independently identify and remove redundant data, the framework reduces unnecessary data storage across the network. The results show that Ripple can improve the deduplication ratio by up to approximately 17%, which leads to better utilization of limited edge storage resources. This improvement is particularly beneficial in environments where large volumes of data are generated continuously, such as IoT and real-time analytics systems.
Another important outcome is the reduction in data retrieval latency. The decentralized design enables data to remain closer to the users who request it, avoiding delays caused by centralized coordination or long-distance data transfers. Our experimental results indicate that Ripple can reduce average data retrieval latency by nearly **60%**, demonstrating its efficiency in decentralized edge environments. when compared with several existing edge deduplication and edge-assisted cloud solutions. This latency reduction improves overall user experience and supports latency-sensitive applications operating at the network edge.
Ripple, other edge deduplication systems such as MEAN and EF-Dedup also demonstrate significant improvements in storage and network efficiency. These systems employ techniques such as clustering and collaborative indexing to reduce overall storage-network costs. In some cases, they report reductions ranging from 20% to 60% in combined network and storage overhead. Such results highlight the growing importance of intelligent data management strategies in edge environments.
Security and robustness are also critical considerations in edge-based data deduplication systems. Recent secure deduplication approaches incorporate multi-level encryption and hybrid client–server techniques to protect sensitive information while still enabling redundancy elimination. These methods help defend against threats such as frequency analysis, side-channel attacks, and brute-force attempts. Furthermore, blockchain-based frameworks introduce additional capabilities such as traceability and secure cross-domain data sharing while maintaining efficient deduplication operations.

Robust optimization models have been proposed to address uncertainties commonly present in edge environments, including fluctuating user demand and potential edge server failures. Experimental evaluations show that these optimization-based approaches can maintain reliable performance even under dynamic conditions, The results confirm that decentralized deduplication frameworks such as Ripple provide a promising solution for improving storage utilization, reducing latency, and maintaining system reliability in modern edge computing infrastructures.

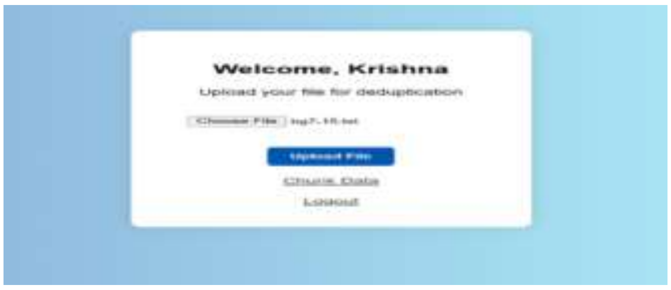
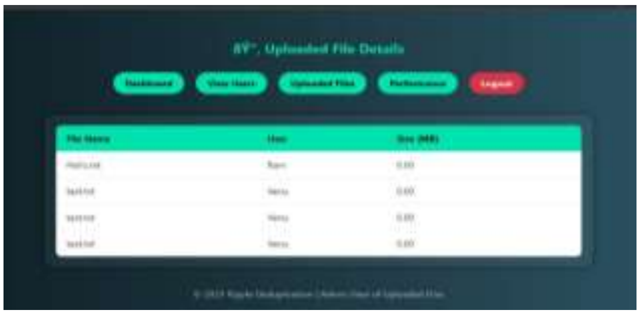


Fig1: Inserting File



File Name	User	Size (MB)
[illegible]	[illegible]	0.00
[illegible]	[illegible]	0.00
[illegible]	[illegible]	0.00
[illegible]	[illegible]	0.00

Fig2: Uploaded file Details

Fig3: Duplication metrics overview

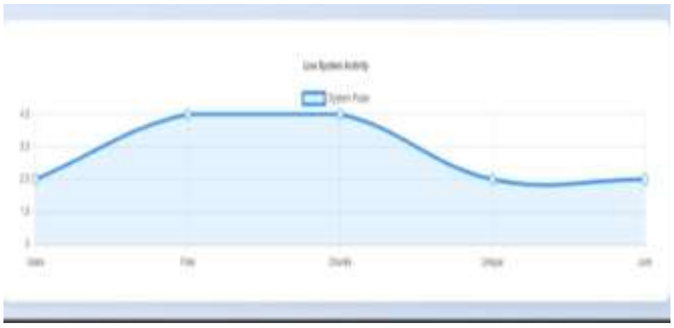
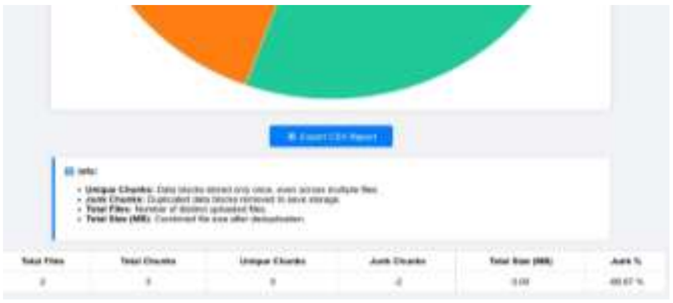


Fig4: Line system activity

Welcome, Krishna! Ripple Deduplication Dashboard

Fig5: Deduplication overview graph



Info:

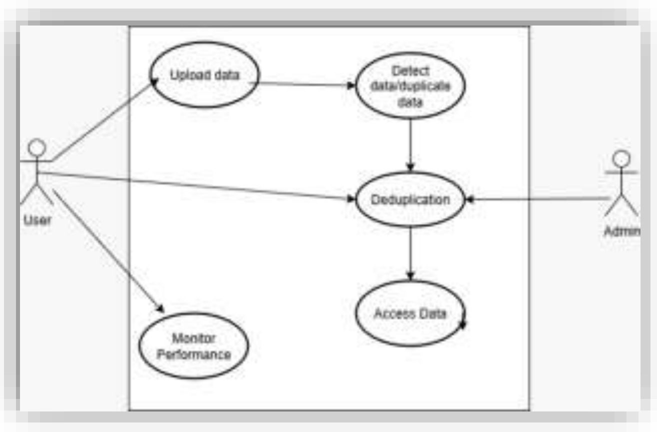
- **Unique Chunks:** Data blocks stored only once, even across multiple files.
- **Junk Chunks:** Duplicated data blocks removed to save storage.
- **Total Files:** Number of distinct uploaded files.
- **Total Size (MB):** Combined file size after deduplication.

Total Files	Total Chunks	Unique Chunks	Junk Chunks	Total Size (MB)	Junk %
2	3	1	2	0.00	66.67 %

Fig6: CSV report

Experimental evaluation demonstrates improved deduplication ratio and reduced latency.

SYSTEM ARCHITECHTURE:



The proposed Ripple framework consists of multiple edge nodes connected through a lightweight coordination network. Each edge node maintains a local data index and performs deduplication independently. When new data arrives, the node divides the data into chunks and generates fingerprints using hashing techniques. These fingerprints are compared with entries in the local index to detect duplicates. In cases where duplicate detection requires confirmation, nodes exchange lightweight metadata rather than transferring full data.

7. INDEX DESIGN

Ripple uses a tree-based indexing structure to efficiently manage and locate data stored across neighbouring edge servers. By examining this tree index, an edge server can quickly determine whether a specific data item (d) exists within a certain number of network hops and identify the node where the data is stored.

Primary Solution:

When an edge server (s_i) needs to release storage resources, it may remove locally stored data that is also available on neighbouring edge servers. However, the server must still be able to retrieve this data from its neighbours later without violating latency constraints. To achieve this, a straightforward solution is to construct a shortest-path tree that organizes neighbouring edge servers according to their network distance.

In this structure, the root node represents the edge server (s_i), and the paths from the root to other nodes represent the shortest communication paths between (s_i) and its neighbouring edge servers. Each leaf node corresponds to one neighbouring server and maintains an index of the data stored on that node. This hierarchical organization enables efficient identification of data locations within the edge storage system.

Challenges in Practical Implementation:

Although the Bloom tree approach satisfies the design requirements of efficient indexing and quick data lookup, several practical challenges arise.

Uniform Node Size:
In a Bloom tree, all nodes must have the same size to allow bitwise aggregation when generating parent nodes from child nodes. Each node has a fixed capacity defined by the number of data items it can index without exceeding the desired false positive rate. However, nodes at higher levels of the tree represent larger datasets than nodes at lower levels. For example, the root node indexes data from all neighbouring servers, whereas leaf nodes only represent individual servers. As a result, higher-level nodes may experience increased false positive rates. Increasing the node size to maintain accuracy may lead to unnecessary memory consumption at lower levels of the tree. Therefore, finding an appropriate balance between accuracy and memory efficiency becomes a challenge.

Fixed Node Size:
Another limitation of counting Bloom filters is that their size cannot be adjusted during runtime. In dynamic MEC environments, the amount of data stored on edge servers may change significantly over time. If a server suddenly receives a large volume of data, the fixed-size filter may become overloaded, reducing indexing accuracy. Conversely, servers storing very little data may waste memory if large filters are allocated. These limitations make fixed-size filters less suitable for highly dynamic edge environments.

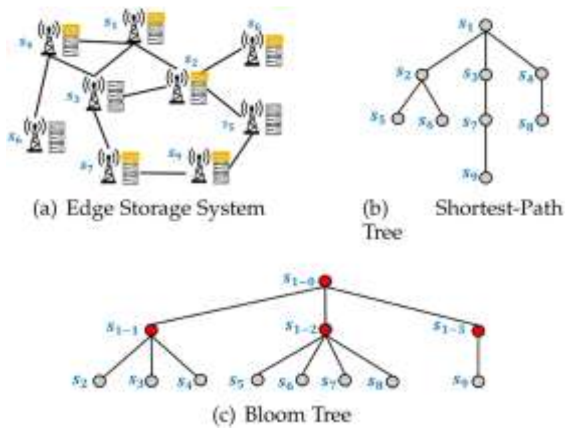


Fig.7. Building a Bloom tree. It shares the same structure, e.g., sizeanddepth, with the Ripple tree.

Bloom filters and their variants are widely used in distributed systems to check whether a particular data item exists in a dataset. Their low memory overhead and fast query performance make them suitable for mobile edge computing environments where resources are limited and low service latency is required. Bloom filters support data insertion and lookup operations with time complexity (O(k)), where (k) represents the number of hash functions used. However, a standard Bloom filter does not support deletion operations because each bit can only take values of 0 or 1. To address this limitation, counting Bloom filters (CBFs) are often used, as they allow deletion while maintaining the same time complexity.

Index Initialization and Maintenance:

To initialize the index tree, an edge server collects the counting Bloom filters from its neighbouring servers rather than transferring complete storage information. These filters allow the server to maintain a compact representation of data stored across the network. Whenever a data update occurs on a neighbouring server (s_j), such as insertion, deletion, or modification of a data item, the update information is broadcast to nearby nodes. These nodes then update their local index trees accordingly to maintain consistency. Although this approach keeps the index accurate, it may result in frequent information exchanges among edge servers.

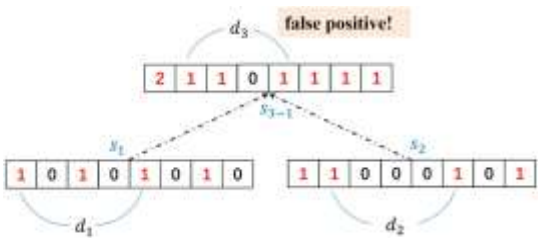


Fig. 8.False positive in counting Bloom filter tree.

Ripple Tree:

We propose an efficient indexing mechanism using a Ripple tree structure for duplicate detection.

Ripple introduces a new indexing structure called the Ripple Tree, which replaces counting Bloom filters with quotient filters. Unlike Bloom filters, quotient filters allow more flexible resizing and efficient data management. This flexibility enables Ripple to adapt to changing data volumes while maintaining high query efficiency.

A quotient filter provides functionality similar to Bloom filters for determining whether a data item exists within a dataset. Additionally, quotient filters support data movement, enabling multiple filters to be merged into a larger filter that represents the combined dataset.

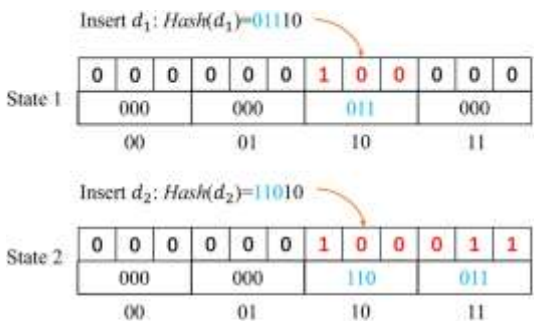

Insert d_1: $Hash(d_1)$=01110
State 1
Insert d_2: $Hash(d_2)$=11010
State 2

Fig. 9.Example of data insertion in quotient filter.

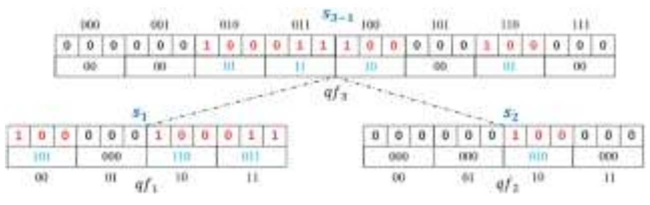


Fig. 10.Aggregating two quotient filters

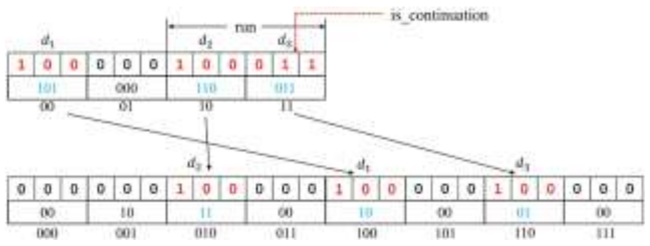


Fig. 11. Extending a 4-slot quotient filter to an 8-slot one.

Filter Contraction:

In this study, the expansion operation will be carried out when the occupancy ratio of the filter reaches or exceeds $\theta_e = 75\%$. Accordingly, it employs the threshold for filter contraction to $\theta_c = \theta_e/4$, also following. When the occupancy ratio of the filter drops to θ_c or below, it contracts. This setting prevents filter oscillation between expansion and contraction.

A quotient filter contraction goes through each data item in the filter as follows:

1) Based on the three flags provided, identify the actual slot address where the data item should have been inserted. This step is crucial because the data might have been forcibly moved from its original position.
2) Use the fingerprint and the actual slot address identified in the previous step, extract the original data from the filter.
3) Create a new filter with half the capacity of the original filter.
4) Modify the fingerprint hash value by shifting it left by one bit.
5) Set the highest bit of the new filter's address to match the lowest bit of the shifted fingerprint.
6) Utilize the remaining $n - 1$ bits (excluding the highest bit you set in the previous step) to determine the new address for the data item in the new filter.
7) Insert the modified fingerprint hash value into the newly determined address of the new filter

Data Query:

When querying whether a data item (d) exists in a quotient filter, the process begins by locating the slot corresponding to the address bits of (d). If the slot indicates that no data is present, the query immediately returns a negative result. Otherwise, the system searches through the corresponding cluster of entries to locate the fingerprint of the data item. By comparing fingerprints within the cluster, the system determines whether the queried data item exists in the dataset.

Data Deletion:

Deleting a data item from a quotient filter involves several steps. First, the fingerprint associated with the data item is located using its hash value. If the item belongs to a sequence of entries, the algorithm searches within that sequence until it finds the matching fingerprint. After removing the fingerprint, the subsequent entries are shifted to maintain the correct ordering within the filter, and the associated flags are updated accordingly.

Filter Aggregation:

When constructing a parent node in the Ripple tree, the filter size must be carefully determined to maintain accuracy. The capacity of the parent filter must be at least equal to the combined capacity of its child nodes. In addition, the false positive rate of the parent node should not exceed the minimum false positive rate among its child nodes. By solving these constraints, appropriate values for filter size parameters can be obtained, ensuring efficient aggregation without compromising accuracy.

Filter Expansion and Contraction:

In edge environments, data demands may change significantly over time. When the occupancy of a quotient filter reaches a predefined expansion threshold (typically 75%), the filter is expanded to maintain accuracy. Conversely, if the occupancy falls below a contraction threshold, the filter is reduced in size to avoid memory wastage. This dynamic resizing mechanism prevents frequent oscillation between expansion and contraction and ensures efficient memory utilization.

During contraction, the system processes each stored data item, reconstructs its original hash information, and reinserts it into a newly created filter with reduced capacity. This allows Ripple to maintain accurate indexing while adapting to changing data volumes within the edge network.

7. CONCLUSION AND FUTURE WORK

In this work, we present **Ripple**, a decentralized framework developed to improve data deduplication in edge computing environments. This utilizes a novel indexing structure called the Ripple tree, which enables edge servers to manage and locate data stored across neighbouring nodes effectively. By leveraging this structure, Ripple performs data deduplication at the edge without compromising data availability or violating latency constraints that are critical for edge computing applications. In addition, the dynamic expansion and contraction mechanisms of the Ripple tree help reduce memory overhead and allow the indexing system to adapt to changing data volumes.

Experimental evaluations tells that Ripple significantly improves system performance compared with existing approaches. Our experimental results indicate that Ripple can reduce average data retrieval latency by nearly **60%**, demonstrating its efficiency in decentralized edge environments. These improvements highlight the effectiveness of decentralized deduplication for improving storage utilization and ensuring efficient data access in modern edge computing environments.

Further research can focus on developing more advanced communication-optimization mechanisms to reduce coordination overhead among edge servers. Improvements may include integrating intelligent data placement strategies, adaptive replication policies, and enhanced security mechanisms to further strengthen the reliability and scalability of decentralized edge storage systems.

REFERENCES

1 Ruikun Luo et al, “Ripple,” Proc. IEEE, 2025

2 W. Shi, G. Pallis, and Z. Xu, “Edge computing,” *Proc. IEEE*, vol. 107, no. 8, pp. 1474–1481, Aug. 2019.

3 AWS, “AWS Wavelength for media & entertainment,” 2021. [Online]. Available: https://d1.awsstatic.com/Wavelength2020/AWS-Wavelengthfor-Media-Entertainment-SolutionBrief-Feb2021-Final.pdf

4 G. Zhu et al., “Pushing AI to wireless network edge: An overview on integrated sensing, communication, and computation towards 6G,” *Sci. China Inf. Sci.*, vol. 66, no. 3, 2023, Art. no. 130301.

5 Q. He, Z. Dong, F. Chen, S. Deng, W. Liang, and Y. Yang, “Pyramid: Enabling hierarchical neural networks with edge computing,” in *Proc. ACM Web Conf.*, 2022, pp. 1860–1870.

6 H. Jin, R. Luo, Q. He, S. Wu, Z. Zeng, and X. Xia, “Cost-effective data placement in edge storage systems with erasure code,” *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1039–1050, Mar./Apr. 2023.

7 W. Xiao, Y. Hao, J. Liang, L. Hu, S. A. Alqahtani, and M. Chen, "Adaptive compression offloading and resource allocation for edge vision computing," *IEEE Trans. Cogn. Commun. Netw.*, to be published, doi: 10.1109/TCCN.2024.3400820.

8 J. Peng et al., "MagNet: Cooperative edge caching by automatic content congregating," in *Proc. ACM Web Conf.*, 2022, pp. 3280–3288.

9 L. Yuan et al., "CSEdge: Enabling collaborative edge storage for multiaccessedgecomputingbasedonblockchain,"*IEEETrans.ParallelDistrib. Syst.*, vol. 33, no. 8, pp. 1873–1887, Aug. 2022.

10 R. Li, Z. Zhou, X. Zhang, and X. Chen, "Joint application placement and requestroutingoptimizationfordynamicedgecomputingservicemanagement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4581–4596, Dec. 2022.

11 X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 281–294, Feb. 2021.

12 J. Zhou, F. Chen, G. Cui, Y. Xiang, and Q. He, "FEUAGame: Fairnessaware edge user allocation for app vendors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 8, pp. 1429–1443, Aug. 2024.

13 X. Ge, S. Tu, G. Mao, C.-X. Wang, and T. Han, "5G ultra-dense cellular networks," *IEEE Wireless Commun.*, vol. 23, no. 1, pp. 72–79, Feb. 2016.

14 M. Sihag et al., "A data-driven approach for finding requirements relevant feedback from TikTok and YouTube," in *Proc. IEEE Int. Requirements Eng. Conf.*, 2023, pp. 111–122.

15 J. Yang, A. Sabnis, D. S. Berger, K. Rashmi, and R. K. Sitaraman, "C2DN: How to harness erasure codes at the edge for efficient content delivery," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2022, pp. 1159–1177.

16 Z. Liu et al., "DistCache: Provable load balancing for large-scale storage systems with distributed caching," in *Proc. USENIX Conf. File Storage Technol.*, 2019, pp. 143–157.

17 X. Zou, W. Xia, P. Shilane, H. Zhang, and X. Wang, "Building a highperformance fine-grained deduplication framework for backup storage with high deduplication ratio," in *Proc. USENIX Annu. Tech. Conf.*, 2022, pp. 19–36.

18 Y. Shin, D. Koo, and J. Hur, "A survey of secure data deduplication schemes for cloud storage systems," *ACM Comput. Surv.*, vol. 49, no. 4, pp. 1–38, 2017.

19 X.Zou,J. Yuan,P.Shilane, W.Xia, H.Zhang,andX. Wang, "Thedilemma between deduplication and locality: Can both be achieved?" in *Proc. USENIX Conf. File Storage Technol.*, 2021, pp. 171–185.

20 Z. Cao, S. Liu, F. Wu, G. Wang, B. Li, and D. H. Du, "Sliding lookback window assisted data chunk rewriting for improving deduplication restore performance," in *Proc. USENIX Conf. File Storage Technol.*, 2019, pp. 129–142.

21 W. Xia et al., "A comprehensive study of the past, present, and future of data deduplication," *Proc. IEEE*, vol. 104, no. 9, pp. 1681–1710, Sep. 2016.

22 W. Y. B. Lim et al., "Decentralized edge intelligence: A dynamic resource allocation framework for hierarchical federated learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 536–550, Mar. 2022.

23 R. Kübler, "How to build popularity-based recommenders with polars," 2023. [Online]. Available: https://towardsdatascience.com/how-to-buildpopularity-based-recommenders-with-polars-cc7920ad3f68

24 F. Niand S. Jiang, "RapidCDC:Leveraging duplicatelocality to accelerate chunking in CDC-based deduplication systems," in *Proc. ACM Symp. Cloud Comput.*, 2019, pp. 220–232.

25 R. Kisous, A. Kolikant, A. Duggal, S. Sheinvald, and G. Yadgar, "The what, the from, and the to: The migration games in deduplicated systems," *ACM Trans. Storage*, vol. 18, no. 4, pp. 1–29, 2022.

26 R. Luo, H. Jin, Q. He, S. Wu, and X. Xia, "Enabling balanced data deduplication in mobile edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 5, pp. 1420–1431, May 2023.

27 P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. S. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Trans. Multimedia*, vol. 21, no. 4, pp. 915–929, Apr. 2019.