Efficient Cloud Storage Mechanism for IoT Environment

ABSTRACT

This thesis aims to address the pressing need for efficient and secure management of multimedia data in today's digital landscape. Motivated by the growing volume

and importance of multimedia data, the thesis endeavors to develop a comprehensive multimedia data management system. The primary problem addressed is the challenge of ensuring efficient storage and retrieval while maintaining data integrity and security in cloud storage environments.

Methodologically, the thesis leverages various techniques, including data retrieval, encryption, deduplication, and performance simulation, to tackle this challenge comprehensively. Data retrieval is facilitated through a robust mechanism designed to efficiently access multimedia data stored in cloud storage systems. By employing structured tables and utilizing identifiers, segment numbers, and hash values, data retrieval ensures both integrity and efficiency.

Security is a paramount concern, and to address it, the system employs symmetric encryption techniques such as the Advanced Encryption Standard (AES). Each multimedia segment undergoes encryption using a unique encryption key, ensuring confidentiality during storage and transmission, thereby safeguarding against unauthorized access and potential data breaches.

Furthermore, storage redundancy is minimized, and resource utilization optimized through deduplication techniques. By maintaining a hash table and identifying duplicate multimedia segments, the system conserves storage space and enhances overall efficiency, reducing the risk of data inconsistency.

The thesis also includes simulations for file transfer times and throughput, considering various file sizes and concurrency levels. These simulations provide valuable insights into system performance under diverse conditions, enabling informed decision-making and optimization strategies.

Through the integration of key generation, block processing, and MDPC algorithm implementation, the multimedia data management system offers enhanced functionality and performance. Overall, this research yields a comprehensive solution for the efficient, secure, and optimized handling of multimedia data in cloud storage environments.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

List of Abbreviations

Abbreviation	Meaning
AES	Advanced Encryption Standard
AI	Artificial Intelligent
HDD	Hard Disk Device
SSD	Solid State Device
DPC	Dynamic Prime Chunking
AE	Asymmetric Extremum
TTTD	Two Threshold Two Divisor
DER	Deduplication Element ratio
BSPS	Byte Saved Per Second
CDC	Content Defined Chunking
CL	Chuck Length
HDFS	Hard Distributed File System
TEE	Trusted Encryption Environment
DD	Data Deduplication
FF	Fingerprint
GH	Gear Hash
CA	Chord Algorithm
CSE	Cloud storage Environment

QoS	Quality of Service
DSFSC	Dual Side Fixed Size Chunking
GDPR	General Data Protection Regular
HIPAA	Health Insurance Probability and Account Act
HV	Hash Value
LMC	Lesk Measure of cohesion chunking
RAM	Rapid Asymmetric Maximum
SHA	Secure Hash Algorithm
MDPC	Multiplicative-Divisive Probabilistic Congestion Control
MIDD	Multiplicative Increase, Divisive Decrease
RTT	Round Trip Time
AIMD	Additive-Increase Multiplicative-Decrease
CSP	Cloud Service Provider
FTP	File Transfer Protocol
NPS	Net Promoter Score

CHAPTER ONE

INTRODUCTION

CHAPTER-1

INTRODUCTION

1.1Introduction

In our increasingly digital world, data has become a precious asset driving innovation and efficiency across industries. However, managing this vast and diverse volume of data poses significant challenges, particularly in the context of IoT environments. Traditional storage solutions are often insufficient, prompting the adoption of cloud storage as a versatile alternative. Cloud storage not only offers ample space but also accessibility and security, making it an attractive option for IoT applications. The integration of (AI) and cloud systems further enhances the potential of IoT data, enabling efficient mining and analysis. IoT's embedded intelligence empowers sensors to collect and analyze data, revolutionizing processes across various sectors. Cloud storage facilitates the storage and processing of this data, paving the way for enhanced operations and insights. [1] Despite its benefits, cloud storage in IoT environments comes with its share of concerns, particularly regarding security and control. Entrusting sensitive data to third-party providers raises apprehensions about data integrity and privacy. Additionally, challenges such as data migration and dependency on internet connectivity must be addressed. [2] While cloud storage offers advantages such as disaster recovery, scalability, and cost-effectiveness, it also poses challenges related to data control, vendor lock-in, and connectivity issues. Understanding these pros and cons is essential for organizations considering the adoption of cloud storage solutions in IoT applications. [3] In conclusion, cloud storage holds promise for revolutionizing data management in IoT environments, offering both benefits and challenges. By carefully evaluating its implications and addressing potential drawbacks, organizations can leverage cloud storage to unlock the full potential of IoT data while ensuring security and efficiency in their operations. [4]

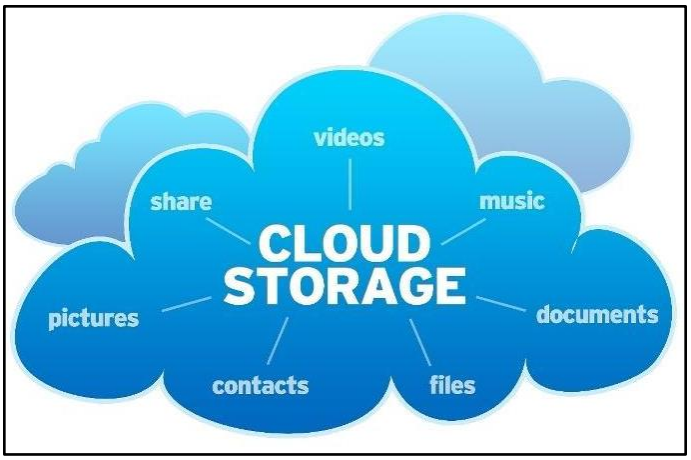


Figure 1.1: Cloud Storage [3]

1.2Cloud Storage Mechanism

Every cloud has a certain amount of storage, so if start uploading duplicate information, the storage will be lost, and dealing with data redundancy will become a major issue. Researchers have been investigating numerous techniques to combat this, and data deduplication is the best answer. A method called data deduplication was developed to improve storage [77]. Different cloud service providers, including Dropbox, Amazon S3 and Google Drive, now use this strategy. Data duplication is prevented by making sure it is never uploaded to the cloud more than once.

A. As the amount of digital data grows, so does the need for greater storage space.

B. Traditional solutions don't have any built-in protection against duplicate data being saved up.

C. Data De-duplication is critical for removing redundant data and lowering storage costs.

The quantity of data generated is growing exponentially in quickly developing digital age. The demand for more storage space has grown as more areas of life, from social media interactions to business transactions, are becoming digitalized. This article looks at how inadequate present storage capabilities are for keeping up with the rate of expansion in digital data and the significance of finding a solution.

- ***A Partial Solution:*** The increased need for storage space has a partial solution in the form of cloud storage. Cloud service providers can offer scalable storage options to consumers and businesses by utilising the enormous capabilities of data centres. This method, however, has its own set of drawbacks, such as worries about data privacy, security lapses, and dependence on outside sources [9]. Additionally, the cost of storing significant amounts of data on the cloud can rise significantly, particularly for long-term retention.
- ***Explosive Growth of Digital Data:*** The internet's rising use, the widespread use of smartphones, and the rise of connected gadgets have all contributed to the digital revolution's data explosion. The amount of digital data is always growing because of all online interactions, transactions, sensor readings, and media uploads.
- ***New Technologies for Data-Intensive Systems:*** The problem with storage is made worse by the emergence of data-intensive technologies like *artificial intelligence, machine learning* , and big data analytics. Massive datasets are needed for these applications in order to build models and gain insightful knowledge. Additionally, the growing use of virtual reality, augmented reality, and high-definition multimedia content puts extra pressure on storage infrastructure by necessitating higher capacity and quicker data retrieval.

The lack of storage capacity is becoming an urgent issue as the digital world develops. Finding scalable and effective storage solutions is urgent given the exponential growth of digital data and the rising demand for data-intensive applications. While cloud storage provides a partial solution, research into next-generation storage systems is necessary to make sure that the storage infrastructure can sustain the ever-growing digital world [11]. It can fulfil the increasing need for storage space and unleash every advantage of the digital age by making investments in technology development and promoting innovation.

The problem of redundant information has grown significantly in importance in the era of expanding digital data. Traditional storage solutions frequently do not have built-in duplicate data management tools. The significance of data deduplication in eliminating redundant data and lowering storage costs is highlighted in this article.

Duplicate data refers to information that is identical and spread across different locations in a storage system. It may be caused by a number of things, including user

error, system backups, or data replication procedures [13]. Duplicate data not only takes up valuable storage space, but it also drives up prices, slows down data retrieval, and uses resources inefficiently. Hard disc drives (HDDs) and solid-state drives (SSDs), two common types of traditional storage, lack built-in techniques for locating and removing duplicate data. Organisations can considerably reduce their storage needs by getting rid of duplicate data. However, ensuring that only one copy of each piece of information is stored, data deduplication increases data efficiency. Enhancing data integrity means reducing duplicate data [14]. Duplicate data can cause conflicts and inconsistencies, jeopardising the accuracy and dependability of data that is kept. Disaster recovery procedures might be hampered by duplicate data since it increases backup and restore times. In today's data-driven world, adopting data de-duplication is essential for effectively managing and maximising the value of digital data.

1.3 **Literature Survey**

K. Vijayalakshmi and V. Jayalakshmi in (2021) [7] suggest data duplication in clouds, which is managed using the de-duplication technique. Although some de-duplication techniques are used to prevent data redundancy, they are inefficient. The major goal of this research is to gain enough knowledge and a decent concept of de-duplication techniques through reviewing existent ways, and this work may aid future research in establishing effective cloud storage management (CSM) solutions for researchers.

M. Ellappan and S. Abirami in (2021) [8] suggest a novel chunking algorithm called Dynamic Prime Chunking (DPC). DPC's major purpose is to modify the window size during the prime value dynamically rely on the maximum and minimal chunk size. DPC in the de-duplication scheme gives good throughput while avoiding large chunk variance. The multimedia and operating system datasets were used for implementation and experimental evaluation. Existing algorithms such as AE, MAXP, TTTD, and Rabin have been compared to DPC. The performance indicators looked at were throughput, chunk count, Bytes Saved per Second (BSPS), chunking time, processing time and De-duplication removal Ratio (DER). BSPS and throughput have both improved. To begin, DPC boosts throughput performance by

greater than 21% when compared to AE. BSPS improves performance by up to 11% over the previous AE method.

P.Anitha et al. in (2021) [9] the secure authorities are given access control mechanisms to do data de-duplication (DD) on the data that was outsourced. Encryption techniques are used in the Access Control Mechanism. It employs convergent randomised encryption and a reliable distribution of owning party keys to allow the cloud service provider to manage outsourced data access even when control shifts on a regular basis. The suggested technique safeguards data integrity against attacks relies on label discrepancies. As a precaution, the suggested technique has been changed to improve security.

Xu and W. Zhang in (2021) [10] QuickCDC improves CDC chunking speed, de-duplication ratio, and throughput by combining three methods. Initially, QuickCDC can move instantly to the chunk boundaries of duplicate chunks that arise frequently. The mapping of the duplicate chunk's first n bytes and last m bytes to chunk length must be registered. The first n bytes and last m bytes of the current chunk are checked to see if they are in the mapping table when chunking is performed. QuickCDC can skip relevant chunk lengths (CL) if they are in the mapping table. QuickCDC can skip the minimal chunk length for unique chunks. Finally, QuickCDC may dynamically alter mask bits length such that chunk length (CL) is permanently more than the minimal chunk length and is distributed in a limited particular location. When the current chunk length (CL) is less than the expected chunk length (CL), should use longer mask bits, and when the current chunk length (CL) is more than the expected chunk length (CL), should utilize shorter mask bits. Experiments show that QuickCDC's chunking speed is 11.4x that of RapidCDC, and the associated de-duplication ratio is somewhat increased, with a maximum de-duplication ratio improvement of 222.3% and a throughput improvement of 111.4%.

N. Kumar and S. Jain in (2019) [11] suggest Differential Evolution DE-rely on TTTD-P optimized chunking to maximize chunking throughput while increasing de-duplication ratio DR The use of a scalable bucket indexing strategy minimizes the time it takes to find and declare duplicated hash values (HV). It chunks about 16 times greater than Rabin CDC, 5 times greater than AE CDC, and 1.6 times greater than FAST CDC (HDFS).

Y. Fan et al., in (2019) [12] system improves the capacity of like cryptosystems to resist selected plaintext and selection ciphertext attacks by augmenting convergent encryption with users' privileges and relying on TEE to provide secure key management. system is secured sufficient to facilitate data de-duplication (DD) as well as protecting the privacy of sensitive data, according to a security analysis. Moreover, create a prototype of system and analyze its performance. Experiments reveal that system overhead is practical in real-world scenarios.

H. A. Jasim and A. A. Fahad, in (2018) [13] novel fingerprint function (FF), a multi-level hashing and matching mechanism, and a novel indexing technicality to hold metadata to progress the TTTD chunking algorithm. These novel technicalities include four hashing algorithms to handle the collision issue, as well as adding a novel chunk stipulation to the TTTD chunking criterion to improve the number of small chunks and hence the De-duplication Ratio.

H. Wu. In (2018) [14] suggests a sampling-rely on chunking algorithm and improve SmartChunker, a tool to predict the appropriate chunking configuration for de-duplication schemes. Smart Chunk's effectiveness and efficacy have been demonstrated in real-world datasets.

M. Oh et al., in (2018) [15] suggest novel de-duplication technique that is extremely compatible and scalable with the exhausted storage currently in use. The approach combines file system and de-duplication meta-information into a single object, and it manages the de-duplication ratio online through initial aware of post-processing-related scheme demands. When executing a variety of standard storage workloads, the experimental findings illustrate that solution could save greater than 90% of total storage space while providing the same or similar performance as traditional scale-out storage.

W. Xia et al. in (2016) [16] suggest FastCDC, a Fast and effective CDC approach, which constructs and enhances on the latest Gear-based on CDC technique, one of the fastest CDC techniques to knowledge. FastCDC's main idea is to integrate five key mechanics: gear-rely on rapid rolling hash, improving and simplifying Gear hash (GH) verdict, skipping sub-minimal chunk cut-points, normalizing the chunk-size distribution in a small specific region to address the issue of reduction de-duplication

ratio caused by cut-point skipping. FastCDC is around 10 times quicker than the best open- source Rabin-based on CDC, and about 3 times greater than the state-of-the-art Gear- and AE-rely on CDC, while obtaining almost the same de-duplication ratio as the standard Rabin-rely solution, according to evaluation results.

X. Xu. et al. in (2016) [17] focus on non-center cloud storage data de-duplication and present a new two-side data de-duplication (DD) mechanism. The Chord algorithm (CA) is optimized. The suggested two-side data de-duplication (DD) technique outperforms the traditional data de-duplication (DD) mechanism in terms of de-duplication rate.

R. Kiruba karan et al. in (2015) [18] present a cloud-based technique for achieving de-duplication of a huge amount of data available. The approach includes data de-duplication before uploading to cloud storage as well as data reverse de-duplication when obtaining the required data. The model is more effective and accurate than existing de-duplication systems because of the type of algorithm utilized.

V. Maruti et al. in (2015) [19] the main goal of this technique is to delete reiterate data from the cloud. It can also aid in the reduction of bandwidth and storage space usage. Each user has their own unique token and has been allocated various privileges based on the duplication check. The hybrid cloud architecture is used to achieve cloud de-duplication. The proposed technique is more secure and uses fewer cloud resources. It was also demonstrated that, when compared to the standard De-duplication technique, the proposed system had a low overhead in duplicate removal. On this work, both content level and file level de-duplication of file data is examined in the cloud.

X. Xu and Q. Tu, in (2015) [20] de-duplication scheme architecture for cloud storage environments (CSE). DelayDedupe, a delayed target de-duplication strategy rely on chunk-level de-duplication and chunk access frequency, is suggested to decrease response time in storage nodes (S nodes). When used in conjunction with replica arrangement, this technique evaluates whether fresh multiplied chunks for data update are hot and, if they aren't, eliminates the hot duplicated chucks. The findings of the experiment show that the DelayDedupe method may successfully minimize response time while also balancing the storage demand on Nodes.

Y. Zhang in (2015) [21] Suggested a novel CDC algorithm indicated the Asymmetric Extremum (AE) algorithm. The major idea behind AE is relies the observance that in dealing with the boundaries-shift issue, the maximum value in an asymmetric local domain is improbable to be exchanged through a novel extreme value, which motivates AEs utilize of asymmetric (instead of symmetric, as in MAXP) local domain to distinguish cut-points and attain high chunking throughput while minimizing chunk size variance. According to the result, AE addresses the issues of low chunking throughput in MAXP and Rabin, as well as excessive chunk-size volatility in Rabin, at the same time. AE enhances the throughput speed of state-of-the-art CDC algorithms by 3x while achieving equivalent or greater de-duplication efficacy, according to experimental results that rely on four real-world datasets.

W. leesakul et al. in (2014) [22] suggested dynamic data de-duplication (DD) strategy for cloud storage, in order to strike a balance among changing storage efficacy and criteria for fault tolerance, as well as to increase cloud storage performance. adjust the number of copies of files in real time to match the changing degree of QoS. The results of the experiments reveal that suggested scheme works effectively and can deal with scalability issues.

Krishnaprasad and B. A. Narayamparambil in (2013) [23] suggested a novel Dual Side Fixed Size Chunking (DSFSC) algorithm to achieve a rising de-duplication ratio for comparison to conventional FSC. This approach can successfully be utilized for audio or video files to produce a best De-duplication ratio without requiring computationally exorbitant variable size chunking or content determined chunking. Storage management and energy expenses will be reduced if storage requests are reduced.

In recent years, the proliferation of multimedia data in various applications has brought forth challenges related to security, data analytics, sharing, and optimization. This literature review synthesizes findings from four key studies in the field, focusing on secure multimedia data processing, data analytics, security models, and optimization techniques in diverse contexts.

Srinivasan et al. (2022) propose a Secure Multimedia Data Processing Scheme for medical applications. They address the crucial need for security in handling sensitive medical data by introducing a scheme that ensures secure processing of multimedia data. By employing encryption techniques and access controls, the proposed scheme aims to safeguard patient privacy and prevent unauthorized access to medical records. This study underscores the importance of security measures in medical applications to maintain data integrity and confidentiality.

Kumari and Tanwar (2022) present a Secure Data Analytics Scheme tailored for multimedia communication within a decentralized smart grid infrastructure. Focusing on the energy sector, their scheme addresses the challenges of securing data analytics processes in decentralized environments. By integrating encryption, authentication, and anomaly detection mechanisms, the proposed scheme enhances the security of multimedia data transmission and analysis in smart grid networks. This study highlights the significance of secure data analytics in ensuring the reliability and integrity of critical infrastructure systems.

Dhar et al. (2023) introduce an Advanced Security Model for Multimedia Data Sharing in the Internet of Things (IoT) environment. Recognizing the vulnerability of IoT devices to security threats, their model offers a comprehensive approach to securing multimedia data sharing in IoT ecosystems. Through the integration of access control, authentication, and encryption techniques, the proposed model aims to mitigate risks associated with unauthorized access and data breaches. This study emphasizes the need for robust security models to address the unique challenges posed by multimedia data sharing in IoT environments.

Sharma et al. (2023) focus on optimizing multimedia data using computationally intelligent algorithms. Their study explores the application of artificial intelligence techniques to enhance the efficiency and performance of multimedia systems. By leveraging intelligent algorithms such as machine learning and optimization algorithms, the proposed approach aims to optimize multimedia data processing, storage, and retrieval. This research underscores the potential of computational intelligence in addressing the complexities of multimedia data management and improving system performance.

Overall, the reviewed studies underscore the critical importance of security, data analytics, and optimization techniques in handling multimedia data across various domains. These studies offer valuable insights and methodologies for addressing the challenges associated with secure processing, sharing, and optimization of multimedia data, thereby contributing to advancements in the field of multimedia tools and applications.

S. Luo and M. Hou in (2013) [24] suggest a new chunk coalescing algorithm (CCA), this refers to the minimal and maximum amount of sub chunks which should be coalesced to form super chunks (SC). Experiments demonstrate that algorithm eliminates the expenses of the chunk coalescing (CC) procedure and speeds up the entire data de-duplication procedure.

Table 1.1: Comparison of studies over Data De-duplication & chunking algorithm

S. No.	Authors	Algorithm/method/ Techniques	Advantages	Drawback
1	N. Kumar and S. Jain 2019	Differential Evolution (DE), Two Thresholds Two Divisors (TTTD-P) algorithm,	Hash values (chunks about 16 times greater than Rabin CDC, 5 times greater than AE CDC, and 1.6 times greater than FAST CDC	Take too much time to calculate the hash value.
2	W. Leesakul et al. 2014	Dynamic Data De-duplication	experiments reveal that our proposed system works effectively	Cannot work with the encryption keys

3	Y. Fan et al. 2019	De-duplication system that includes the processes of duplicate checking	implement the security analysis and also performance evaluation is effective and feasible in practice	Take too much processing power of the system and consume more power
4	M. Oh et al. 2018	A novel de-duplication technique	experimental findings illustrate that our solution could save greater than 90% of total storage space	It will occupy more than 20% more storage than other algorithms
5	P. Anitha et al. 2021	secure rising scalable data de-duplication architecture	The system is virtually as successful as the existing ones (minor increase in computational overhead)	Risk factors high
6	R. Kiruba karan et al 2015	a cloud-rely technique for de-duplication of huge data	The model is more efficient and accurate compared to that of the existent de-duplication techniques.	The model is efficient but it is too costly.
7	M. V. Maruti et al. 2015	novel duplication check technique that configuration the token for the private file	the system achieve is 98 %	Consume more power for execution

8	K. Vijayalakshmi and V. Jayalakshmi 2021	data duplication (DD) in clouds	the system achieves efficient knowledge and a good idea concerning de-duplication techniques	Can not manage TB of data in the cloud environment
9	X. Xu et al 2016	two- side data de-duplication (DD) technique, Chord algorithm	two-side data de-duplication (DD) technique outperforms the traditional data de-duplication technique in terms of de-duplication rate	Can not manage more than 50 VMs
10	X. Xu and Q. Tu 2015	de-duplication scheme architecture for cloud storage environments (CSE)	Delay Dedupe method may successfully minimize response time while also balancing the storage demand on Snodes	algorithms often lack comprehensive validation and may not be well-understood by the research or practitioner communities
11	M. Ellappan and S. Abirami 2021	Dynamic Prime Chunking (DPC), Existing algorithms,	DPC's durable performance over the another existent algorithms in terms of BSPS and the efficacy of the backup	Storage and cost high

			storage scheme	
12	H. A. Jasim and A. A. Fahad 2018	a novel fingerprint function (FF),	good de-duplication ratio and rapid execution time, efficacy of the suggest algorithm was evaluated utilizing two relatively datasets	Efficiency increases but attack rate is high
13	W. Xia et al., 2016	FastCDC, a Fast and effective CDC approach	FastCDC is around 10 times quicker than the best open-source Rabin- based on CDC, and about 3 times greater than the state-of-the-art Gear- and	Algorithms in terms of Chunk and the efficacy of the backup storage is less
14	Z. Xu and W. Zhang 2021	Content Defined Chunking (CDC)	Show that QuickCDC's chunking speed is 11.4x that of RapidCDC, and the associated de-duplication ratio is somewhat	drawback of the Content-Defined Chunking (CDC) algorithm is its potential sensitivity to changes in data

			increased, with a maximum de-duplication ratio improvement of 222.3%	patterns.
15	S. Luo and M. Hou 2013	a new chunk coalescing. algorithm (CCA)	demonstrate that our algorithm eliminates the expenses of the chunk coalescing procedure and enhance the efficacy of hash-comparison	CCA may not perform optimally across all types of data or workloads. It is primarily designed to reduce redundancy in similar chunks, so it may not be as effective for datasets.
16	H. Wu et al. 2018	a sampling-based on chunking algorithm and improve SmartChunker	illustrate that a sampling-based chunking algorithm and enhance SmartChunker application-specified chunk configurations	The algorithm's efficiency can be compromised if the chosen sampling strategy introduces bias, leading to suboptimal chunk boundaries

and reduced effectiveness

1.4 Research problem

The rapid proliferation of the Internet of Things (IoT) has revolutionized the way data is generated, transmitted, and utilized, fostering unprecedented opportunities for innovation and efficiency across various domains. However, the seamless integration of IoT devices with cloud computing platforms has brought forth a myriad of challenges, particularly concerning the storage and management of vast volumes of IoT-generated data. In this context, the pressing research problem lies in the development of a robust and secure data storage mechanism tailored explicitly for cloud based IoT applications. The current landscape of data storage in such environments is fraught with obstacles, ranging from data security and privacy concerns to the optimization of storage and retrieval processes.

First and foremost, the paramount concern revolves around ensuring the security and privacy of IoT-generated data stored within the cloud. Given the sensitive nature of much of this data, including personal and proprietary information, stringent measures must be implemented to safeguard against unauthorized access, data breaches, and malicious attacks. Furthermore, compliance with regulatory frameworks, such as GDPR and HIPAA, adds an additional layer of complexity to data security requirements. Moreover, the scalability and accessibility of data storage solutions in cloud based IoT environments pose significant challenges. As the volume of data continues to escalate exponentially with the proliferation of IoT devices, traditional storage architectures struggle to keep pace with the demands for scalability and efficiency. Hence, there is a critical need for innovative approaches that can seamlessly scale storage resources in response to fluctuating workloads

while ensuring high availability and reliability. Furthermore, optimizing storage and retrieval processes to enhance overall system performance and efficiency is imperative. With the diverse nature of IoT-generated data, ranging from real-time sensor readings to multimedia content, the design of storage mechanisms must be tailored to accommodate varying data types and access patterns efficiently. This entails the exploration of novel data storage architectures, data indexing techniques, and data retrieval algorithms optimized for cloud-based IoT environments.

Addressing these multifaceted challenges requires a holistic approach that encompasses technological innovation, robust security measures, regulatory compliance, and efficient resource management. By developing an effective and highly secure data storage mechanism specifically tailored for cloud based IoT applications, we can unlock the full potential of IoT technologies while mitigating the associated risks and ensuring the integrity and confidentiality of sensitive data. This research endeavour holds immense significance in shaping the future trajectory of IoT and cloud computing ecosystems, paving the way for a more connected, secure, and resilient digital

1.5 Research Objectives

The aim of this research is to design a new efficient mechanism for cloud storage and Internet of things environments. The proposed mechanism is designed to gain attention in large-scale storage systems based on text, image, and video. Hence, in order to achieve the research aim, the following objectives are formulated:

1) To design a new mechanism to improve the performance of a large storage system by applying the de-duplication technique.

2) To evaluate the performance of the proposed mechanism in comparison with available solutions in a simulated environment.

3) To verify and validate the proposed mechanism based on the results obtained from the simulation experiments that ensure the correctness of its implementation.

1.6 Contribution of the Research

The networked machines can connect with one another thanks to middleware, which is another piece of software employed by the central server.

1) Mechanism Design: The research proposes a mechanism for cloud storage in IoT environments, focusing on enhancing performance through the implementation of deduplication techniques. This approach aims to optimize resource utilization and reduce storage costs, addressing a key challenge in large-scale storage systems.
2) Performance Evaluation: The research conducts comprehensive performance evaluations of the proposed mechanism compared to existing solutions in simulated environments. By rigorously assessing its efficiency, scalability, and reliability, the study provides valuable insights into the effectiveness of the proposed approach in real-world applications.
3) Validation and Verification: Through rigorous validation and verification processes based on simulation experiments, the research ensures the correctness and effectiveness of the proposed mechanism. By verifying its functionality and performance against established benchmarks, the study establishes the reliability and viability of the proposed solution for practical deployment in cloud-based IoT environments.

1.7 significance of the research

The successful resolution of the problem regarding an effective secured data storage mechanism for cloud based IoT promises to bring about a multitude of significant by enhancing security and privacy measures within the proposed mechanism, the research directly addresses Objective. Achieving heightened security ensures the protection of IoT-generated data stored in the cloud, aligning with the objective to design a mechanism to improve large storage system performance through deduplication techniques. As the proposed mechanism ensures secure and efficient data storage, it directly contributes to instilling greater confidence in the utilization of IoT technologies, supporting Objective. Organizations and individuals will trust IoT applications more knowing their data is securely stored, thereby validating the mechanism's performance in comparison with existing solutions. The enhanced accessibility and reliability of IoT data resulting from the proposed mechanism

directly support Objective through verification and validation processes, the research confirms the correctness and effectiveness of the mechanism, ensuring its reliability in storing and retrieving data seamlessly. Efficient resource utilization, including cost and energy savings, is a direct outcome of the proposed mechanism, in line with Objective by optimizing storage efficiency through deduplication techniques, the mechanism minimizes resource wastage, contributing to the performance improvement of large-scale storage systems. The proposed mechanism's ability to seamlessly scale to accommodate increasing volumes of IoT data aligns with Objective through simulation experiments and performance evaluations, the research verifies the mechanism's scalability, ensuring its suitability for evolving IoT deployments without storage limitations.

1.8 Outlines of Thesis

The following chapters are presented in this thesis: Chapter One presents the basic introduction, problem statement, methodology objective of the study and some other aspects. Chapter Two presents the theoretical background. It theoretically explains the method and techniques used in this study. The proposed methods or methodology used in this study will be depicted in Chapter Three. The collected methods, techniques, algorithms collected in the proposed methodology will be analyzed in this chapter. The primary outcomes of the proposed system employing various strategies are shown in Chapter Four. The findings are given separately for each model. Chapter Five summarises the results reached throughout this thesis, overall conclusion derives from the study and briefly lists potential future works.

CHAPTER TWO

THEORETICAL BACKGROUND

CHAPTER – 2

THEORETICAL BACKGROUND

2.1 Performance Metrics and its Types:

In today's interconnected world, the utilization of cloud storage in IoT environments has become paramount for efficient data management. However, the success of cloud storage implementation depends not only on its benefits but also on its performance. Performance metrics play a crucial role in evaluating the effectiveness and efficiency of cloud storage solutions for IoT applications. This paper explores various types of performance metrics essential for assessing cloud storage performance in IoT environments.

1. Throughput: Throughput measures the rate at which data can be transferred to and from the cloud storage system. In IoT environments, where data is continuously generated and transmitted by numerous sensors and devices, high throughput is critical for timely data processing and analysis. Throughput metrics assess the system's ability to handle data influx efficiently, ensuring smooth operations and real-time insights.
2. Latency: Latency refers to the delay between data transmission and reception, affecting the responsiveness of IoT applications. Low latency is essential for applications requiring immediate data processing, such as real-time monitoring and control systems. Latency metrics evaluate the speed of data retrieval and processing within the cloud storage infrastructure, ensuring minimal delay and optimal performance for IoT devices.
3. Availability: Availability measures the accessibility of data stored in the cloud storage system. In IoT environments, where data accessibility is vital for decision-making and operation, high availability is crucial to ensure uninterrupted access to critical information. Availability metrics assess the reliability of the cloud storage infrastructure, including backup and redundancy mechanisms, to maintain continuous data availability and prevent downtime.
4. Scalability: Scalability evaluates the ability of the cloud storage system to accommodate growing data volumes and user demands. In dynamic IoT environments, where data volumes can fluctuate rapidly, scalable storage solutions

are essential to accommodate evolving needs. Scalability metrics assess the system's capacity to scale resources seamlessly, ensuring consistent performance and resource utilization as data requirements evolve over time.

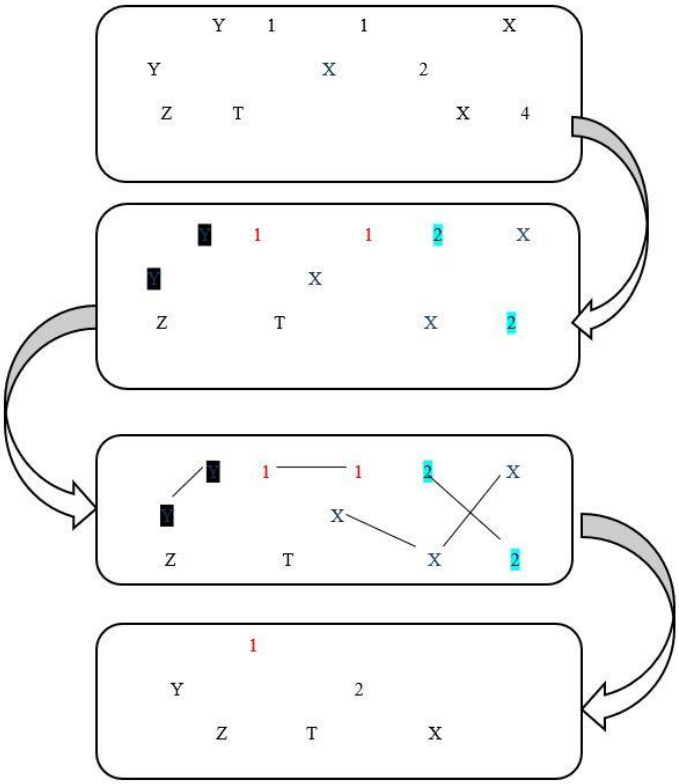
5. Reliability: Reliability measures the consistency and predictability of cloud storage performance over time. In IoT environments, where data integrity is crucial for accurate analysis and decision-making, reliable storage solutions are essential to maintain data consistency and integrity. Reliability metrics assess factors such as system stability, data durability, and error handling capabilities to ensure consistent performance and data integrity in diverse operational conditions.

2.2 Data Deduplication

Data duplication in Cloud IoT environments represents a critical challenge and opportunity in the modern digital landscape. As IoT devices become more prevalent and diverse, generating vast volumes of data, efficient data management is paramount. Data duplication, in this context, refers to the occurrence of redundant data across multiple IoT devices and cloud storage systems. Addressing this issue is pivotal to optimizing storage resources, enhancing data processing speed, and ensuring cost-effective operations in cloud based IoT setups. Cloud IoT environments leverage sophisticated algorithms and techniques to identify and eliminate duplicated data efficiently. By employing deduplication methods, such as hash-based comparisons and metadata indexing, redundant data can be systematically identified and stored only once, saving precious storage space and network bandwidth [25].

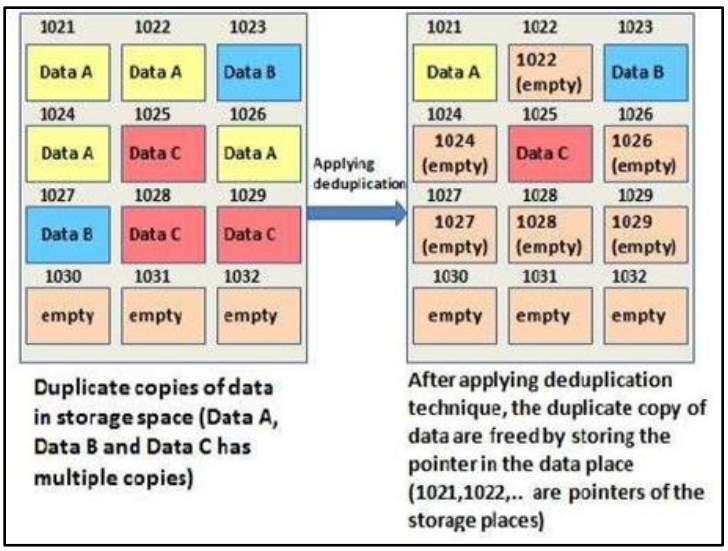
Furthermore, in the context of Cloud IoT, data deduplication plays a crucial role in ensuring data integrity, security, and real-time processing efficiency. Reducing data duplication not only conserves storage resources but also enhances data analytics and decision-making processes. In scenarios where real-time responses are essential, eliminating duplicate data ensures that the analytics systems receive accurate and up-to-date information, leading to more informed decisions. Additionally, deduplication mitigates the risks associated with storing multiple copies of sensitive IoT data, promoting data security and privacy compliance [26]. By implementing robust data deduplication techniques within Cloud IoT environments, businesses can

unlock the full potential of their IoT ecosystems, fostering innovation and enabling seamless integration of IoT technologies into various applications and industries.



(a) Fig: 2.1 Data Duplication [78]

The data duplication figure illustrates the process of data deduplication, which involves identifying and eliminating duplicate copies of data. It includes steps such as chunking data into smaller pieces, identifying unique data chunks, and replacing redundant chunks with references to the unique ones. The figure also depict how data deduplication improves storage efficiency and network transfer by reducing the amount of redundant data stored and transmitted.



(b) Fig: 2.1 Data Duplication [79]

The data duplication figure illustrates various techniques and processes involved in data deduplication for storage of big data in the cloud. It depicts methods such as chunking, hashing, indexing, and duplicate detection algorithms. The figure also highlights how these techniques contribute to reducing storage overhead and improving storage efficiency in cloud environments.

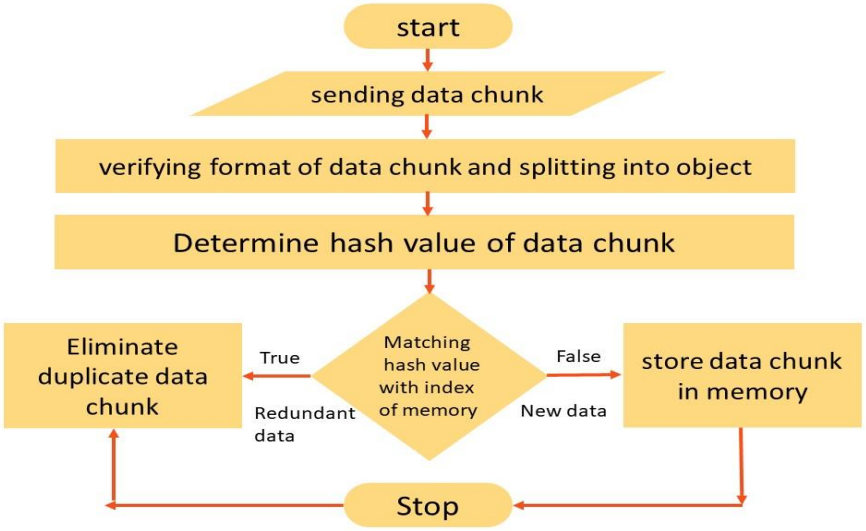


Figure 2.1: (a,b&c): Data De-duplication [28]

Figure 2.1 in the research portrays the intricacies of data deduplication. It encapsulates the process through three key elements: (a) Data Input, signifying the initial data influx into the system; (b) Deduplication Process, showcasing various stages such as chunking, hashing, and duplicate identification; and (c) Deduplicated Data Output, illustrating the storage of unique data along with potential metadata or pointers. This visualization serves to elucidate the methodology employed in minimizing data redundancy and enhancing storage efficiency within the deduplication framework.

A deduplication flowchart typically represents the process of identifying and eliminating duplicate data within a system. The flowchart begins with the input of data, which then undergoes a comparison process. During this step, the data is analysed to identify duplicate elements. If duplicates are found, a decision point is reached, leading to the removal of redundant data. After deduplication, the flowchart might involve storing the cleaned data in a database or another storage system. This process ensures that only unique and non-redundant data is retained, optimizing storage space, improving data accuracy, and enhancing overall system efficiency. By visually representing these steps, a deduplication flowchart provides a clear and structured outline of the data cleaning process, aiding in understanding and implementation for various applications, such as databases, cloud storage [27].

Data de-duplication, sometimes known as Dedup for short, is functions that can assist minimize the cost of duplicate data storage. When Data De-duplication is enabled, it optimizes free space on a volume by evaluating the data on the volume and looking for duplicated portions. Duplicated portions of the volume's dataset are stored just once and (optionally) compressed to save space. Data de-duplication reduces redundancy while maintaining data authenticity and integrity.

Data de-duplication is a procedure that eliminates redundant data copies and dramatically reduces storage capacity requirements. De-duplication can be performed as an inline procedure as data is written into the storage system and/or as a background operation to remove duplicates after data is stored to disk.

The performances for de-duplication operations are small since it runs in a separate efficiency domain from the client read/write domain. It runs in the background,

regardless of which application is running or how the data is accessed (NAS or SAN). De-duplication savings are preserved as data travels - when it is copied to a disaster recovery site, backed up to a vault, or moved between on-premises, hybrid cloud, and/or public cloud.

Chunking is the process of dividing a data stream into several pieces. When the chunk size is high, the cost of computation is reduced, but the result of deduplication may not be immediately apparent. When the chunk size is on the tiny side, the cost of computation is high, and the impact of deduplication is noticeable.

2.2.1 Methods of Data Deduplication

The data gathered through various sources and the emergence of the IoT has significantly increased the volume of data from petabytes to yottabytes, therefore necessitating the cloud computing paradigm in order to process and store data. The duplicated sections of the dataset are stored once along with being subjected to optional compression to free up even more space. It is also beneficial in ensuring veracity along with maintaining data integrity. [43] There are various methods of data deduplication such as inline deduplication, post processing duplication, source deduplication, target deduplication and client-side deduplication.

There are two approaches that may be used to remove unnecessary deduplicate from material. [44]

1) Deduplication In-Line.

Due to the fact that it is processed inside a reinforcement framework, inline deduplication simplifies the information. When information is maintained in contact with reinforcement accumulating, it is possible to eliminate instances of duplication. Although inline deduplication needs less stockpiling of reinforcements, it might still result in bottlenecks. The capacity exhibit provider recommends that their inline data deduplication solutions have their output twisted off in order to achieve high throughput.

Inline deduplication is a widely prevalent method that comprises deduplication and compression where data reduction takes place before the incoming data is written to

the stored media. Inline deduplication is essentially the removal of redundancies from a given data along with being a software defined storage solution or a storage controller that is in control of the places and the processes through which the data is saved and secured. The Inline deduplication method takes account of the entirety of data going through the tool and is scanned, deduplicated and compressed in real-time. Additionally, inline processing is also found to reduce the raw disk capacity that is needed in the system.

It takes place because the un-deduplicated and uncompressed dataset in its original size is never written to the disk. Therefore, the write operations that are executed are also comparatively lower thereby reducing the wear on the disks. However, it can also be observed that in inline deduplication the process significantly slows down the data backups that eventually is found to impede the entire process. This eventually reflects the fact that the result will thereby be devoid of any redundant or inefficient data. Inline deduplication is found to rely on the processes that exist between the data origin servers and the data backup destinations.

2) **De-duplication After Processing**

Simultaneously, post-processing data duplication is the process where the data at first is written to the storage media which is then followed by the analysis of duplication along with identification of any scopes for compression opportunities. The deduplication and compression is executed only after the data is securely stored in the storage device. In addition to this, in the process of post-processing data duplication the initial capacity that is required is somewhat related to the raw data size. Simultaneously, the optimised data is then saved back to storage media. It is done with relatively lesser space requirements in comparison to that of before data reduction.

Post-processing dedupe is a 735 synchronous reinforcement operation that eliminates repeated data after it has been maintained in contact with capacity. The data that has been entered more than once is removed, and it is replaced with an indication that is positioned toward the principal focus of the square. The post-processing method provides customers with the flexibility to dedupe certain remaining jobs at hand and the speed to quickly recoup the most recent

reinforcement without requiring water. The trade-off for this is a larger reinforcement stockpile limit than would be required with inline deduplication [45].

Post-processing data duplication is identified as an asynchronous backup process that is beneficial in the removal of redundant data after it is successfully written to storage. This process provides the user with enough flexibility and independence towards deduping specific workloads along with efficient recovery of the most recent backup. The post-processing data duplication is found to utilise the latest backup and is therefore found to take up more disk space in comparison to other deduplication processes. However, the post-processing data duplication takes a relatively lengthier processing time because of the fact that data is identified prior to the removal of the duplicate data from the storage unit.

3) source deduplication

Source deduplication, also known as client-side deduplication, is a data deduplication technique that occurs at the source of data generation or transmission. In this approach, data deduplication processes are performed on the client or source device before the data is transferred over the network to the storage destination, such as a cloud server or backup appliance.

This technique involves identifying duplicate data blocks or chunks within the data stream at the source device and eliminating redundant copies before transmitting the data to the storage system. By eliminating duplicate data at the source, source deduplication reduces the amount of data transferred over the network and stored on the destination storage system, leading to significant savings in bandwidth and storage capacity.

4) target deduplication

Target deduplication is a data deduplication technique that occurs at the storage destination or target device, such as a backup appliance or storage array. In contrast to source deduplication, which eliminates duplicate data at the source before transmission, target deduplication identifies and removes redundant data after it has been transferred and stored on the destination storage system.

In target deduplication, data deduplication processes are performed on the storage device itself, where duplicate data blocks are identified and eliminated based on predefined algorithms or patterns. This approach allows organizations to achieve data reduction and storage optimization benefits without requiring changes to the client or source devices.

5) client-side deduplication

Client-side deduplication, also known as source deduplication, is a data deduplication technique that occurs at the source or client device before data transmission or backup. In client-side deduplication, duplicate data blocks are identified and eliminated locally on the client device before transferring the unique data to the storage destination.

2.2.2 Data Deduplication strategies

Primarily, there is the record level, the square level, and the byte-level method, and each of them may be improved for increased storage capacity.

- File-level data deduplication strategy: This strategy functions at the file level and not at the sub-file level or the block level. File-level data deduplication is a technique used for data optimization. This helps in eliminating redundancy at the file level. This is what helps this strategy significantly save storage space and improves the efficiency of data storage. This strategy first identifies the duplicate files and then retains only a single instance of each unique file. The duplicates are replaced as references and pointers to the original file. The duplicate files are identified across the whole storage system. The duplicate files are identified regardless of their location or format.

  This technique is particularly effective where the files are frequently duplicated. It is also effective in an environment where many similar files are stored. For example, it will be very effective to use a file-level data de-duplication strategy in file servers or data repositories [46]. The major benefit of file-level data de-duplication is that it helps in reducing storage space. In addition to that, this technique also helps to reduce backup windows, improving backup and restore performance. This involves only unique files,

which makes the backup of files faster and reduces the recovery times of the files.

These benefits help to reduce the corruption of files as the number of files gets reduced. This definitely enhances the entire data management system. The two steps used in this technique include

1. The system scans the storage environment, which includes analysing the metadata and the duplicate content files. Metadata contains details like names, sizes, creation dates, and more attributes of the file [47]. The Metadata helps to differentiate between two or more different files. The analysis of the content involves an actual data examination within the files.
2. The identification of the duplicate files is followed by keeping one single copy of the file as the reference file and the other duplicate files are saved as pointers or references to the primary file [48]. This gives easier access to duplicate files with the help of pointers and clearly saves storage space.

- Block-level data deduplication technology: This technique is different from the file-level de-duplication technique as in this; the duplicate file is identified at the granular level. These are called “data blocks”. The data from different files are broken into blocks to identify duplicate data. The identified duplicate data is then replaced with pointers or references to the single instance of the block [49]. The three main benefits of this technique include saving storage space, reducing backup windows, and enhancing data transfer speeds. The data in this technique is stored in fixed or variable-sized blocks. The sizes of these blocks range between a *few kilobytes to several megabytes*.

  Each block identified in this technique is processed individually and the unique hash value for each block is calculated. This hash value represents the data within each block and hence serves as a fingerprint for accessing the data. The significant steps in this data deduplication technique are:

  1. The data from the files are broken into blocks after a thorough scanning

of the files.

2. The hash values are assigned to each block, which helps in easy access to these data. This helps to find the duplicate data in these files.
3. The hash value brings forward the duplicate data and these are then replaced with pointers or references to the single block file. This block is called the *"reference file".*

This technique helps in making the storage process efficient. Organisations can reduce storage space by eliminating the identified duplicate files. Organisations often use this method to store higher amounts of data in the same storage system. This technique also helps to have an efficient backup and restore system [50]. This happens because this technique only uses unique blocks and these are transferred and stored as it is. This makes the backup time lesser and creates shorter backup windows.

- **Block-Level Innovation**

Modifications made on the inside of the file will result in the whole document having to be stored. PPT and other documents may need to undergo minor adjustments to their fundamental information. For instance, if a page has to be updated to display the most recent report or the dates, this may need a complete restore of the archive. The block level information de-duplication technology saves just one version of the paper and the subsequent portion of the differences that have been made between versions. The file-level innovation, which is often under a 5:1 compression ratio, whereas the block-level storage innovation may pack the information limit of 20: 1 or even 50: 1

- **Evacuate file level innovation**

File-level information de-duplication technology, the record is extremely little, and the rehashing of the information by the designated authority takes practically no time to calculate. Because of this, the method for expulsion has very little impact on the execution of reinforcement. Due to the fact that the file is little and has a low recurrence level, the report level handling load needed to evacuate the innovation is also comparatively modest. A less impact on the amount of time required for

recovery. Remove the technical need to "reassemble" the information square by using the square level essential file coordinating square and the information square pointer. The record level innovation consists of a one-of-a-kind archive storage and highlighting the document pointer, which significantly reduces the amount of time required to rebuild.

- **Cloud Storage Mechanism**

Every cloud has a certain amount of storage, so if start uploading duplicate information, the storage will be lost, and dealing with data redundancy will become a major issue. Researchers have been investigating numerous techniques to combat this, and data deduplication is the best answer. A method called data deduplication was developed to improve storage [77]. Different cloud service providers, including Dropbox, Amazon S3 and Google Drive, now use this strategy. Data duplication is prevented by making sure it is never uploaded to the cloud more than once.

A. As the amount of digital data grows, so does the need for greater storage space.
B. Traditional solutions don't have any built-in protection against duplicate data being saved up.
C. Data De-duplication is critical for removing redundant data and lowering storage costs.

The quantity of data generated is growing exponentially in quickly developing digital age. The demand for more storage space has grown as more areas of life, from social media interactions to business transactions, are becoming digitalized. This article looks at how inadequate present storage capabilities are for keeping up with the rate of expansion in digital data and the significance of finding a solution.

- Delete this line, comment solved.

- A Partial Solution: The increased need for storage space has a partial solution in the form of cloud storage. Cloud service providers can offer scalable storage options to consumers and businesses by utilising the enormous capabilities of data centres. This method, however, has its own set of drawbacks, such as worries about data privacy, security lapses, and dependence on outside sources [9]. Additionally,

the cost of storing significant amounts of data on the cloud can rise significantly, particularly for long-term retention.

- Explosive Growth of Digital Data: The internet's rising use, the widespread use of smartphones, and the rise of connected gadgets have all contributed to the digital revolution's data explosion. The amount of digital data is always growing because of all online interactions, transactions, sensor readings, and media uploads.

- New Technologies for Data-Intensive Systems: The problem with storage is made worse by the emergence of data-intensive technologies like artificial intelligence (AI), machine learning (ML), and big data analytics. Massive datasets are needed for these applications in order to build models and gain insightful knowledge. Additionally, the growing use of virtual reality, augmented reality, and high-definition multimedia content puts extra pressure on storage infrastructure by necessitating higher capacity and quicker data retrieval.

The lack of storage capacity is becoming an urgent issue as the digital world develops. Finding scalable and effective storage solutions is urgent given the exponential growth of digital data and the rising demand for data-intensive applications. While cloud storage provides a partial solution, research into next-generation storage systems is necessary to make sure that the storage infrastructure can sustain the ever-growing digital world [11]. It can fulfil the increasing need for storage space and unleash every advantage of the digital age by making investments in technology development and promoting innovation.

The problem of redundant information has grown significantly in importance in the era of expanding digital data. Traditional storage solutions frequently do not have built-in duplicate data management tools. The significance of data deduplication in eliminating redundant data and lowering storage costs is highlighted in this article.

Duplicate data refers to information that is identical and spread across different locations in a storage system. It may be caused by a number of things, including user error, system backups, or data replication procedures [13]. Duplicate data not only takes up valuable storage space, but it also drives up prices, slows down data retrieval, and uses resources inefficiently.

Hard disc drives (HDDs) and solid-state drives (SSDs), two common types of traditional storage, lack built-in techniques for locating and removing duplicate data. Organisations can considerably reduce their storage needs by getting rid of duplicate data. However, ensuring that only one copy of each piece of information is stored, data deduplication increases data efficiency. Enhancing data integrity means reducing duplicate data [14]. Duplicate data can cause conflicts and inconsistencies, jeopardising the accuracy and dependability of data that is kept. Disaster recovery procedures might be hampered by duplicate data since it increases backup and restore times. In today's data-driven world, adopting data de-duplication is essential for effectively managing and maximising the value of digital data.

2.2.3 Process of Data

A method known as "data deduplication" may be used to get rid of multiple copies of data that is repeated. You may also know it by the name Single Instance Storage. There are two distinct methods of deduplication, which are referred to respectively as deduplication at the file level and at the block level **[50].** While deduplication at the file level takes into consideration the whole file, deduplication at the block level applies deduplication to data blocks using hashing methods.

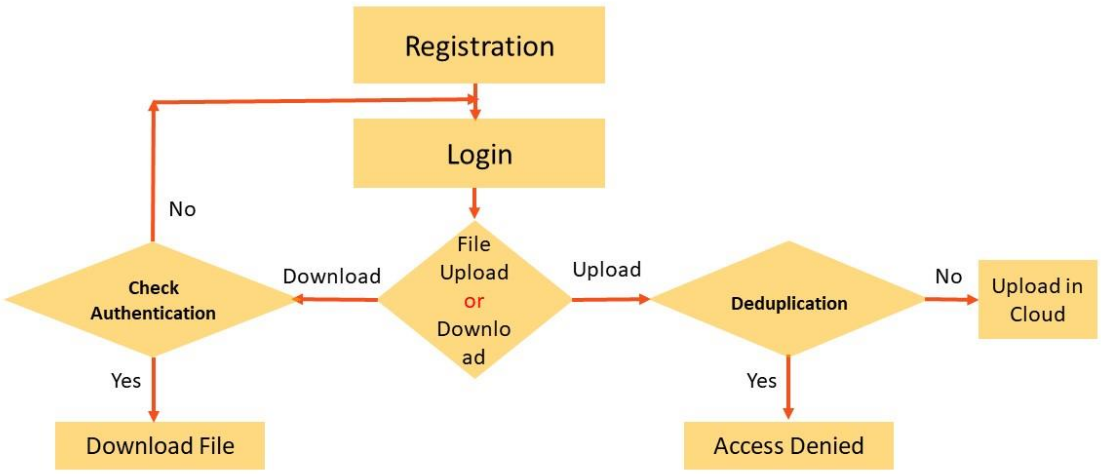


Figure 2.2: Deduplication Flowchart [51]

The figure 2.2 deduplication flowchart effectively displayed the process of data optimization through applying deduplication procedures. The procedure starts with the registration process from the end of users. Users provide various primary and well-organised information about themselves or their organisations in order to register themselves into the cloud storage system. The successful registration takes them to the login page of the cloud storage system. The users are required to provide their login id and password in order to access their data stored in the database. The login id and password is used to ensure the safety and privacy of all the stored data. However, if the registration process of the user fails then the user is asked to re authenticate their credentials and basic information. The successful login using the correct credentials take the users to the upload and download section. Downloads of the stored files require authentication from the system. Users can download the asked files if they are authenticated to do so. However, if the user wants to upload a file in the cloud storage then the duplication of the file will be checked. The access is denied if any kind of duplication is found on the provided file. Cloud storage systems grant the permission to upload any new file if no duplication is found on the provided file.

2.3 Purpose of Data Deduplication

It is crucial to eliminate duplicate data within a dataset for efficient management of data and save storage space. Therefore, it can be said that data De-duplication helps enhance the integrity of the data while improving the system's performance as well. In order to get an in-depth picture of the significance of data De-duplication, here are some key points explained in details:

1) **Optimization of System Storage:**

After reviewing other studies on this subject, it has been understood that not only duplicate data takes up unnecessary storage space, but also hampers the overall system performance. Data De-duplication identifies duplicate data and files in the device, and removes them to make space for other important data. Examples of data De-duplication in real world scenarios can be found in backup systems, archives, and cloud storage. These services use data De-duplication to prevent data redundancy while improving the data retention capabilities of itself.

2) Bandwidth Conservation:

Bandwidth conservation becomes a key factor when data is to be transferred across domestic networks. It also becomes crucial while data backup to different locations (offsite). The Data Deduplication comes in use in this case selectively remove repetitive data prior to the transfer. This is done so that the data that is to be transferred is reduced in size, and only takes up space that is crucial for the core dataset. However, this also helps in faster transfer of the data, lower bandwidth needed for the transfer of data, and lesser network traffic.

3) Data Governance Regulations and compliance of them

Government has placed several strict regulatory compliance measures on companies and industries regarding data handling. In such cases, data deduplication comes into play by helping companies meet most of these regulations. Additionally, it also helps in "data tracking" efficiently, and helps to follow the data governance practices as prescribed.

4) Data integrity and Data loss:

Data De-duplication can improve the integrity of the data and confirm only one version of each data to exist in the data set It is important to avoid any sort of duplication of data as they can cause errors and inconsistencies. If in any circumstances, there is data loss, data de-duplication makes the data recovery process much simple. It also ensures that there are less risks of data corruption and faster process of system restore.

The expenditures that are connected with duplicated data may be reduced by storage managers with the assistance of data de-duplication. When dealing with large datasets, it is common to find a significant amount of duplication, which drives up the cost of storage. As an example:

- It's possible that different users' file sharing includes several copies of the same or similar files.
- Virtualization guests may often be almost exactly the same from one VM to the next.
- There may be some variation from one day to the next in the backup snapshots.

The dataset or the workload on the volume will determine the amount of space that can be saved thanks to data de-duplication. High-duplication datasets have the potential to reach optimization rates of up to 95%, which would result in a 20-fold decrease in the amount of storage space required. The following table provides a summary of typical cost reductions that may be achieved by de-duplication of different categories of material:

Table 2.1: Data De-duplication Scenario & Typical space savings

Scenario	Content	Typical space

		savings
User documents	Office documents, photos, music, videos, etc.	30-50%
Deployment shares	Software binaries, cab files, symbols, etc.	70-80%
Virtualization libraries	ISOs, virtual hard disk files, etc.	80-95%
General file share	All the above	50-60%

2.4 Chunking Algorithm

Chunking is referred to as the process of splitting file into smaller units where efficient chunking is one of the key elements that provides an estimation of the deduplication performance. Chunking is important in certain applications such as data compression, data synchronisation, as well as data duplication as it helps in determining the duplicate detection performance of the system. Subsequently, in the perspective of the cloud storage ecosystem and about data duplication chunking is of two types that are fixed size and variable size. The chunking process is beneficial in breaking the data input stream into smaller pieces or chunks where the chunking method is the first stage of the deduplication system. A chunk is the largest physical disc unit dedicated to storing database server data.

Chunks give managers a much larger unit to work with when allocating disc space. An individual chunk can be up to 4 TB in size. The maximum number of chunks allowed is 32,766. If you upgraded from a version prior to version 10.00, you must perform the on-mode BC2 command to enable the maximum chunk size and maximum number permissible otherwise, the maximum chunk size is 2 GB.

2.4.1 Storage areas made up of chunks

Dbspaces, or database spaces, act as logical storage containers in database systems, consisting of chunks. Chunking divides the storage into manageable parts, optimizing storage utilization and enabling flexible data management. In case of corruption, only the affected chunk is impacted, minimizing the effect on other data. Blobspaces are designated for large binary objects like images and videos. Chunking breaks down these objects, enhancing data integrity and recovery. Managing large binary data becomes more efficient as chunking ensures easier storage and retrieval Segregated Buffer spaces store diverse data types within a single database, categorized based on different criteria. Chunking allocates fixed-sized units, facilitating easy access and parallel processing. It enables efficient storage utilization and enhances database performance. Temporary spaces handle temporary data, aiding query processing and sorting. Chunks store specific parts of temporary data, allowing seamless management and deletion when data is no longer needed. These specialized buffer spaces store only temporary data, like intermediate results. Chunking optimizes storage by predetermining chunk configurations.

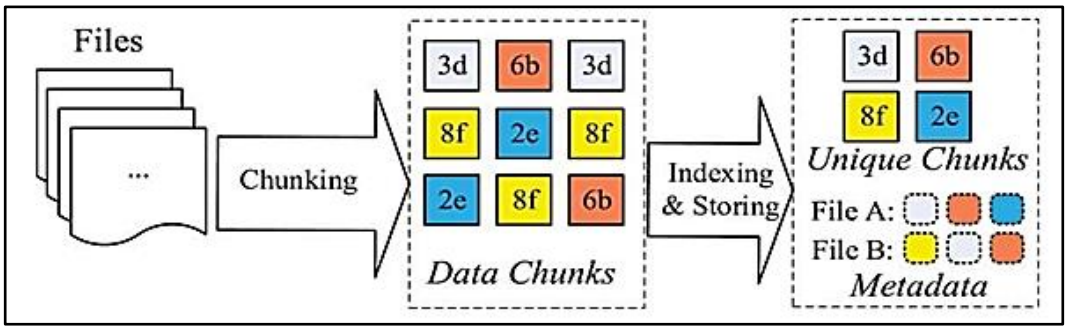


Figure 2.3: Chunking Algorithm [66]

Data deduplication is an emerging technology that involves the introduction of reduction of storage use and is an important way of handling data replication in the cloud storage mechanism. It can be mentioned here that data deduplication involves three basic components that are chunking, hashing, and comparing hashes in order to detect redundancy. A chunking algorithm is considered the first step in achieving efficient data duplication ratio and throughput, certain unique hash identifiers are implemented to draw a comparison between the chunks between the current to that of the previously stored ones.

2.5 Hash Value (HV)

A hash value is identified as a numeric value of a definite length that uniquely defines data. The hash value generally represents a large range of data in the form of much smaller numerical values in order to make it eligible to be used with digital signatures. The utility of hash value is significantly higher than in comparison to the original larger value and is important in verifying the integrity of the data that has been transmitted through non secured channels. Generally, data is hashed at a definite time along with ensuring its value is protected at the same time.

Different hash function values are allocated to various slices or chunks of data and after comparing a hash value (HV) with all other slices, the updated hash values are returned. This procedure is reiterated until the value convergence of assignment to a state of no change. A numeric number of a predetermined length that may be used to uniquely identify data is referred to as a hash value. Hash values are employed in digital signatures because they can represent enormous quantities of data with much smaller numeric values. This makes them useful [40].

Hashes are generally identified as the output of a hashing algorithm where the primary objective of these algorithms is to produce a unique, fixed-length string – the hash value, for a given piece of information or data. The hashing algorithm prevents the reconstruction of a file's content and therefore, validates and evaluates the content of two different files along with maintaining privacy and without acquiring any information about the contents. Hash values are significant to security searches and are important in evaluating the queries related to a particular dataset over an existing network, it also helps in the early identification of threats.

A hash value (HV) usually requisites a particular number of bits, and when subsequent chunks of data search for and locate chunks with the same hash value; the chunks are viewed as duplicate data and aren't kept in the data de-duplication (DD) procedure. If the hash value (HV) is unique and not existing among previously recorded values, the hash value is saved, and the matching data chunk is examined and saved in databases (DB).

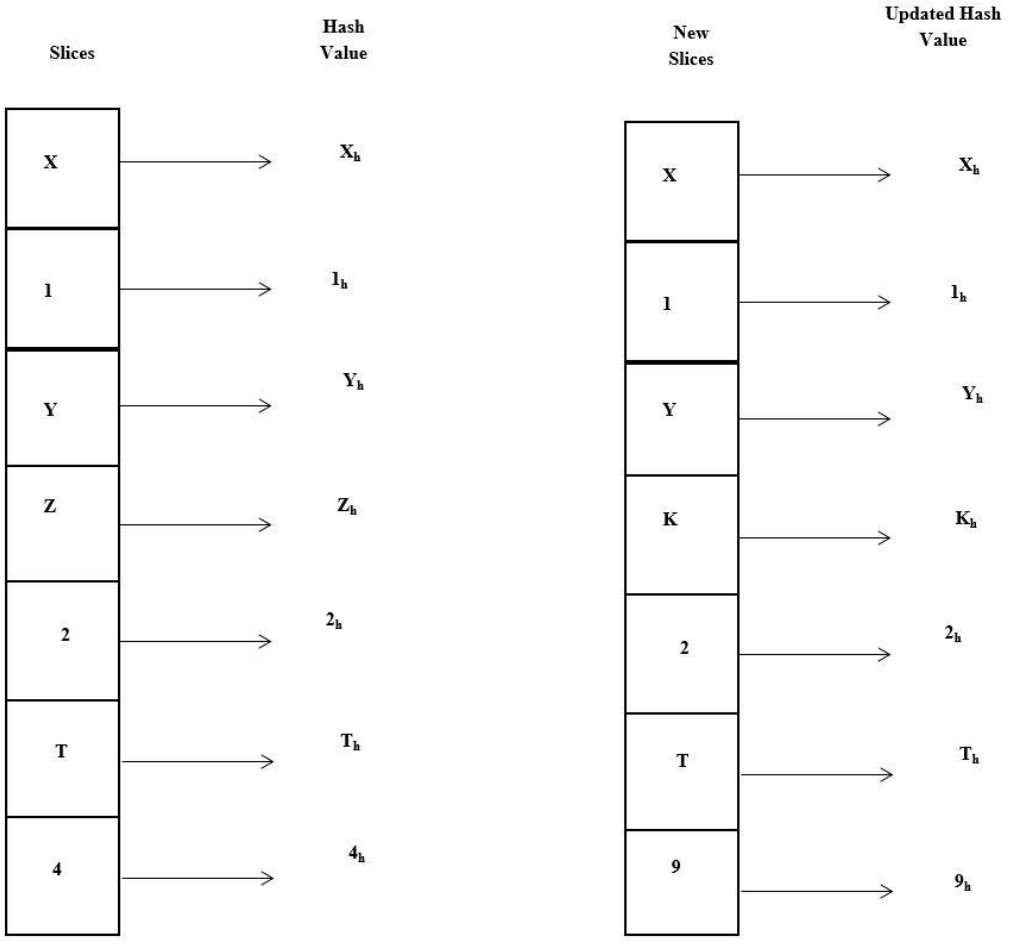


Figure 2.4: Hash value

Cloud storage has evolved as one of the leading options to store huge amounts of data; however, the hash value is also the representation of a longer document from which it was computed. The contents of a file is processed through the implementation of a cryptographic algorithm where a unique numerical value is generated and identified as a hash value. Hash values are important as they can be used to assess data of various sizes into a limited fixed size value. Hash values are deterministic along with being efficient in adapting to any change in the input thereby incorporating it in the output.

2.6 Dynamic Prime Chunking

The process flow of the chunking method, in addition to its primary and essential qualities. Dynamic Prime Chunking is a sophisticated data management technique designed to optimize storage efficiency and enhance data retrieval processes. Unlike traditional chunking methods, DPC dynamically adjusts the size of data chunks based on the content being processed. This adaptability ensures that chunks are of optimal size, preventing both underutilization and excessive fragmentation of storage space. By intelligently resizing chunks according to the data's nature, DPC improves storage utilization, accelerates data access, and minimizes storage wastage.

2.6.1 Dynamic Prime Chunking Design

The Dynamic Prime Chunking does not have a fixed size of sub problems, or chunks, and reduces computational cost. They are subjected to dynamic changes that depend on various heuristics. In simpler words, those algorithms can modify the size of the chunks depending on various factors, including the input number's properties and computational resources available onsite.

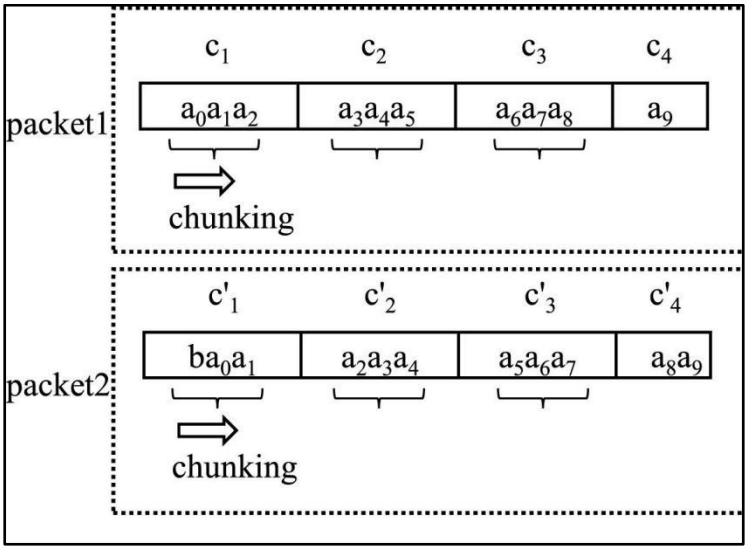


Figure 2.5: Fixed size chunking of data packet

Dynamic prime chunking algorithm aims to maintain a balance between memory usage of the data, and the "computational efficiency" [52]. Breaking the problematic bigger chunk into smaller chunks will dynamically reduce their size, making the processing much more efficient, and also reduce memory space.

Step 1: Data Input Stream

Strat from I, I is the initial byte position of the data input string.

Step 2: Calculate the dynamic window size dw based on prime number.

Step 3: Finding the maximum byte position.

M is threshold value if, Chunk breakpoint determine the following two condition

1. The interval [I, N] is empty, or the value of M is greater than the values of all bytes in the interval.

2. The value of M is not less than the values of all bytes in the interval [O, C]

Step 4: Declaring chunk boundary.

Return C as breakpoint I' is first byte of the remaining input string.

The version of AE that uses the dynamic prime chunking technique has been made better. DPC is primarily applicable to two crucial qualities, namely position and value. As can be seen in Figure 2.5, the DPC design process consists of four distinct components. First, start by reading the data input stream coming from the source. Begin at point I, where I is the beginning byte location of the data input stream. Start from there. Following this, we go on to step 2 of the process, where we use steTp 3 to compute the size of the dynamic window (DW) using prime integers. DPC makes use of two windows: one with a configurable size, and another with a dynamic changing size. The algorithm decides whether the lowest or maximum value of the input stream is the maximum value or the maximum value to use as the threshold (M). The procedure will decide what the threshold value is, and it will always be the highest or most extreme number. The third phase consists of determining the maximum value for a byte and locating the border of a chunk based on the two requirements that are listed below:

(1) To ensure that the interval [I, N] is also empty, or that the highest threshold value of M is greater in significance than any of the byte values included inside [I, N].

(2) In the dynamic window with a changeable size, the extreme value M must be greater than the value of every byte that falls between the coordinates [O, C].

In order to ensure that the highest byte point is represented as the maximum local value, it is necessary to assess whether or not the first byte satisfies the requirements described above, which are related with a threshold value. On the other hand, the maximum byte location has been established, and DPC has declared the byte that is most to the right to be the chunk breakpoint for the right-side window [52]. The algorithm will return the breakpoint location C once the chunk boundaries have been specified in step four once they have been declared. After that, the sequence that begins at the first byte location continues with the letter I. Repeat the methods from the previous section until you locate the very last boundary of a chunk in the incoming data stream.

2.6.2 Workflow of DPC

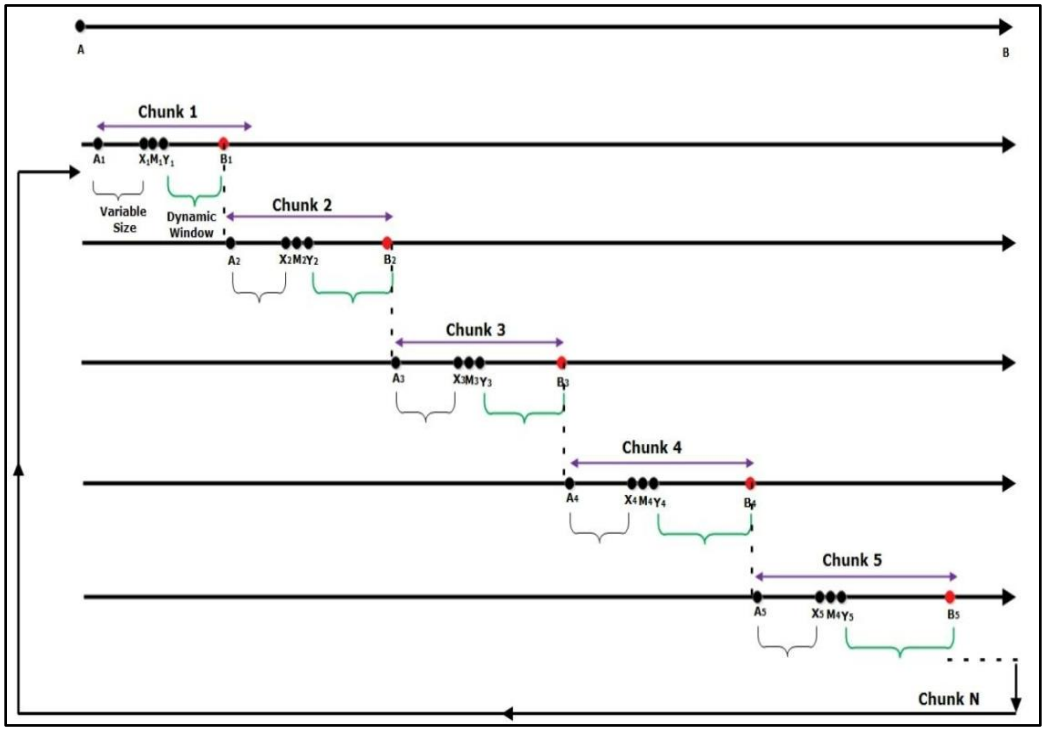


Figure 2.6: The workflow of DPC algorithm

In the example shown in Figure 2.6, the first byte position, which is indicated by the letter A1, continues to advance in the correct direction until it reaches the end of the

byte position B. The threshold value M1 is used to partition the whole data stream into several parts. The location of the leftmost byte, which comes before the threshold, must thus be a window of variable size. M1 refers to the gap that exists between each successive byte, beginning with A1 and ending with X1. As the right motion, the byte position is moved forward once again, this time from Y1 to B1. As stated in Chunk 1, DPC is also a dynamic window with an adjustable width and height. The precise procedure is carried out from chunk 1 all the way through chunk N. The reason why there is a dynamic window is because the point at which the chunks split is constantly changing in size. AE, on the other hand, just the left side has a varied size; the right section remains the same throughout. As a result, the effects will be felt greater in AE. In order to circumvent this problem, the DPC technique that we've presented makes use of a variable window size. This helps to get rid of the lengthy chunk sequence and boosts the deduplication throughput.

2.7 Content Defined Chunking (CDC) Algorithms

The term "content-defined chunking" (CDC) refers to a technique for dividing files into chunks of varying lengths, with the cut points being determined by the inherent characteristics of the files themselves. In contrast to chunks with a set length, chunks with a variable length are less susceptible to byte shifting.

Due to its strong redundancy detection ability, Content-Defined Chunking, also known as CDC, has been playing a pivotal role in data deduplication systems for the better part of the last 15 years. Existing CDC-based techniques, on the other hand, result in a significant increase in the amount of CPU overhead. This is because the chunk cut points are determined by calculating and evaluating the rolling hashes of the data stream byte by byte.

The technique of chunking divides a single file into many smaller files that are also called pieces. Chunking is significant in some applications because it impacts the performance of the system in terms of duplicate detection. Some examples of these applications are remote data compression, data synchronisation, and data deduplication. The term "content-defined chunking" (CDC) refers to a technique for dividing files into chunks of varying lengths, with the cut points being determined by the inherent characteristics of the files themselves. In contrast to chunks with a

set length, chunks with a variable length are less susceptible to byte shifting [53]. As a result, the likelihood of discovering duplicate chunks both inside a file and across files is raised as a result of this. However, in order to locate the cut spots, CDC techniques need extra calculation, which might be computationally costly for particular applications.

A content-defined variable-length chunking method [52] is offered as a solution to the issue of byte shifting in fixed-length algorithms. This algorithm reads files as a data stream and creates chunks according to the Rabin fingerprint of a window data. It has been suggested that the Rabin method use two divisors instead of only one in order to overcome the problem that it is difficult to locate the cut-off point. Of the two divisors, one is simple to do and the other is the complete opposite. The most difficult divisor needs to be used right from the start when trying to locate an appropriate stopping point. If the data cannot be fulfilled within a lengthy data period, then it will be replaced by the easier one in order to prevent huge chunks of data wherever possible. In addition to this, the Rabin fingerprint suffers from an issue known as size variation of pieces. A technique known as LMC, or Local Maximum Chunking, has been suggested as a solution to this problem **.** The method comes to the conclusion that a cut-off point should be established if the greatest value of a window's data is located in the centre of the window. This allows the programme to avoid the time-consuming process of generating the Rabin fingerprint. At the same time, the size of the chunks may be restricted because the window size can be set, and the distribution of the chunk size is reasonably constant. This is because the window size can be set. AE [48] and RAM [35] are two techniques that have been presented in order to expedite the process of validating the window data. Increasing the speed of chunking may be accomplished by modifying the validation technique of window data; this process will be discussed in more detail later on. In addition, the concept of parallel computing is used to the algorithms that are used for data chunking in order to make the process move more quickly.

2.8 Types of Chunking Algorithm

2.8.1 Rabin chunking Algorithm

Input: input file,file; default value,Value;length of sliding window, W; Output: cut point,I;

```
function RabinChunking(file,Value, W)
i=1
index=0
while(byte=readByte(file))
array[index%W+1]=byte
if array.length>=W then
else
if hashValue(array,index, W)==Value then
return i
end if
continue
end if
i=i+1
end while
end function
```

The Rabin chunking algorithm is also popularly known as "Rabin Fingerprinting Algorithm" which was developed back in 1981, by Michael O. Rabin. This system is very helpful when it comes down to breaking the data into smaller, and fixed size chunks. This breakdown of the data depends on their data content. Therefore, it is clearly suggested that it is a technique used in de-duplicating data.

This algorithm apparently creates a "rolling hash function". This function then proceeds to calculate each of the data block's hash value, which is most popularly known as a fingerprint of the data as well [54]. This fingerprint plays a crucial role in identifying duplicate data chunks on the data, which are similar to one another. Therefore, it is understood that any small change made in the data itself can result in different hash values.

Sliding window approach is used in this type of algorithm to perform chunking. An initial data window starts the process, and calculates that window's hash value at the same time After the calculation is done, the algorithm shifts the window position by one byte, only to calculate the hash value for the new position of the window. The goal of this is for the hash value to satisfy certain criteria.

The Rabin Fingerprinting Algorithm is capable of identifying duplicate data chunks within a larger dataset in a more efficient way. [56] This comes in use in the case of

backing up specific chunks of data to save space in the storage device. The Rabin chunking algorithm can compare the hash values in order to recognise the duplicate data chunks even if data blocks are somewhat dissimilar.

However, one of the biggest disadvantages of this algorithm is that it can give false results [55]. For instance, it might show the result as false positive, which can happen when coincidently, two completely different data blocks produce the same hash value, therefore they can be flagged as duplicate data. Similarly, false negative results occur when unfortunately, two of the same blocks of data show different hash values.

2.8.2 LMC Chunking Algorithm

```
Output: cut point,I;
function LMCChunking(file, W)
i=1
start=1
while(byte=readByte(file))
if byte<=max.value then
if i==max.position+w and max.position>=start+w then
end if
start=max.position+1
return max.position
else
max.value=byte
max.position=i
end if
i=i+1
end while
end function
```

The LMC, or Lesk's Measure of Cohesion Chunking Algorithm was Introduced in 1986 by Michael Lesk. It is essentially a language processing technique, which can detect meaningful chunks from a text. This technique calculates the Cohesion scores of every word present in a text . This calculation is primarily done by examining the overlap of the context of one word to its immediate next word. These contexts are a group of words in a window, which has a fixed size around the main word.

The use of this algorithm is mainly found in extracting information or parts of speech tagging, etc. The identification of valuable chunks and extracting them from a text allows in-depth understanding of the chunk's content. Thus, the LCM Algorithm can assess the relationship shared between words by analysing their context, which results in accuracy in identifying chunks.

2.8.3 Asymmetric Extremum (AE) Chunking algorithm

```
Algorithm for AE chunking Input: input file, file; size of fixed window, W; Output:
cut point,I;
function AEChunking(file, W)
i=1
while(byte=readByte(file))
if byte<=max.value then
if i==max.position+w then return i
end if
else
max.value=byte
max.position=i
end if
i=i+1
end while
end function
```

This algorithm looks for phrases, which appear to be important. This decision is based on external factors such as the high level of information of the word, in comparison to its neighbours. AE chunking algorithm reduces traffic redundancy to be more efficient. After Tokenization, the features of each word, such as syntactic patterns and parts of speech tags are computed.

The algorithm then proceeds to group words with best external features to form something meaningful. Therefore, the AE chunking algorithm group’s words that have the appearance of being informative to make a meaningful phrase, and this is in use while extracting keywords from a text or retrieving information.

2.8.4 RAM Chunking Algorithm

```
function RAMChunking(file, W)
i=1
```

```
while(byte=readByte(file))
if byte>=max.value then
if i>w then
return i
end if
max.value=byte
max.position=i
end if
i=i+1
end while
end function
```

"RAM or Rapid Asymmetric Maximum Chunking Algorithm" is a helpful approach for the identification and segmentation of handwritten text in a phrase [56]. The RAM chunking algorithm was developed so that the accuracy of the segmenting of the handwritten characters increases [54]. In order to be able to achieve this goal, the RAM chunking algorithm uses a group of image processing systems, known as "threshold-based image processing" It helps to overcome challenges posed by the overlapping strokes of the character, their irregular sizes, etc. The use of asymmetrical chunking (smaller chunk) is Done by detecting the physical features such as strokes and slants.

2.9 Secure Hash Algorithm

Secure Hash Algorithm (SHA) are a kind of cryptographic function that is used to keep data secure. It transforms data using a hash function, which is a method composed of bitwise operations, modular additions, and compression functions. The hash function then returns a fixed-length string that has no resemblance to the original. These methods are meant to be one-way functions, which means that once they've been translated into their corresponding hash values, it's almost hard to reverse the process. SHA-1, SHA-2, and SHA-3 are three algorithms of interest, each of which was built with ever better encryption in response to hacker attempts. Because of publicly publicised weaknesses, SHA-0, for example, is now outdated. [56]

SHA is often used to encrypt passwords since the server just has to maintain track of a single user's hash value rather than the actual password. If an attacker steals the database, they will only obtain the hashed functions and not the real passwords, therefore if they enter the hashed value as a password, the hash function will turn it into another string and prohibit access. Furthermore, SHAs display the avalanche effect, in which changing a few characters in an encrypted string generates a large change in output; or, conversely, vastly dissimilar sequences give comparable hash values. As a result of this consequence, hash values do not provide any information about the input text, such as its original length. Furthermore, SHAs are used to identify data tampering by attackers; for example, if a text file is slightly altered and hardly apparent, the modified file's hash value will be different from the original file's hash value, and the tampering will be rather obvious.

There are several advantages and disadvantages of using Secure Hash Algorithm-1. The primary advantage of using SHA-1 algorithm is it reduces the risks of brute force attack by the hackers. It is useful for storing the passwords, as it is a very slow process. It is also used to compare codes or files in order to identify the *"unintentional only corruptions"*. It also has the capability to replace the SHA-2 when the matter of interoperability issue is noticed with the legacy codes. However, it also suffers from various drawbacks including it is less secure as compared to other algorithms. The collision is extremely easy to find in the SHA-1. The length of the key in the SHA-1 is too short to resist the potential attacks. It is not suitable for uses other than storing the passwords, as it is slow in nature.

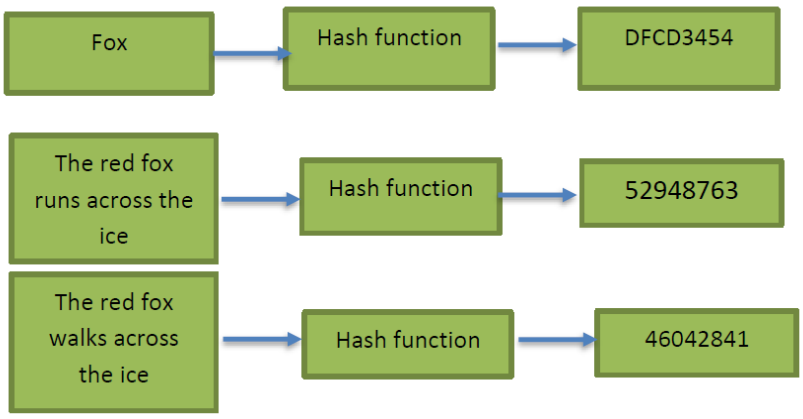


Figure 2.8: Hash function

2.9.1 SHA – 1

It is a 160-bit or 20-byte long hash-based function-based encryption technique that is used to mimic the MD5 algorithm, which has been around for a while. The NSA, or National Security Agency, conceived and developed the specific algorithm, which was intended to be part of the crucial component- Digital Signature Algorithm (DSA). Weaknesses in cryptographic methods were discovered in SHA-1; the encryption standard was eventually discontinued and was hardly used.

SHA-1 generates a 160-bit hash value or message digests from the inputted data (data that needs

encryption), which is similar to the MD5 hash value. To encrypt and protect a data item, it performs 80 rounds of cryptographic procedures. SHA-1 is used in a number of protocols, including:

- Transport Layer Security (TLS)
- Secure Sockets Layer (SSL)
- Pretty Good Privacy (PGP)
- Secure Shell (SSH)
- Secure/Multipurpose Internet Mail Extensions (S/MIME)
- Internet Protocol Security (IPSec)

SHA-1 is widely employed in cryptography applications and contexts where data integrity is critical. It is also used to index hash functions, as well as to detect data corruption and checksum issues.

The SHA-1 or the “Secure Hash Algorithm 1” is considered the cryptographic algorithm that includes the input and produces a 160-bit hash value. This hash value is called the “message digest” which usually is rendered as a kind of hexa-decimal

number that is 40 digits longer. It is also considered to be in the “US Federal Information Processing Standard" and was said to be designed by the “United States National Security Agency” [57]. The SHA-1 is presently considered to be insecure since the year 2005. The giant technical browsers which include Google, Microsoft, Mozilla and Apple have prevented accepting SHA-1 SSL certificates by the year 2017. The requirements to calculate the graphical value is included in Java where the “MessageDigest class” is utilised under the package for “java.security”.

This class offers various cryptographic hash functions, including MD2, MD5, SHA1, SHA224, SHA256, SHA384, and SHA512, which can be utilized to compute the hash value of a given text. These algorithms can be initialized using the static method "getInstance()". Once an algorithm is selected, the message's digest value is calculated, and the results are returned as a byte array. To convert this byte array into a readable format, the class utilizes "BigInteger". This conversion enables the representation of the signal, which is then further converted into hexadecimal format to obtain the expected result from the message digest.

These algorithms could be used in several forms such as:

1) **Cryptography:** The primary application of SHA-1 is to provide protection to the communication from being interrupted by parties from outside. It generates singular, irreversible and fixed size values. The data integrity can also be confirmed through the comparison of this hash value with the original hash value [57]. It also makes it easy in confirming that the data that is used is not tampered or changed with the manner during the transmission of the data.
2) **Digital Forensics:** The hash value of a file that includes the digital evidence can be manufactured making use of the SHA-1 algorithm in the digital forensics. This also helps in ensuring that the evidence has not been changed during the process of investigation using the hash value as a type of proof [58]. It also proves that the file is not altered if the hash value for the original file and the file of evidence matches.

2.9.2 SHA-512

There are multiple applications of hash functions in the digital environment. The mechanism applies to internet security, block chains and others. The hashing

algorithm constitutes a one-way program (). The primary advantage of such a type of algorithm is it cannot be restructured and decoded. Therefore, if any third party gets access to the server, the entire data remains unreadable. The Hashing algorithm holds the following properties in brief.

a. Mathematical - It maintains strict rules to design the algorithm.
b. Uniform - All hashing programs are uniform in nature. Whatever be the length of the data it produces a fixed length of output.
c. One way - Once it is created, it will be nearly impossible to decode it. Therefore, it is secure for programmers as well as users.
d. Consistent - A hashing program only one process that is compressing the given data.

SHA-512 works in the following manner -

1) ***Input Formatting*** It has an input size limitation. SHA - 512 can not execute an input of any size. The entire message constitutes three parts namely - *original message, padding bits and the size of original message.* The message will be executed as blocks of 1024 bits.
2) ***Hash Buffer Initialization*** It is already mentioned that the process works with a block of 1024 bits and collects from the previous blocks. However, it generates a problem for the first 1024-bit block, therefore, it is unable to use the result from the previous block. This problem can be solved by providing a default value to the first block in order to start the process. The intermediate results will be used in the next block. Therefore, the result should be stored somewhere for later use. This will be done by the *hash buffer.*
3) **Message Formatting** It takes one block of 1024 bits at a time and message formatting is done on it. The actual execution is done by using two things that is a block of 1024 bits and the result from the previous processing.
4) **Output** After the message-processing phase we get a 512-bit hash value for the original message. From each block, intermediate results are used for processing the next block. When the execution of the final bit of 1024 is finished, we get the result of the SHA 512 algorithm.

SHA-512 is a function of the cryptographic algorithm SHA-2, an extension of the well-known SHA-1.

SHA-512 is essentially similar to Sha-256, except that it uses 1024 bit "blocks" and accepts a maximum length string of 2128 bits as input. In addition, SHA-512 differs from Sha-256 in terms of algorithmic alterations.

A cryptographic hash (also known as a 'digest') is a kind of ‘signature' for a text or data file. For a text, SHA-512 provides a nearly unique 512-bit (32-byte) signature. The source code is available below.

This is a companion script to the SHA-256 script (which has more information). This is a reference implementation, as close to the NIST specification as possible, to aid in understanding the algorithm (section numbers relate the code back to sections in the standard); it is not at all optimized (in timing tests, using Chrome on a low-to-middle Core i5 PC, this script will hash a short message in around 0.4 - 0.6 ms; longer messages will be hashed at a speed of around 0.5 - 1 MB/sec).

Because SHA-512 is based on 64-bit unsigned integers, which JavaScript does not natively handle, it is more difficult to implement in JavaScript than SHA-256. For an optimised implementation, I've developed a long library for UInt64 operations; there would be more efficient ways of accomplishing this.

2.10 Dataset

The research utilizes two primary datasets, each offering unique insights and data characteristics. These datasets are integral for conducting comprehensive evaluations and comparisons of various cloud storage solutions, providing a robust foundation for the benchmarking process.

2.10.1 Multimedia and OS Datasets

Multiplicative-Divisive Probabilistic Congestion Control (MDPC) is a congestion control algorithm used in computer networks. It is designed to control the rate at which data is transmitted over a network to avoid congestion.

To implement MDPC, the researcher will need the following:

1. Operating System: MDPC is a congestion control algorithm that can be implemented on any modern operating system, such as Windows, Linux, or macOS.

2. Network stack: The MDPC algorithm operates at the transport layer of the network stack. It requires access to the congestion control module of the network stack to be able to control the rate at which data is transmitted.

3. Multimedia support: MDPC is designed to handle multimedia traffic, which includes audio and video streams. Therefore, the operating system and network stack must support multimedia traffic.

4. Hardware requirements: The hardware requirements for MDPC depend on the size and complexity of the network. In general, MDPC can be implemented on any modern computer hardware with a network interface card (NIC).

5. Software requirements: To implement MDPC, the researcher will need to install the congestion control module that supports the MDPC algorithm. This module can be a part of the operating system or a separate software package that needs to be installed.

To implement MDPC, the researcher needs a modern operating system and network stack that support multimedia traffic, as well as hardware with a network interface card. Additionally, he or she need to install a congestion control module that supports the MDPC algorithm.

The researchers have conducted several studies on MDPC and have found that the alternative systems that are asymmetric in nature other than the traditional systems impact the “McEliece cryptosystem”. The system of McEliece is considered to be based on the codes of “QC-MDPC” that is considered to have an extremely

interesting contribution since it has excellent performance on the limited sources and embedded systems. This code also extends the concept of lower density with a parity check that makes use of a certain matrix that checks the parity with the moderated sparse [44]. This also leads the procedure to a significantly degraded performance of error correction. The MDPC decodes certain instances that attempt to decode the complicated coding. The random parity check matrix and the random error factor are considered to be generated through the computer corresponding syndrome. It prints the instances every five seconds that are generated and distributed with the number of iterations it will be required to decode.

The MDPC codes are considered the LDPC codes of greater density than the usually adopted applications for telecommunication. This also leads to worse means of error correcting capability. However, the cryptography based on the MDPC code is not interested in correcting several errors but only a specific number of errors that ensure and the level of security that is a condition, which is satisfied by the codes of MDPC. The benefits of using the MDPC code include many benefits. The MDPC codes reduce the distinguishing problems related to McEliece, which includes the problem of decoding the codes that are linear [44]. The attacks of message against the scheme is also required in reducing the problem along with the security of providing the scheme that has the benefit of reliance on a single and well-studied problem regarding the coding theory

CHAPTER THREE

Proposed Mechanism

CHAPTER – 3

Proposed Mechanism

3.1 Introduction

This chapter presents the research methodology for the proposed model for cloud IoT environment. The efficient algorithm for constrained IoT devices was covered in detail in this chapter. It began with an overview of lightweight efficient cloud storage and its inherent difficulties, which were presented in sections 3.1 and 3.1.2. The overview of the efficient algorithm then provided in section 3.4, highlighting the two-step process of IoT device authentication and resource-efficient message encryption. The efficient AES-based algorithm's implementation pseudocode is shown, and it describes the encryption and decryption procedures as they are described in sections 3.5 and, respectively.

3.2 Design Research Methodology

The Design Research Methodology for thesis on IoT and cloud storage focuses on developing an efficient, scalable cloud storage system for IoT environments. approach involves a comprehensive examination and integration of various components. Firstly, analyse the dataset and simulation tools used, ensuring compatibility with IoT requirements. The operating system is chosen for optimal performance in processing IoT data. A significant part of methodology includes the adoption and modification of algorithms, notably Content-Defined Chunking (CDC) and standard chunking algorithms, tailored to enhance data processing and storage efficiency. The system architecture is designed with specific considerations for IoT applications, ensuring seamless integration and operation. Additionally, we assess software requirements, benchmarking standards, and design goals to ensure research meets the evolving needs of IoT environments. This methodology aims to create a

robust, adaptable cloud storage solution, addressing the unique challenges in IoT data management.

3.3 MDPC Algorithm and Difference with DPC

MDPC (Multiplicative-Divisive Probabilistic Congestion Control) and DPC (Deterministic Probabilistic Congestion Control) algorithms both aim to manage network congestion, yet they differ in their approach. MDPC operates by probabilistically increasing or decreasing the congestion window size based on network conditions, utilizing multiplicative and divisive factors to adjust the window size dynamically. In contrast, DPC employs a deterministic approach, where the congestion window size is adjusted based on predetermined thresholds and probabilities, without the multiplicative and divisive factors utilized in MDPC. While MDPC offers adaptability to varying network conditions through probabilistic adjustments, DPC provides deterministic control over congestion window size changes, potentially offering more predictable behavior in certain network scenarios.

3.3.1 Dynamic Programming with Clustering (DPC) Algorithm

The DPC algorithm combines dynamic programming with clustering techniques to reduce the computational complexity of solving optimization problems with large state spaces. It involves the following steps:

1) **Clustering:** Initially, the state space is divided into clusters based on certain similarity measures. Clustering helps in grouping similar states together, reducing the overall size of the state space. Like DPC, the state space is initially partitioned into clusters using clustering techniques. Clustering helps in reducing the complexity of the problem by focusing on smaller, manageable subsets of the state space.

2) **Dynamic Programming within Clusters:** Within each cluster, dynamic programming techniques are applied to find the optimal solution. By solving smaller

subproblems within clusters, the computational complexity is reduced compared to solving the entire problem space.

3) Inter-Cluster Communication: Information exchange or communication between clusters is facilitated to ensure coherence and consistency in the final solution. Inter-cluster communication can involve sharing boundary information, optimal policies, or other relevant data.

3.3.2 Multi-Point Dynamic Programming

Dynamic programming techniques are applied to each cluster independently to find the optimal solution within each cluster. The dynamic programming process considers multiple decision points or stages, allowing for sequential decision-making.

3.3.2.1 Inter-Cluster Communication

Similar to DPC, communication between clusters is essential to ensure coherence and consistency in the final solution. Inter-cluster communication involves sharing information about optimal policies, boundary conditions, or other relevant data. This method involves a detailed comparison and contrast of existing cloud storage systems, examining their various features and performances. The core of this methodology is the development and application of a specialized benchmarking tool designed for assessing the efficiency, flexibility, and user-friendliness of cloud storage systems. In the initial stage, the research involves gathering data about various cloud storage solutions currently available. This step includes examining the infrastructure of these systems, understanding their data organization, storage capacities, scalability, and the nature of their virtualized storage environments. This examination helps in identifying the key characteristics that impact the performance and cost-effectiveness of these services. The research

moves to a critical comparison of these cloud storage solutions. This comparison is not merely theoretical; it involves practical analysis based on specific parameters such as storage capacity, scalability, ease of access, and cost-efficiency. The focus is on how these systems manage and maintain data, their ability to scale up or down based on user requirements, and the overall user experience in terms of managing and accessing stored data.

An essential part of the methodology is the creation of a benchmarking Concept. This Concept is designed to test operational cloud storage systems, evaluating them on various performance metrics. The tests conducted using this tool are critical in assessing the efficacy of the cloud storage systems under real-world conditions. These tests are aimed at determining the systems' efficiency in data management and retrieval, their response to varying storage demands, and their cost-effectiveness. The research methodology also includes analyzing open-source cloud storage systems through code analysis to obtain reliable results. This aspect is crucial as it provides insights into the architecture and design principles of these systems, contributing to a deeper understanding of their operational efficiencies. The research methodology is a blend of theoretical study and practical evaluation, aimed at providing a comprehensive analysis of cloud storage systems and developing an effective benchmarking tool for their assessment.

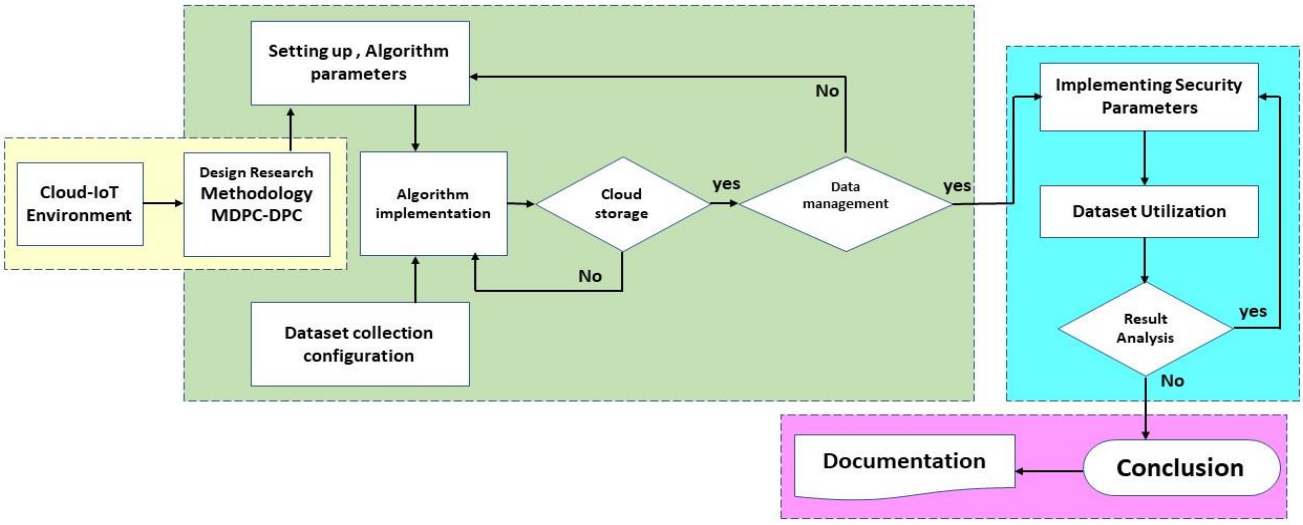


Figure 3.1: Flowchart Methodology

3.4 The Proposed System

The MDPC algorithm is a cloud storage mechanism that is designed to efficiently manage and store data in IoT environments. This algorithm is based on a probabilistic approach that enables efficient utilization of cloud storage resources while ensuring high reliability and availability of data. Multiplicative-Divisive Probabilistic Congestion Control (MDPC) is a congestion control algorithm used in computer networks to manage the flow of data packets and prevent congestion. It operates by dynamically adjusting the congestion window size based on network conditions. MDPC encompasses various subtypes or kinds, each with its specific characteristics and approaches to congestion control.

Multiplicative Increase Divisive Decrease (MIDD): In this subtype of MDPC, the congestion window size is increased multiplicatively when the network is operating efficiently and there are no signs of congestion. However, when congestion is detected, the window size is reduced divisively to alleviate the congestion and prevent further packet loss. MIDD aims to strike a balance between exploiting available network capacity and avoiding congestion.

1. Adaptive MDPC: Adaptive MDPC is a subtype that adjusts its congestion control parameters dynamically based on observed network conditions. It continuously monitors network metrics such as round-trip time (RTT), packet loss rate, and available bandwidth to adapt its multiplicative and divisive factors accordingly. By adapting to changing network conditions, adaptive MDPC aims to optimize network performance and minimize congestion-related issues.
2. Probabilistic MDPC: Probabilistic MDPC introduces randomness into the congestion control process by probabilistically increasing or decreasing the congestion window size. Instead of deterministic rules, probabilistic MDPC utilizes probabilities to adjust the window size, allowing for more flexibility and adaptability in response to varying network conditions. This approach helps prevent synchronization effects and can lead to more stable network behavior.
3. Delay-based MDPC: Delay-based MDPC focuses on controlling congestion based on the observed network delay. It adjusts the congestion window size proportionally to the measured delay, aiming to maintain an optimal level of delay while avoiding congestion. By considering delay as a congestion indicator, delay-based MDPC can effectively manage congestion in networks with variable latency.

Overall, MDPC and its subtypes provide a flexible and adaptive approach to congestion control in computer networks. By dynamically adjusting congestion window sizes based on network conditions, MDPC algorithms aim to optimize network performance, minimize packet loss, and prevent congestion-related issues. Each subtype of MDPC offers unique features and capabilities, allowing network

administrators to choose the most suitable variant for their specific networking environment and requirements.

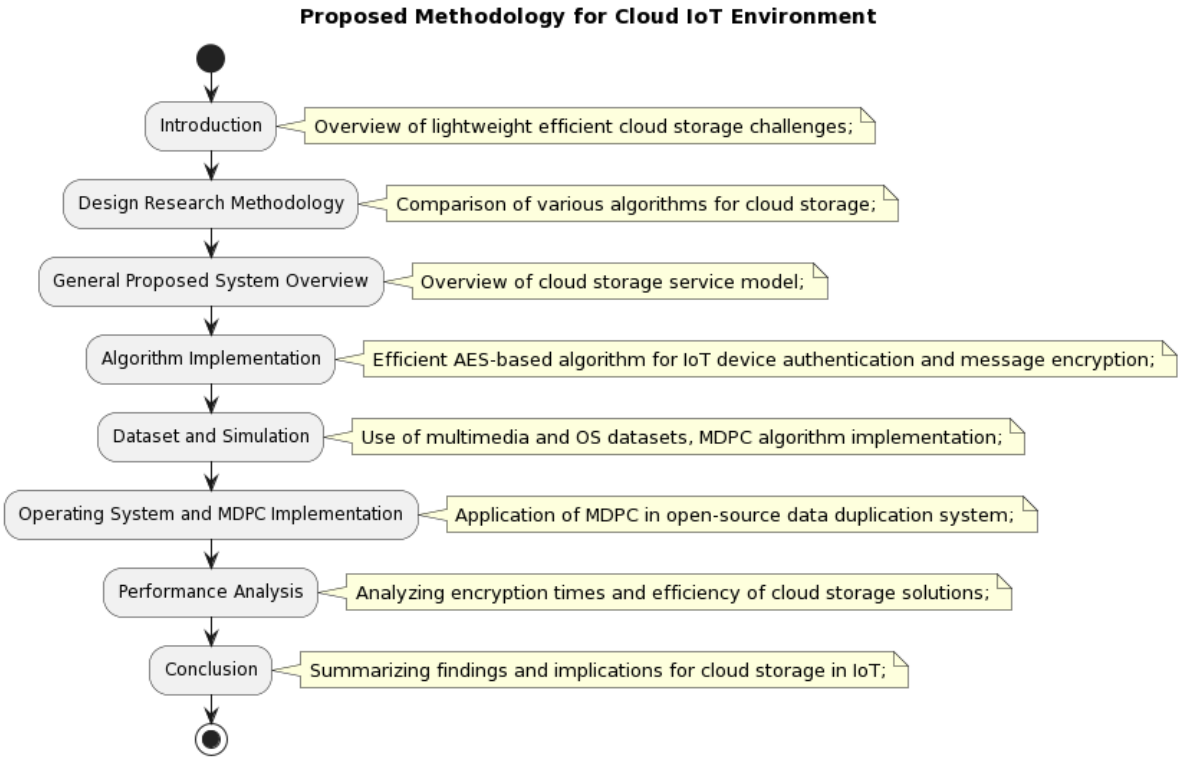


Figure 3.2: Design Research Mechanism

Programming language: MDPC can be implemented in a variety of programming languages, including C, C++, and Python [59]. The choice of language would depend on factors such as performance, ease of development, and existing code base. Required libraries: The MDPC algorithm may require certain libraries, such as (Multiple Precision Arithmetic Library) for high-precision arithmetic or OpenSSL for cryptographic operations . Configuration and optimization: The performance of the MDPC algorithm can be improved through various configuration and optimization techniques, such as parallel processing, vectorization, and code optimization. These techniques would depend on the specific implementation and hardware used. Overall, running the MDPC algorithm requires a standard computer

system with modern hardware, an appropriate operating system and programming language, and any required libraries and configurations for optimal performance.

The MDPC algorithm works by dynamically adjusting the storage capacity allocation for each IoT device based on its data usage patterns and storage requirements [73]. It achieves this by maintaining a probabilistic congestion control mechanism that ensures that the available storage resources are optimally utilized.

3.4.1 MDPC Algorithm Works in Two Phases

1. Probabilistic Allocation: In this phase, the algorithm assigns storage capacity to each device based on a probabilistic model that takes into account the device's data usage patterns and storage requirements. The algorithm calculates the probability of congestion for each device and assigns storage capacity accordingly.
2. Dynamic Adjustment: In this phase, the algorithm monitors the data usage patterns of each device and dynamically adjusts its storage allocation to ensure optimal utilisation of the available resources. The algorithm also considers the reliability and availability requirements of the data and ensures that the storage capacity allocation is sufficient to meet these requirements.

The MDPC algorithm has several advantages over traditional cloud storage mechanisms. First, it optimises the utilisation of cloud storage resources, which leads to reduced storage costs. Second, it ensures high reliability and availability of data by dynamically adjusting storage allocation based on data usage patterns. Finally, it is highly scalable and can handle large numbers of IoT devices with varying data usage patterns.

Overall, the MDPC algorithm is an efficient cloud storage mechanism for IoT environments that can help organisations reduce storage costs and ensure high reliability and availability of data.

3.5 Simulation Used

A study has been implemented that focuses upon encryption algorithms implemented by researcher Habeeb, Ahmed. (2018). Multiplicative-Divisive Probabilistic Congestion Control (MDPC) algorithm is a variant of the Additive-Increase Multiplicative-Decrease (AIMD) algorithm. It is used to control the rate of data transmission in computer networks to avoid congestion [76]. MDPC adds a probabilistic component to the AIMD algorithm to reduce the chances of congestion. In this algorithm, the congestion window size is multiplied or divided by a factor depending on the network conditions.

To perform configurations and settings of the MDPC algorithm, the researcher will use the following algorithm and code:

Algorithm (3.1): Multi-Dimensional Partial Congestion Control (MDPC)

```
//Initialize Variables
Set cwnd (Congestion Window Size) to initial congestion window size
Set threshold to initial threshold value.
Set ack_counter to 0
Set nack_counter to 0
While true
    if received_ack():
        Increment ack_counter by 1
    end
    if ack_counter ≥ threshold
        Multiply cwnd by 2      (Multiplicative Increase)
        Reset ack_counter to 0
        Reset nack_counter to 0
    end
    Else if received_nack()
```

```
12:         Increment nack_counter by 1
13:     end
14:     if nack_counter ≥ threshold
15:         Divide cwnd by 2      (Divisive Decrease)
16:         Reset ack_counter to 0
17:         Reset nack_counter to 0
18:         Set threshold to calculate_new_threshold(threshold)
19:         Send data with congestion window size cwnd
20:     end
21: end
```

Where:

cwnd be the current Congestion Window Size.

threshold be the threshold value for determining congestion control actions.

ack_counter be the count of received acknowledgments.

nack_counter be the count of received negative acknowledgments (NACKs).

Upon receiving an acknowledgment:

• If ack_counter > threshold:

cwndnew = cwnd x 2 (Multiplicative Increase).

Upon receiving a negative acknowledgment:

• If nack_counter > threshold:

cwndnew = cwnd/2 (Divisive Decrease).

Update threshold using a function to calculate a new threshold value.

• Send data using the updated cundnew

The MDPC algorithm helps in providing a number of options that include securing the cryptographic exchange over the channel that is public with secure form of messaging and digital signature. Most of these types of systems are included in the number of the problems related to theory such as the factorization of the larger number with the discrete form of algorithm in the elliptic curve. Strong form of

cryptography is considered extremely essential for providing a secured electronic device for the consumers. These are the suspicious positions after the sleeping of the tentative position of each of the candidates. The “Block Rate of Error (BLER)” is also evaluated by simulation of the computer and the resultant represents the bit-flipping algorithm that provides lower BLER that is compared in order to exist within the algorithms of decoding.

3.6 MDPC Algorithm

MDPC (Multiplicative-Divisive Probabilistic Congestion Control) Algorithms play a key role in the benchmarking process. They are designed to analyze cloud storage systems from multiple dimensions, such as speed, reliability, and scalability. These algorithms provide a comprehensive understanding of how each cloud storage system performs under various conditions and workloads.

3.6.1 Mathematical Model:

1) Objective Function

Let f (x) be the objective function to be optimized, where x is the vector of decision variables. The objective is usually either to maximize or minimize f(x).

2) Constraints

The optimization problem may have constraints that define feasible regions for the decision

variables. These constraints can be represented as equality or inequality constraints, denoted as

$$g(x) < 0 \text{ or } h(x) = 0 \quad (3.1)$$

3) Decision Variables

Let x = (x1, x2,...,n) represent the decision variables. These variables determine the solution to the optimization problem.

4. State Space

The state space represents all possible states of the system at any given point in time. Each state

is associated with a set of decision variables and constraints.

Mathematical Formulation:

Let's denote the following:

• S: State space representing all possible states of the system.

• A(s): Set of feasible actions or decisions available in state s.

• T(s, a): State transition function representing the probability distribution of transitioning from state s to state s' after taking action a.

• R(s, a): Immediate reward or cost associated with taking action a in state s.

* V*(s): Optimal value function representing the maximum expected cumulative reward from state s to the terminal state.

Q*(s, a): Optimal action-value function representing the maximum expected cumulative reward from taking action a in state s and then following the optimal policy.

The dynamic programming recursion for MPDP can be formulated as follows:

$$V^*(s) = \max \{R(s, a) + \Sigma T(s,a, s') \cdot V' (s')\} \quad (3.2)$$

The optimal action-value function Q*(s, a) is given by:

$$Q^*(s, a) = R(s, a) + \Sigma T(s, a, s') \cdot V^* (s') \quad (3.3)$$

In the project, the Multiplicative-Divisive Probabilistic Congestion Control (MDPC) algorithm serves as a pivotal component in optimizing data transmission and managing network congestion effectively. Integrated within the project's mathematical model, MDPC dynamically adjusts the congestion window size based on real-time network conditions. Utilizing a probabilistic approach, the algorithm detects congestion by calculating the probability of packet congestion, thus enabling proactive measures to mitigate potential congestion events. By employing both

multiplicative and divisive factors, MDPC ensures adaptive control of the window size: decreasing it by a larger multiplicative factor in the presence of congestion and increasing it by a smaller additive factor during efficient network operation. This dynamic adaptation to network conditions, including round-trip time, packet loss rate, and available bandwidth, enables MDPC to maintain optimal performance and efficiency. Evaluated within the project's benchmarking environment, MDPC undergoes rigorous testing and comparison with other congestion control algorithms to assess its effectiveness and suitability across various network scenarios. Overall, MDPC significantly contributes to the project's objectives by enhancing multimedia data processing and communication through efficient congestion management and optimization strategies.

3.6.2 Properties MDPC Algorithm

The Multiplicative-Divisive Probabilistic Congestion Control (MDPC) algorithm is a type of congestion control algorithm used in computer networks to manage traffic congestion. Here are some key properties of the MDPC algorithm:
Multiplicative and Divisive Feedback: The MDPC algorithm uses both multiplicative and divisive feedback mechanisms to adjust the congestion window size . Multiplicative feedback increases or decreases the window size by multiplying it by a factor greater or less than one, while divisive feedback divides the window size by a factor greater than one.

1. Probabilistic Control: The MDPC algorithm is probabilistic in nature, meaning that it uses probability to determine the congestion window size . This approach is more effective in managing congestion in networks with high levels of variability and uncertainty.
2. Feedback Signal Estimation: The MDPC algorithm estimates the feedback signal based on the network conditions, such as the round-trip time, packet loss

rate, and available bandwidth . It then uses this estimate to adjust the congestion window size.

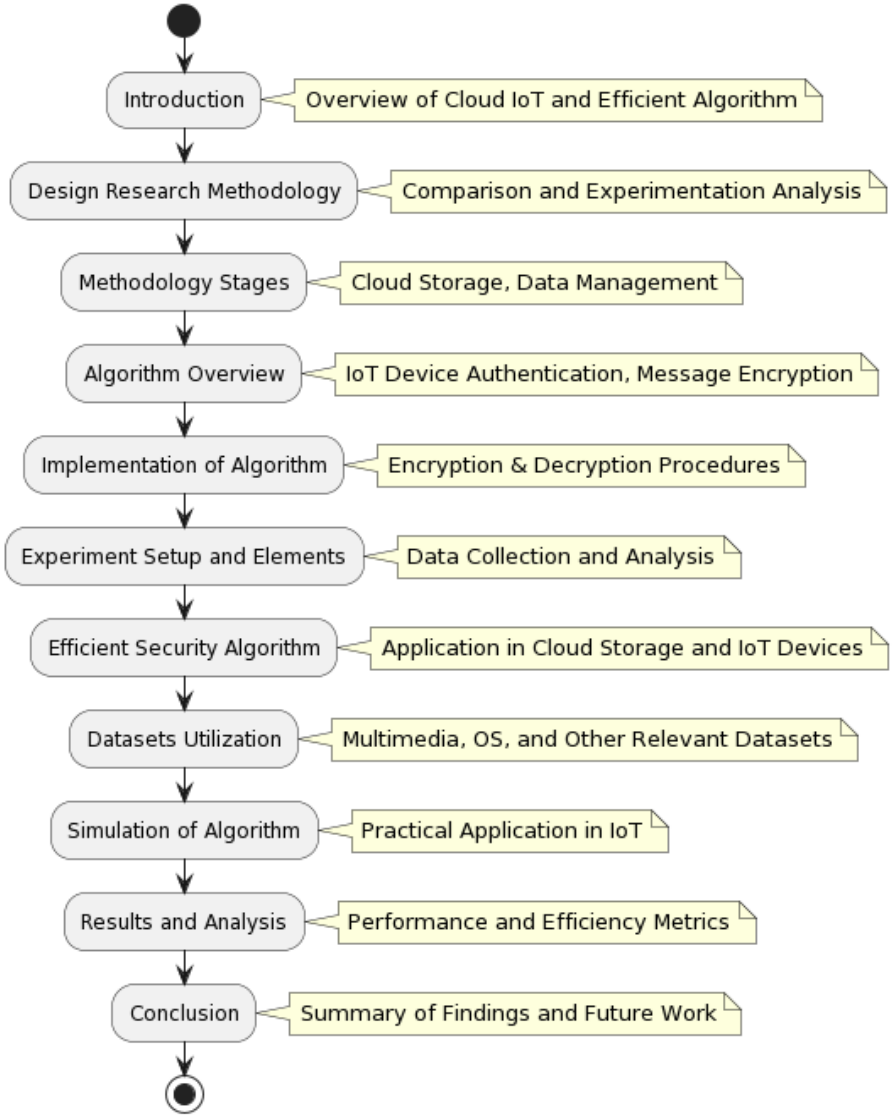


Figure 3.3: IoT Cloud Benchmark Architecture

3. Fairness: The MDPC algorithm aims to provide fairness to all the flows sharing the network resources [51]. It achieves this by adjusting the congestion window size based on the number of flows and the amount of traffic each flow generates.

4. Stability: The MDPC algorithm is designed to be stable and avoid oscillations in the congestion window size . This is achieved through the use of appropriate feedback mechanisms and control parameters.
5. Scalability: The MDPC algorithm is scalable and can be used in large-scale networks with a large number of flows. It can efficiently manage traffic congestion in such networks without compromising on performance.

Overall, the MDPC algorithm is an effective congestion control algorithm that provides fairness, stability, and scalability in computer networks.

3.6.3 Key Properties When Use MDPC in IoT Environment

When considering the use of the Multiplicative-Divisive Probabilistic Congestion Control (MDPC) algorithm in an efficient cloud storage mechanism for an IoT environment, some key properties are:

1. Adaptability: The MDPC algorithm is adaptable and can adjust to changing network conditions . In an IoT environment, where the number and type of connected devices can vary significantly, the MDPC algorithm can dynamically adjust the congestion window size to accommodate the changing traffic load.
2. Low Latency: In an IoT environment, low latency is critical for real-time applications . The MDPC algorithm is designed to achieve low latency by estimating the feedback signal based on the round-trip time, packet loss rate, and available bandwidth.
3. Energy Efficiency: IoT devices often have limited battery life, and energy efficiency is critical [45]. The MDPC algorithm can help reduce energy consumption by avoiding unnecessary retransmissions caused by congestion.
4. Robustness: The MDPC algorithm is robust and can withstand network disturbances such as link failures, node failures, and network partitions [48].

In an IoT environment, where nodes can be added or removed frequently, the MDPC algorithm can adapt to the changes and maintain network stability.

5. Scalability: The MDPC algorithm is scalable and can be used in large-scale IoT environments with a large number of devices. It can efficiently manage traffic congestion and provide fair access to network resources without compromising on performance.
6. Security: In an IoT environment, security is a critical concern. The MDPC algorithm can be used in conjunction with secure communication protocols to ensure the integrity and confidentiality of data transmitted over the network.

Overall, the MDPC algorithm is well-suited for use in an efficient cloud storage mechanism for an IoT environment, providing adaptability, low latency, energy efficiency, robustness, scalability, and security [47].

Here's a comparison table summarizing the overhead associated with the Multiplicative-Divisive Probabilistic Congestion Control (MDPC) algorithm compared to other commonly used congestion control algorithms:

Overhead Type	Overhead Description
Header Overhead	The MDPC algorithm requires additional header information to be added to each data packet to facilitate feedback and probabilistic control. This header overhead is generally small and can be managed with appropriate packet size optimization techniques.
Feedback Overhead	The MDPC algorithm uses feedback mechanisms to adjust the congestion window size based on network conditions [42]. This feedback involves the exchange of feedback packets between the sender and receiver, which can add some overhead to the network and increase the packet delay. However, the feedback overhead is generally small and can be optimized with appropriate feedback control parameters.

Probability Calculation Overhead	The MDPC algorithm uses probabilistic control to determine the congestion window size [46]. This involves the calculation of probability distributions, which can add some overhead to the network in terms of processing power and memory usage. However, this overhead is generally small and can be optimized with appropriate probability distribution estimation techniques.
Fairness Overhead	The MDPC algorithm aims to provide fairness to all the flows sharing the network resources. To achieve this, it requires some overhead in terms of monitoring and controlling the traffic flows to ensure that they are all treated fairly. This fairness overhead is generally small and can be optimized with appropriate fairness control parameters.

Overall, the data input stream overhead associated with the MDPC algorithm is generally small and can be managed with appropriate optimization techniques. The overheads related to feedback, probability calculation, and fairness control are generally manageable and can be optimized with appropriate control parameters and network optimization techniques.

Compared to some other congestion control algorithms, MDPC has some overhead due to the nature of its probabilistic and feedback-based approach. Here are some key overheads associated with the MDPC algorithm:

Computational Overhead: The MDPC algorithm requires frequent estimation of network conditions such as round-trip time, packet loss rate, and available bandwidth. This estimation involves some computational overhead in terms of processing power and memory usage.

Feedback Overhead: The MDPC algorithm uses feedback mechanisms to adjust the congestion window size, which involves the exchange of feedback packets between the sender and receiver [56]. This exchange can add some overhead to the network and increase the packet delay.

Probability Calculation Overhead: The MDPC algorithm uses probabilistic control to determine the congestion window size [46]. This involves the calculation of probability distributions, which can add some overhead to the network in terms of processing power and memory usage.

Fairness Overhead: The MDPC algorithm aims to provide fairness to all the flows sharing the network resources. To achieve this, it requires some overhead in terms of monitoring and controlling the traffic flows to ensure that they are all treated fairly.

Overall, the MDPC algorithm has some overhead associated with its feedback-based, probabilistic, and fairness-oriented approach. However, these overheads are generally reasonable and can be managed with appropriate control parameters and network optimization techniques. Compared to some other congestion control algorithms, such as TCP Reno, MDPC is generally considered to have lower overhead and better performance in networks with high levels of variability and uncertainty.

3.7 Enhanced Congestion Control Mechanism

To modify the Dynamic Prime Chunking (DPC) algorithm into the Multiplicative-Divisive Probabilistic Congestion Control (MDPC) algorithm, the following changes can be made:

Introduce a window size: In the MDPC algorithm, a window size is introduced to limit the number of packets in flight. The window size determines the amount of data that can be transmitted without acknowledgement from the receiver. The window size is adjusted dynamically based on the current network conditions.

Additive-increase, multiplicative-decrease: The window size is updated based on the success or failure of packet transmission. If a packet is successfully transmitted, the window size is increased by a small additive factor. If a packet is lost, the window

size is decreased by a larger multiplicative factor. This is similar to the Additive-Increase/Multiplicative-Decrease (AIMD) algorithm used in TCP congestion control.

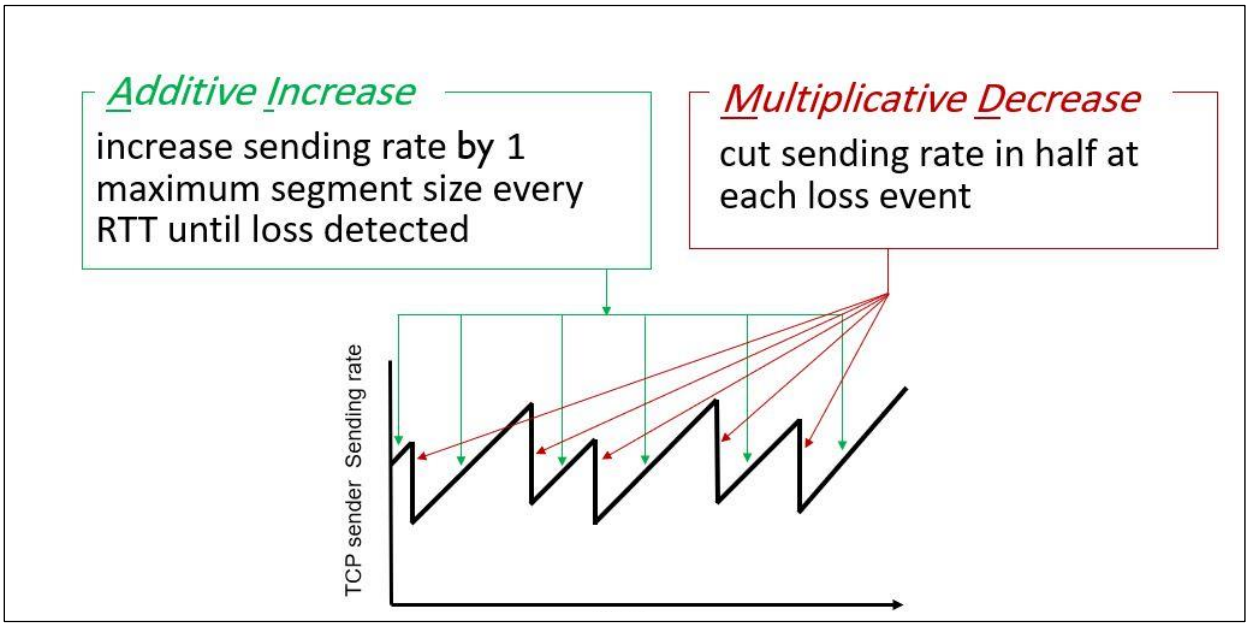


Figure 3.4: Additive-Increase/Multiplicative-Decrease

Introduce a probabilistic approach: In the MDPC algorithm, a probabilistic approach is used to adjust the window size. The probability of a packet being marked as congested is calculated based on the window size and the congestion level of the network [67]. The higher the window size, the higher the probability of a packet being marked as congested. This probabilistic approach ensures that the window size is adjusted in a stable and efficient manner.

Introduce a multiplicative-divisive component: In the MDPC algorithm, a multiplicative-divisive component is introduced to adjust the window size. If a packet is marked as congested by the network, the window size is decreased by a larger multiplicative factor. This helps to prevent congestion collapse in the network. Overall, the MDPC algorithm is a modification of the DPC algorithm that introduces a probabilistic approach and a multiplicative-divisive component to congestion

control [55]. This enables the algorithm to adjust the window size dynamically in response to changing network conditions, leading to better network performance and reduced packet losses.

3.8 Analysing the MDPC's behaviour for the CDC's

Multiplicative-Divisive Probabilistic Congestion Control (MDPC) is a congestion control algorithm that aims to regulate the congestion window size of a network transport protocol, such as TCP. In terms of the essential characteristics of congestion control, MDPC exhibits the following behaviors:

1. Responsiveness: MDPC is designed to be responsive to changes in network conditions, including changes in available bandwidth and congestion levels. It uses feedback from the network in the form of packet loss and delay to adjust the congestion window size accordingly.
2. Stability: MDPC is designed to be stable and avoid excessive oscillations in the congestion window size. It achieves this by using a probabilistic approach to adjusting the window size, where the probability of increasing or decreasing the window size is based on the current congestion level.
3. Fairness: MDPC is designed to be fair in its allocation of network resources among competing flows. It achieves this by using a multiplicative decrease and divisive increase approach to adjusting the congestion window size, which penalises flows that cause congestion and rewards flows that reduce congestion.
4. Efficiency: MDPC is designed to be efficient in its use of network resources. It achieves this by dynamically adjusting the congestion window size based on the current network conditions, which allows it to maximize network utilization without causing congestion.

Overall, MDPC exhibits the essential characteristics of congestion control by being responsive, stable, fair, and efficient in its regulation of network congestion.

However, the specific behavior of MDPC may depend on the implementation and configuration of the algorithm in a particular network environment.

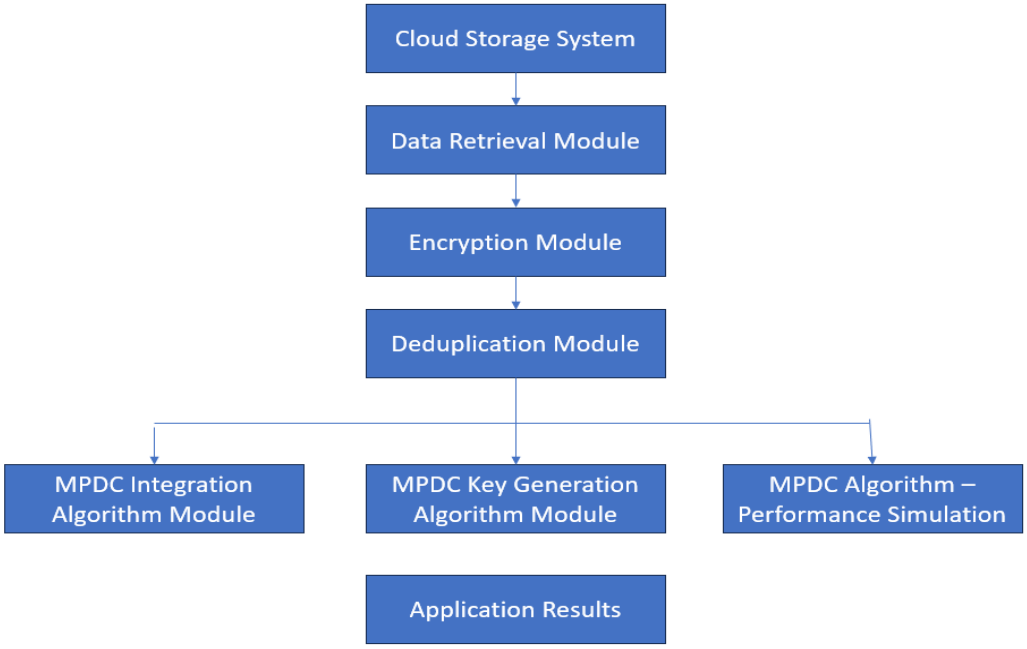


Figure 3.5: Implemented Model

The figure 3.5 illustrates the comprehensive integration of the MDPC (Multiplicative-Divisive Probabilistic Congestion Control) algorithm within cloud storage system. At its core, the Cloud Storage System serves as the repository for multimedia data, facilitating efficient data management. The Data Retrieval Module ensures seamless access to stored data, while the Encryption Module enhances data security through encryption techniques. Additionally, the Deduplication Module optimizes storage space by identifying and eliminating duplicate data entries. The MDPC Algorithm Integration is pivotal, harmonizing the algorithm's functionality with the system's architecture. Key Generation and Performance Simulation Modules play essential roles, generating encryption keys and evaluating system performance, respectively. Through a User/Application Interface, stakeholders

interact with the system, ensuring user-friendly access and management of multimedia data.

The MDPC codes are also designed using the binary cyclical through construction of the polynomial parity check that is obtained directly from the idempotent code using the cyclotomic cosets. The design of the MDPC codes include a lower complexity for the encoding and decoding scheme with the practical utilisation of the study. It also proposes a lower complexity of SISO diversity decoder [66]. The AD decoder includes the use of a small number of parity checks that are redundant and it attempts to minimise the operations that are not included in the regular algorithm. The decoding algorithms initially begins with decoding the length in with soft input vector that makes use of the regular algorithm sum product with (m * n) that is redundant according to the matrix of parity check that consists of the decoder that operates over the MDPC codes.

3.9 Theoretical Comparison

Rabin is a well-known duplication technique for use with CDC algorithms; nonetheless, it has a very poor chunking throughput and a substantial amount of chunk size volatility. The TTTD broke up data into smaller pieces, but it was unable to pinpoint where data duplication was occurring to account for the larger chunk sizes. In addition, since the processing time has increased, it adds to the overhead that is associated with indexing. In the end, the chunking AE method was superior to the Rabin in terms of the number of low-entropy strings it removed. We suggest using the dynamic prime chunking algorithm as a means to improve the throughput and take the performance to an even higher level.

- Low chunking throughput and time consumption are problems with Rabin.

- The TTTD algorithm adds a minimum and maximum threshold to lessen chunk volatility. The threshold is applied using a backup divisor. For bigger chunks, data deduplication cannot be properly recognised. Additionally, the longer processing times result in extra expense for indexing.

- Deduplication efficiency is also much greater in AE. Additionally, the computational cost is greatly reduced, and the tiny chunk variance is raised.

To reduce the computational cost in the Multiplicative-Divisive Probabilistic Congestion Control (MDPC) algorithm, the following techniques can be employed:

- Use Fixed Probability: Instead of calculating the probability of packet marking based on the window size and congestion level of the network, a fixed probability can be used [45]. This eliminates the need for computing the probability for each packet, which reduces the computational cost.
- Limit the number of packets marked: Instead of marking every congested packet, only a limited number of packets can be marked. This reduces the number of computations required to mark packets and also reduces the amount of feedback required from the network.
- Use Sampling: Instead of monitoring every packet in flight, a sampling approach can be used to monitor a subset of packets. This reduces the amount of monitoring required and reduces the computational cost.
- Use Approximation Techniques: Instead of using exact calculations, approximation techniques can be used to estimate the probability of packet marking. This reduces the computational cost while still providing a reasonable estimate of the probability.
- Reduce the number of parameters: The MDPC algorithm has several parameters that need to be adjusted, such as the window size, the additive and

multiplicative factors, and the probability of packet marking [66]. By reducing the number of parameters, the computational cost can be reduced while still maintaining good network performance.

Overall, these techniques can help to reduce the computational cost of the MDPC algorithm while maintaining good network performance. It is important to strike a balance between computational cost and network performance, and these techniques can help achieve that balance.

3.10 SHA -1 - Method Used to Eliminates Redundancies

The SHA -1 fingerprint technique is used by the Cryptographic Hash Function method to remove duplicate data and reduce redundancies at the full file or chunk level. The data deduplication procedure divides the incoming information into a variety of fragments.

SHA-1 is the original 160-bit hash function has a similarity to the earlier MD5 algorithm.

Destor uses the SHA-1 cryptographic hash algorithm in order to locate and get rid of the superfluous piece. The system is subjected to experimental testing in order to investigate the effects that the different chunk sizes and throughput have. The processing time as well as the chunk time that is executed in the system is the performance.

Steps Followed

- Padding of Bits
- Append Length
- Divide the input
- Initialize variable
- Process blocks

The technological overlap between the database redundancy, database backup and often lead to certain confusion however, each has a separate role to play in order to safeguard and streamline the data used. The backup is considered significant in order to create a duplicate copy of the data at a particular time point, which is ideally kept as multiple historic copies. The redundancy also establishes a straight copy for the entire system, which is considered ready to take over if the system originally fails. The backup also offers a certain level of redundancy that is neither considered the solution that is standalone [67]. The primary copy of the data that is selected reduces the data redundancy that is seen with the aim of retaining the data in the long term. All the data is continent within the backup that is ultimately ending to achieve that is really considered to be the solution for the backup but a complement to provide an optimised data storage procedure.

The incremental and differential backups also help in filling up the gaps in between the full backups and includes storing any type of changes to the data. It is also required as a fraction of the cycles in the CPU with the bandwidth and the storage space down the data loss risk is considered greater than the full backup restoring the times, which are considered slower [61]. Elimination of data is considered significant and it reduces the amount of the data that is required to be transferred or to be stored by eliminating and identifying both inter-object and intra-object. These are duplicated elements of data with the pointer all the reference to the unique copy of data.

The data redundancy increases the disparities. It includes preserving the data within the multiple areas that can cause the disparity of the information, which fails in updating across all the locations. This can also happen if the real storage location changes by the copies do not. It also creates certain opportunities for corruptive data. Data corruption occurs if something damages the information during transferring the

data or creating the procedure. This also means that storing several copies of similar data can provide more opportunities for data corruption. The costs are also considered for more data to be preserved. The data return density is considered costly to be maintained and to be interesting whether it is considered intentional or accidental.

The various ways that can be used in order to reduce the data redundancy, that is seen to include normalising the database. The normalisation of the database includes arranging the data into the database in an efficient manner ensuring the elimination of redundancy. This procedure also shares the database of the company that contains the details, which are read similarly throughout the entire record [61]. The normalising data also includes range in the tables and columns of the database in order to ensure that these are correctly and forced with their dependency. The various companies are considered to have certain special sets of the criteria about data normalisation, which are considered different approaches for data normalisation.

3.11 Tool for Cloud Storage in IoT

3.11.1 Software Requirements

When designing and developing software, it is best practice to first thoroughly understand the product's intended use. Here is a rundown of everything you'll need to meet BenchCloud's functional specifications:

- **Authentication and authorization for cloud services**.

Consumer identification is confirmed through confirmation, and their permissions and privileges are established through authorisation. Despite both of these phrases have a similar sound; they serve different but just as important functions in protecting systems and information [68]. It is essential to comprehend the differences. They establish a system's reliability when taken together.

- **Support various cloud storage services and product vendors.**

A *"CSP (cloud service provider)"* is a third-party firm that offers expandable hardware and software, such as cloud-based processing, storage, structure, and programming services, that organisations may use on request across an internet connection [69]. Data is sent over a communication link, usually through the web, and kept in distant data centres where it is up-to-date, controlled, and eventually made accessible to subscribers as part of a cloud storage structure.

- **Support various file operations, such as sharing, downloading, and uploading.**

Installing a *"File Transfer Protocol (FTP)"* client is the most popular approach for transmitting content to the website. Files may be sent coming from a single device (individual system) to a different one *(webserver)* via *"FTP (File Transfer Protocol)"* [61]. Anyone is able to transfer *(upload, download)* files from a single system to a different machine using *FTP software* that resembles an archives editor.

- **Support a variety of file generators to produce files with various patterns.**

MPS (Mathematical Programming System) manages an index of file formats, for every that connects an alphabetical facility using any number of naming designs. These kinds of documents are used for expressing linguistic-specific capabilities (such as "*syntax annotation"* and "*code estimation"*) in files embodying different dialects and techniques [62]. Every aspect of applicable naming sequence is included in the directory of file formats by default, yet it may add fresh file varieties for language-specific folders and modify the names of the file sequences that go with current file formats.

- **Assistance with multithreaded operations**

A program or computer's *"operating system (OS)"* that supports numerous users simultaneously despite necessitating numerous copies of the software to execute on a device is known as multithreading. Several inquiries travelling an identical person

can be handled via multithreading as well. Most operating systems offer combined *"kernel-level threads"* and threads created by users [69]. *Solaris* may be one of these instances. Different threads operate concurrently in the identical platform in this particular approach.

- **Compile benchmarking results into statistics.**

Through comparing a business's accomplishments to that of other people, and comparable businesses, anyone may determine whether, there is an achievement discrepancy, which can be filled by enhancing its own efficiency. Observing other businesses may show how long it is needed to boost an organization's productivity and establish a stronger position in the sector. The company may seek to increase productivity exponentially by discovering points at which it wishes to make improvements and measuring its present standing compared to rivals [61]. Through applying benchmarking in such a way, organisations have been able to surpass their rivals and raise the standard of excellence.

- **Automatically record and preserve benchmarking results.**

The *"Symanto Insights Platform"* analyses every feedback and summary's wording to determine if that writer is endorsing the business disparaging the business, or using a tone, which is neutral. A *"Net Promoter Score (NPS)"* is calculated by subtracting the opponents from the marketers. An excellent *NPS* is a sign of devoted and satisfied consumers [62]. The *"Symanto Insights Platform"* connects to popular online ratings and social networking sites like *Amazon*, *Trustpilot*, and *Google Reviews* to make it simple to quickly collect and evaluate countless language inputs.

- ***Record network packets while benchmarking is being done.***

The speed of transmitting data connecting two computers installing *"Performance Test"* needs to be tested using the *"PassMark Advanced Network Test"*, which happens to be a component of *"Performance Test"*. The storage device will be

among the devices, which will remain idle while it anticipates an internet link [70]. Any *TCP/IP* connectivity option is compatible with the internet sample evaluation including *Ethernet, wireless networking (WiFi), local area networks (LAN), wide area networks (WAN), cable modems, dial-up modems,* and *ADSL*. Exceptionally fast gigabyte *Ethernet* connectivity may be benchmarked according to the application's optimisation for minimal *CPU* time usage [70].

- **Able to test cloud storage systems' native clients and web APIs.**

An *API,* or *application-programming interface*, for cloud computing, interfaces a natively installed software to an online-based database so that users can transfer and receive content as well as manipulate the data held there. Similar to disk-based storage, a cloud-based memory framework is essentially another prospective medium for the programme [63]. A cloud *API* is unique based on the data storage provider it is intended to support. An internet-based archiving provider could. For instance, provide an *API* that can generate, gather, and destroy items on that system in addition to carrying out similar item-related operations [70]. A *file preservation API* supports actions like sending and receiving items and distributing documents with many individuals at the component and category layers.

3.12.2 The specifications for a benchmarking tool for cloud storage systems

The global rise of cloud computing along with the development of many cloud storage systems have been built with the objective of providing decentralised and reliable file storage. Therefore, it is important to be well aware of their specific features and performances along with the ways through which it could be optimally used. The market witnesses an exponential rise in cloud storage systems nowadays, and therefore certain guidelines could be instrumental in choosing the appropriate system that can potentially satisfy the requirements. [60] The storage systems are

found to have more or less similar functions and therefore springs up the requirement of benchmarking it.

These days, there are a great number of cloud storage solutions available, and there are always new companies entering the market. As a result, we need some direction to pick the proper solution that will provide the highest level of satisfaction for needs. We need to evaluate these cloud storage systems since the performance of the systems is a major concern that we need to take into account, and because many cloud storage systems share similar duties, this is why we need to compare them. The following are some examples of probable situations when it may be beneficial to have a benchmark.

- **Select the quickest cloud storage solution for regular usage**

Suppose a user is going to give any cloud storage system a try so that he may store his data in the cloud and synchronise the information across the computers in his home and office. The customer's primary concern is that the service should be able to upload and download files as quickly as feasible. A benchmarking has to be done in order to establish which cloud system has the greatest performance when it comes to the uploading and downloading of files. This is necessary since different cloud systems have different network bandwidth and different locations for their data centres.

Certain aspects should be borne in mind prior to choosing the ideal cloud storage system such as the storage location as the physical location of a cloud server can potentially affect the recovery and the performance. Simultaneously, there could be issues regarding compliance or regulatory requirements on data storage locations therefore, the decisions regarding locations should be based on the importance of the data, authorisation and cost. [61] In addition to this, issues regarding security are of top concern when it comes to cloud storage and therefore it should be emphasised

that while the protection of the data is the responsibility of the cloud service provider the user also is responsible to maintain security guidelines while transferring data on cloud server.

Additionally, performance evaluation is yet another important factor in the process of finding the appropriate cloud service. Certain performance related aspects such as response time, processing time, bandwidth, latency, CPU, infrastructure, RAM and so on are critical in the process of choosing cloud storage. In addition to this, the viability of integrating along with other applications should also be prioritised. Therefore, prior to selecting the cloud storage "*Application Program Interface (APIs)*" should be assessed. [63] In addition to this, the compatibility of the cloud server with the existing applications as well as storage devices should be checked in order to ensure the ease of accessibility.

- **Find out how to use a cloud storage system as a backend storage system for web and mobile apps**

Many of the web applications that we use today store the data of their users in the user's own personal cloud system, as opposed to storing the data in a dedicated server that is maintained by the application's developer. This is made possible by the development of SaaS and mobile computing**.** These kinds of web apps that are hosted in the cloud come with a few distinct benefits. To begin, the creator of the programme does not have to keep any dedicated storage servers running, which means that the overall cost may be significantly lowered. Second, the fact that the data is saved in the user's own cloud space, which is maintained by a reliable cloud storage service provider, allows the user to have peace of mind regarding their data. This, in turn, will make the application more appealing to users who place a high priority on the security of their data. Thirdly, the data that is saved in the cloud is able to take use of certain additional features offered by the cloud, such as the ability

to share and synchronise files. Site441, a web application that is built on Dropbox and has the ability to convert Dropbox files into websites that are available to the public, is one example of this kind of programme. As the developer of an application that makes use of a cloud storage service, he may need to be aware of the most effective technique to make use of the service. For instance, while uploading data to the cloud, is it possible to make advantage of multithreading? If the answer to that question is affirmative, then how many different threads should be employed to provide the highest possible performance? Should the data be divided up into many files of a lower size before it is uploaded if we want the uploading of enormous amounts of data to go as smoothly as possible? In order to provide answers to such issues, a benchmark is often seen as being beneficial for evaluating the levels of performance achieved by using various cloud storage service utilisation methodologies.

- **Analyse the effectiveness of Cloud Storage Systems for a certain use case**

The vast majority of the cloud storage solutions that are available to us today were developed for typical, day-to-day activities such as the casual archiving of images, audio tracks, and documents. However, being a cloud storage system with a broad range of applications, it is possible to utilise it for purposes other than the typical, everyday ones. It is feasible, for instance, to utilise a cloud storage service as the backend storage system of an Internet of Things thesis with multiple sensors that constantly take data from the environment and transfer it simultaneously to the backend. This particular use case differs from others in that it involves simultaneously uploading a huge number of little files that have been generated in enormous quantities. A benchmark is always required in order to investigate whether or not a cloud-based storage system can be used in a certain situation and to evaluate its performance.

In a nutshell, doing benchmarks on cloud storage systems is beneficial in a variety of different ways. In point of fact, we are able to do ad hoc benchmarking manually; but, doing so will need a significant amount of time, and the procedure itself will be difficult to replicate. In addition, if one has to carry out sophisticated benchmarks, such as multithreaded uploading with random file creation, it is often impossible to avoid the need of developing scripts and programming. Because of these drawbacks of manual benchmarking, an automated benchmarking tool is the key to improving the efficacy of benchmarking jobs. This is the motivation for the creation of BenchCloud, which was developed specifically for this purpose.

3.11.3 System Architecture Goals

a. Flexibility

Flexibility in BenchCloud refers to its adaptability to a wide range of benchmarking needs. This adaptability is crucial because benchmarking tasks vary greatly in their objectives and methodologies. To achieve this, BenchCloud is designed with high configurability and extensibility. Configurability allows users to make detailed adjustments to benchmarking parameters, such as selecting the cloud storage system to test, defining the operations (like uploading or downloading files), setting the number of operations, and determining the number of threads for execution. Extensibility, on the other hand, ensures that BenchCloud can evolve to include new functionalities or support new cloud services without the need for extensive modifications to its existing components. This aspect of flexibility is especially important in cloud computing, where the ability to customize applications and access services from anywhere with an internet connection is highly valued. The cloud's popularity has surged due to its ease in data access and storage, coupled with the capability to scale resources and swiftly adapt to consumer demands.

b. Usability

Usability in BenchCloud is about providing an intuitive and accessible user experience. Recognizing that not all users have a background in Python, despite BenchCloud being developed in this language, the system uses configuration files for customization. These files allow users to easily modify almost every aspect of a benchmark's settings without needing extensive technical knowledge. This approach to usability is particularly significant in cloud computing, where a diverse range of consumers and service providers often find the plethora of options overwhelming. With BenchCloud, usability is enhanced by simplifying the configuration process, making it more approachable for a wider audience. This feature is vital in helping users navigate the complexities of cloud services and make informed decisions, especially when selecting resources like virtual machines (VMs) for deployment.

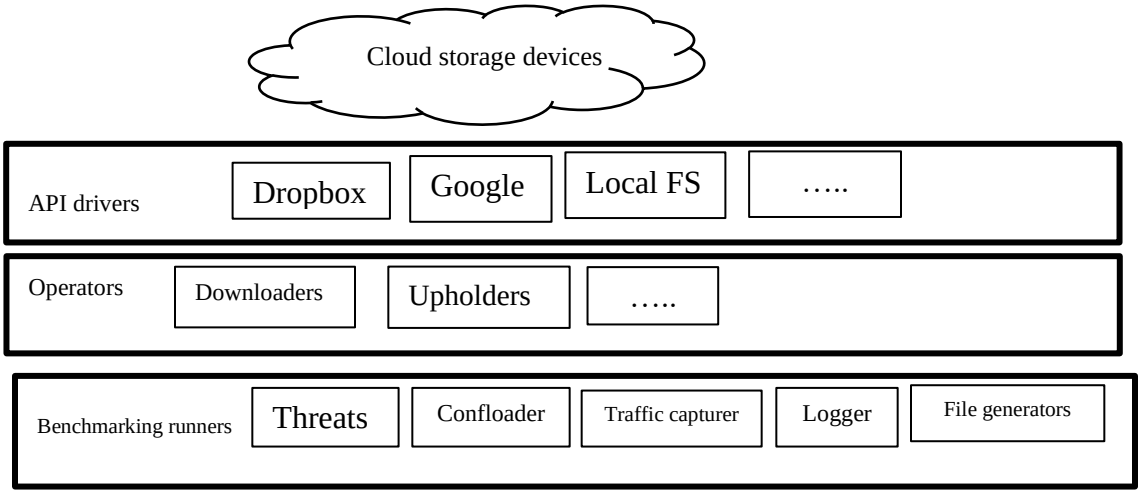


Figure 3.6: System Architecture of Bench Cloud

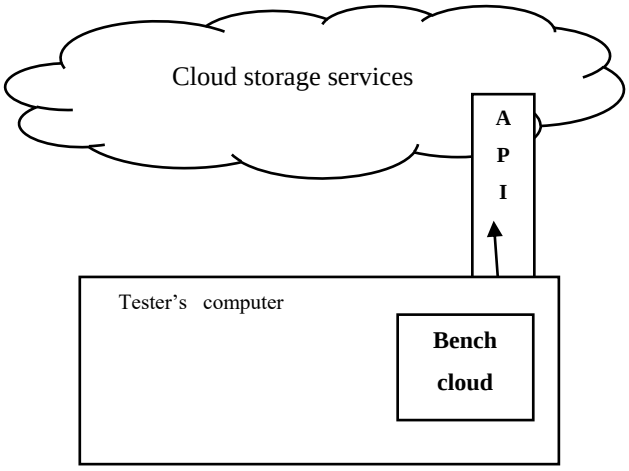
Advantages that are unique and obvious for each cloud client, as well as cloud service providers, are what is driving the increase in the use of cloud computing. Consumers now find it more difficult to select a cloud provider due to the growth in both the number of operators and the type of services they deliver [67]. A difficulty for internet service providers is also presented by the variety of alternatives for constructing a cloud infrastructure, including cloud administration tools and various

networking and storage techniques. Considering choosing ***"virtual machines (VMs)"*** to use for the deployment of an implementation, asset benchmarking might be useful. Performance benchmarking is crucial to comprehend the dependability and volatility of the cloud-based services delivered [71].

3.11.4 System Architecture

Bench Cloud utilizes an architecture that is layered. As can be seen in Figure 3.5, it is composed of three primary layers.

A. The API Driver Layer



(a) Test via web APIs

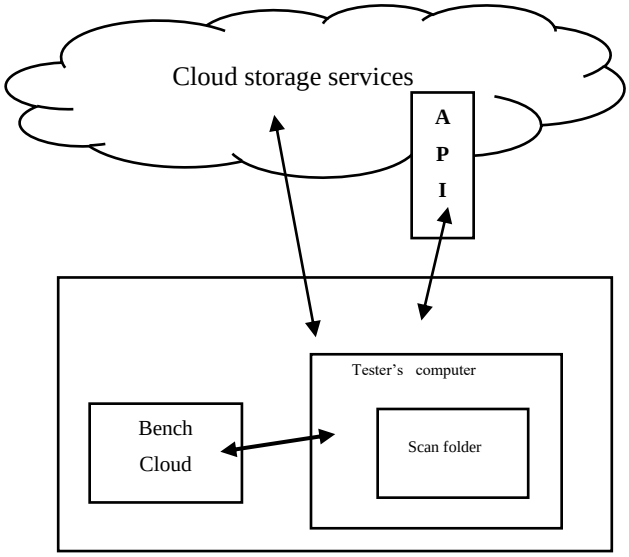


Figure 3.7: (a), (b) Two styles of test architecture

The Application Programming Interface (API) Driver layer is responsible for providing communication end points to cloud storage providers. It provides cloud service wrappers that the Operators layer may use to activate cloud services. A cloud service wrapper establishes communication with cloud storage services by using RESTful APIs**,** and it offers features such as service authentication and authorization, the acquisition of file metadata, file uploading and downloading, file

sharing, and other similar features. The "uploading" and "downloading" of files to and from the tester's local file system is handled by a specialised driver known as the "Local FS driver." The Local FS driver, in contrast to other drivers, does not utilize online APIs that are accessed from cloud storage services. Instead, it simply performs standard file copy operations inside the confines of the local file system. In the event that you do not want to test against online APIs but rather to the native clients of some cloud storage services, you will need to make use of a local file system driver. The synchronisation client for these kinds of systems runs on the users' computers and synchronises the users' local data (often inside a designated synced folder) with the cloud.

By "uploading" files to the synchronized folders and letting the synchronization client handle the processing and actual uploading operation, can study the client in some ways and see what kinds of optimization it engages in. Such a client may have interesting features that cannot be discovered by testing against web APIs directly. The high-level testing architecture may be split into two different forms, as illustrated in Figure 3.6, depending on whether a web API or client is to be evaluated.

B. **The Operators Layer**

The Operators Layer serves as an intermediary between the user-facing applications and the API Drivers layer, translating high-level actions into specific API calls. This layer encapsulates the complexity of interacting with various cloud storage APIs by providing a unified interface for common operations such as uploading and downloading files. By doing so, it abstracts away the idiosyncrasies of individual cloud storage providers, allowing developers to write code that is agnostic to the underlying cloud service. Developers can leverage the Operators Layer to build applications without the need for deep knowledge of the specific API details for each cloud storage service. This not only speeds up the development process but also

enhances the portability of applications across different cloud environments. The general-purpose tools within this layer are designed to be flexible and extensible, enabling them to support a wide range of cloud storage options and new features as they are released by cloud providers. The Operators Layer promotes a modular architecture where new functionalities can be easily integrated. It is engineered to handle error checking, retries, and other resilience strategies, thus ensuring reliable file operations. This layer plays a crucial role in the scalability and maintenance of cloud storage applications, as it simplifies the process of updating or swapping out API Drivers without significant changes to the application logic.

C. **The Benchmarking Runner Layer**

The task of parsing and loading configuration files and running the benchmark depending on the configuration falls within the purview of the Benchmarking Runner Layer. The logger is in charge of meticulously recording all of the precise actions and time spent while running benchmarks. When doing benchmarks for uploading files, benchmarking runners often utilizes a tool called a file generator to generate files depending on specified setup. There are four basic types of file generators that provide various file content patterns:

- **Random File Generator**. It generates files with unpredictable content that are difficult to compress well and very unlikely to share the same content as other created files.
- **Identical File Generator**. A succession of identical files is created using an identical file generator. It is crucial for evaluating a cloud storage system's file deduplication function.
- **Sparse File Generator**. It produces files with little material. Content that has repeated strings is said to be sparse. A high compression rate may be used to effectively compress files created by a sparse file generator. A crucial

component of evaluating a synchronisation client's file compression capability is the sparse file generator.

- **Delta File Generator**. A delta file generator creates a number of identically contented files that are all the same size. The contents of the remaining portions of the files are random and not similar. A synchronisation client's delta encoding functionality must be tested using the Delta File Generator.

In order to capture and dump network packets during a benchmark, a trac capturer is included in the benchmarking layer. The resultant dump file's data format, PCAP6, is one that is widely used for recording network packets and can be read and analysed by a variety of packet capture and analysis programmes, including Wireshark7. The PCAP format keeps detailed records of the packets created, allowing for use in post-analysis to examine the characteristics of the network traffic.

3.11.5 Cooperation with other tools

As mentioned in the previous section, BenchCloud offers a simple method for evaluating cloud storage systems and can track the amount of time spent on each stage of the benchmarking procedure. However, users could need more details in addition to saving time, and they might use certain other tools to examine the recorded packets to get further knowledge. BenchCloud does not provide a comprehensive packet analysis tool since there are already established tools available to do this. The tools for packet analysis that may be used in conjunction with BenchCloud are introduced in this section.

Wireshark

Wireshark is considered as a open source network protocol analysis software program that is prevalently considered as an industry standard program where it is found to capture network traffic ranging from ethernet to Bluetooth and stores it for offline analysis. It is found to be helpful in troubleshooting problems across a

network, in debugging protocol implementations, verification of applications and so on. [64] It is significant in tracing connections specifically in connection to cybersecurity issues along with keeping track of suspect networks and identifying bursts of network traffic. Therefore, it is widely utilised for troubleshooting networks, software communications, and analysis and so on.

An open-source network packet analyzer is Wireshark**.** It is an effective tool for studying and debugging network traffic. Being cross-platform, it may be set up on other operating systems, including GNU/Linux, Mac OS X, Solaris, Microsoft Windows, etc. Both a command line tool and a visual user interface are available for Wireshark. Some of Wireshark's key characteristics include:

- Capture live packets from a network interface
- Import/Export packets
- Show packet data in a detailed and structured way
- Show the protocol-specific information of packets
- Filter packets according to various rules 6. Make various kinds of statistics

tcpdump

A packet analysis tool comparable to Wireshark is tcpdump**.** But tcpdump differs from Wireshark in that it only offers a command-line tool and no graphical user interface. Most Unix-like systems can run tcpdump, which is often supplied with these systems. Additionally, WinPcap8 is the name of the tcpdump port for Windows.

Tcpdump is a packet analyser that is generally launched from the command line packet analysis. It is found to be incredibly useful as a packet analysis tool, as it is fast in examining individual packets or communication that therefore is one of the most widely used and prevalent analysis tools. It is beneficial in the sense that it provides consistent output, therefore, enabling manipulation of packet data with

scripts rather easily. [65] tcpdump provides beneficial insights regarding the behaviour of the networks, however, tcpdump is found to lack fancier analysis features as a result of its simplicity in comparison to other graphical tools like Wireshark.

CHAPTER FOUR

SYSTEM IMPLEMENTATION AND RESULTS

CHAPTER - 4

SYSTEM IMPLEMENTATION AND RESULTS

4.1 Introduction

Imagine a symphony of tiny data collectors scattered across a field, each capturing snapshots of sound, sight, and maybe even movement. These are wireless multimedia sensor nodes, whisperers of a thousand stories waiting to be told. Yet, their voices need a conductor, a path from their sensors to the world. That's where revolutionary routing protocol, guided by the magic of the MDPC algorithm, steps in. But before the music can play, need to carefully set the stage.

Think of it like building a miniature city for these data whisperers. First, need houses – tiny, brainy homes called microcontrollers or processors. These whiz kids of tech will crunch numbers, run the operating system, and orchestrate communication. But each house needs a different kind of resident – powerful processing units for the central hub (the base station) and more frugal versions for the sensor nodes, all siphoning energy like careful mice from a tiny battery.

Next, each house needs a rulebook, an operating system to keep things humming. Efficiency is key here, like a miniature traffic cop ensuring data flows smoothly, especially for those fleeting moments captured in a blink or a whisper. To build this city, need tools, screwdrivers of code and debuggers to fine-tune the system. And, of course, a language – not just any language, but one that tiny processors understand and that routing and MDPC algorithms can sing their magic in.

But the city needs more than just houses and rules. Walls and gates come in the form of security measures, protecting the secrets these sensors hum. need power stations, too, carefully managing energy so data whisperers don't fall silent too soon. And finally, the city must be adaptable, growing and changing with new sensors and the whispers they bring.

By meticulously crafting this foundation, pave the way for symphony of sensors. With each component in place, novel routing protocol and the brilliant MDPC algorithm can take the stage, transforming whispers into a captivating chorus, a story told through a thousand tiny eyes and ears.

A network topology that can support the wireless multimedia sensor network and the routing protocol. This can be a star, mesh, or tree topology. The topology should be designed to optimize energy consumption and reduce communication overhead. The protocol should be designed to handle the specific requirements of multimedia data, such as high bandwidth and low latency. The protocol should also be designed to handle the dynamic nature of wireless sensor networks, such as node failures and mobility. The protocol should be implemented on each sensor node and base station using the appropriate software tools and programming language. The nodes should be configured with appropriate parameters, such as transmission power, routing table, and MDPC parameters. The network should be tested to verify the effectiveness of the protocol and MDPC algorithm.

Performance Evaluation: The performance of the protocol and MDPC algorithm should be evaluated in terms of throughput, delay, energy consumption, and packet delivery ratio. The evaluation should be performed in a real-world environment to ensure the effectiveness of the protocol in practical scenarios. Overall, the implementation of a novel routing protocol for wireless multimedia sensor networks using the MDPC algorithm will require careful consideration of hardware and software requirements, network topology, protocol design, implementation steps, and performance evaluation.

4.2 deduplication Technique for cloud storage

Cloud storage mechanism using deduplication technique can be applied in the above scenario to reduce storage overheads and improve storage efficiency. The following are the steps involved in implementing cloud storage using deduplication technique:

1-Data Segmentation: The multimedia data collected from the wireless multimedia sensors can be segmented into smaller chunks. Each chunk can be given a unique identifier or hash value.

2-Data Deduplication: The hash values of the data chunks are checked for duplicates. If there are any duplicates, only one copy is stored in the cloud storage. This reduces the storage overhead and improves storage efficiency. It is an efficient approach in the process of handling and storage of a vast amount of data and is imminent in identifying duplicate content with the implementation of cryptographically secure hash signature. Simultaneously, it also helps in the reduction of the transmission of redundant data particularly in low-bandwidth network environments.

3-Indexing: An index is maintained for all the data chunks and their hash values. This index helps in quickly identifying whether a particular data chunk already exists in the cloud storage or not. Indexing helps in smooth retrieval of entries from database files with the implementation of attributes that have already been indexed.

4-Encryption: To ensure data security and privacy, the multimedia data can be encrypted before storing in the cloud storage. Only authorised users with proper authentication and access rights can decrypt the data. Encryption is generally employed in order to encrypt data in the process of outsourcing it.

5-Data Retrieval: When a user requests for a particular multimedia data, the cloud storage system retrieves the corresponding data chunks and reconstructs the original multimedia data. Overall, cloud storage mechanism using deduplication technique provides efficient storage and retrieval of multimedia data in a secure and reliable manner. It reduces the storage overhead and improves storage efficiency by storing only unique data chunks.

4.2.1 Data Segmentation

Data segmentation is the process of grouping the similar categories of data based on the specific parameters in order efficiently use them. It helps the cloud service providers easily stock the data along with having proper knowledge of locations of all the files. It also helps the users easily access the correct data within a minimum amount of time [74]. During data segmentation, the memory is divided into small parts of various sizes in order to manage the memory of the cloud system effectively. Each small part of the memory is referred to as a segment of the process.

K-means clustering segmentation is used for the purpose of image segmentation in the cloud storage system. There is another algorithm called FCM, which helps to categorise the pixels of the image into different classes in order of their varying degree of membership. K-means is a very simplified machine-learning algorithm. It helps to classify any image through the implementation of specific numbers of clusters [75]. It initialises its working process by grouping the image space into K pixels, which represent the centroids of the K group. Each group is assigned with an object based on the distance of separation between them and the centroid.

Here's an example of data segmentation for the above scenario with tables and graphs:

Assume have a multimedia data file of size 50 MB. To segment this data into smaller chunks, can use a fixed-size segment of 1 MB each. This means will have 50 segments of 1 MB each.

Table 4.1 Data Segmentation

Segment Number	Start Offset	End Offset	Size
1	0	1048575	1 MB
2	1048576	2097151	1 MB
3	2097152	3145727	1 MB
...	...	...	...
50	47185920	48234495	1 MB

As shown in the table, the 50 MB multimedia data file is divided into 50 segments, each of 1 MB size. These segments are identified by their segment number and start and end offsets. The segmentation graph shows the 50 MB data file divided into four segments of 1 MB each. This segmentation process makes it easier to handle large multimedia data files and helps in efficient storage and retrieval of data in a cloud storage environment.

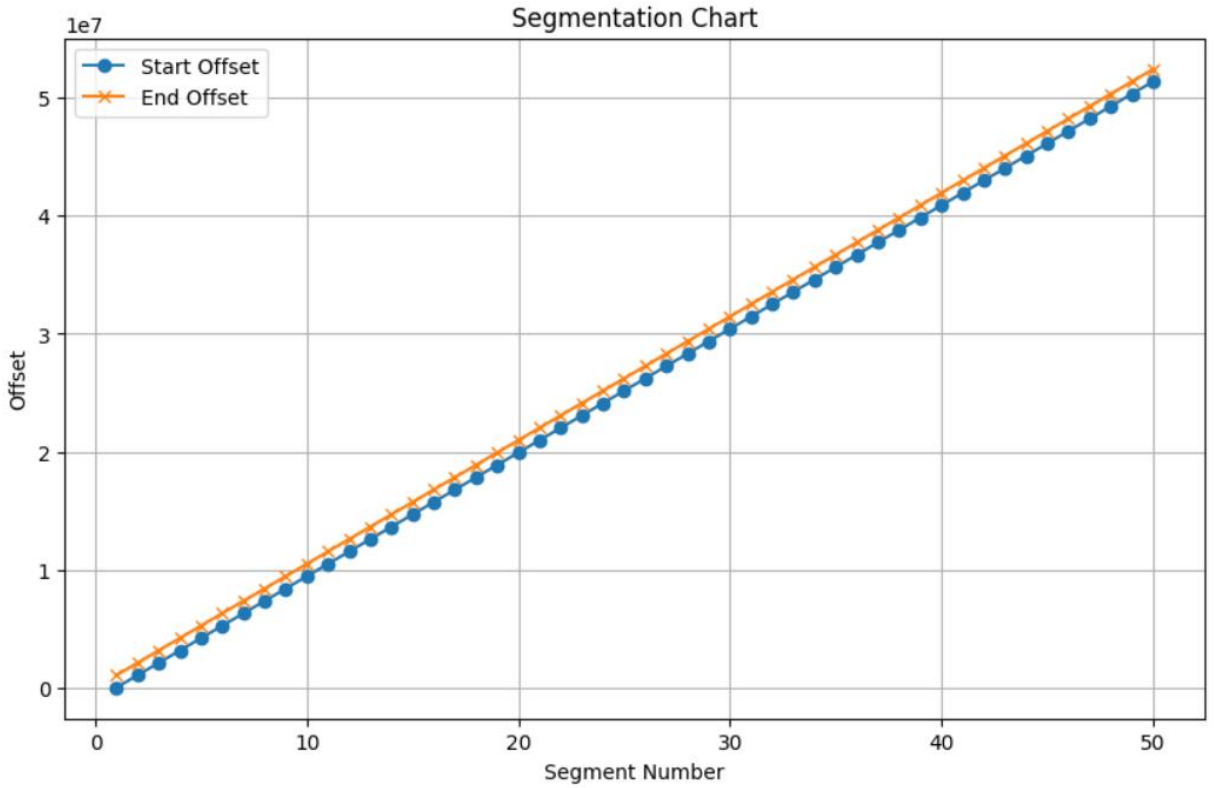


Figure 4.1 Segment Number

As shown in figure 4.1, the segmentation chart displays the start and end offsets for each segment number. Here's what you can observe from the chart:

X-axis: Segment Number - Each segment is represented along the x-axis, ranging from 1 to 50.

Y-axis: Offset - The offset values (in this case, start and end offsets) are represented on the y-axis.

Start Offset: Marked with circles ('o') - Each circle represents the start offset of a segment.

End Offset: Marked with crosses ('x') - Each cross represents the end offset of a segment.

Trend: As segment number increases, both start and end offsets increase linearly. This suggests a consistent segmentation pattern where each segment has a fixed size.

This chart provides a clear visual representation of the segmentation pattern, making it easy to understand how the data is divided into segments.

4.2.2 Deduplication

Here's an example of deduplication for the above scenario:

Assume have collected multimedia data from 10 wireless multimedia sensors. Each sensor has captured a video of size 50 MB. To store this data in a cloud storage system, can use deduplication techniques to reduce storage overhead and improve storage efficiency.

Table4.2: Deduplication

Sensor ID	Segment Number	Hash Value
Sensor 1	1	2f8085b95f5b26cf
Sensor 1	2	3b9ebc534f2ea695
Sensor 1	3	7e70d10845f8c2b2
...	...	...
Sensor 10	50	1a56830c8f153a0c

As shown in table 4.2, each segment of multimedia data captured by the sensors is given a unique hash value. The hash value of each segment is checked for duplicates in the cloud storage system. If there are any duplicates, only one copy is stored in the cloud storage system, and the duplicate references are updated to point to the original copy. In this way, can reduce the storage overhead and improve storage efficiency. The deduplication graph shows how the multimedia data from each sensor is divided into 50 segments of 1 MB each, and each segment is given a unique hash value. The deduplication table shows the hash values of each segment, along with the sensor ID and segment number.

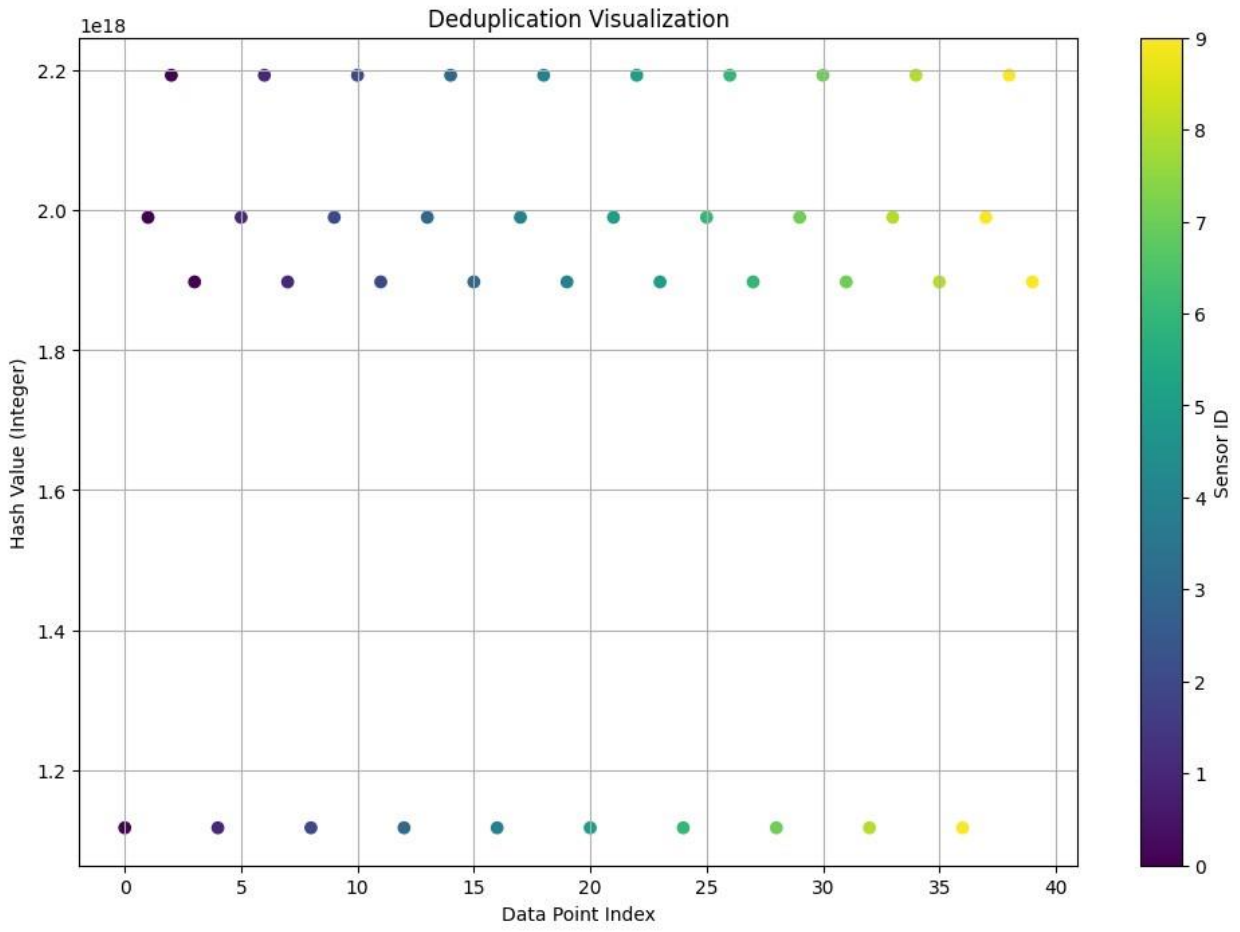


Figure 4.2 Data Point Index

As shown in figure 4.2, The deduplication visualization displays the hash values of data points across different sensors. Here's what you can observe from the chart:

X-axis: Data Point Index - Each data point is represented along the x-axis, with indices ranging from 0 to the total number of data points.

Y-axis: Hash Value (Integer) - The integer representation of hash values is represented on the y-axis. The hash values are converted to integers for visualization purposes.

Color: Sensor ID - Each data point is colored based on its corresponding sensor ID. The color bar on the right indicates which color corresponds to each sensor.

Distribution: The scatter plot shows the distribution of hash values across different data points and sensors. Data points with similar hash values are likely to be duplicates, as they would map to the same y-coordinate on the plot.

4.2.3 Indexing

Assuming have stored multimedia data from 10 wireless multimedia sensors in a cloud storage system using data segmentation and deduplication techniques, can use indexing to efficiently retrieve the data from the cloud storage system.

Table 4.3: Indexing

Sensor ID	Segment Number	Hash Value	Cloud Storage Path
Sensor 1	1	2f8085b95f5b26cf	/cloud_storage/sensor1/segment1
Sensor 1	2	3b9ebc534f2ea695	/cloud_storage/sensor1/segment2
Sensor 1	3	7e70d10845f8c2b2	/cloud_storage/sensor1/segment3
...	...	...	...
Sensor 10	50	1a56830c8f153a0c	/cloud_storage/sensor10/segment50

As shown in table 4.3, have indexed each segment of multimedia data with its sensor ID, segment number, unique hash value, and cloud storage path. The cloud storage path represents the location of the segment in the cloud storage system. By using this index, can quickly retrieve any segment of multimedia data from the cloud storage system by specifying its sensor ID, segment number, or hash value. The indexing graph shows how the multimedia data from each sensor is stored in the cloud storage system, and how the indexing is done for each segment of data. The indexing table shows the indexing details for each segment, including its sensor ID, segment number, hash value, and cloud storage path.

4.2.4 Encryption

Table 4.4: Encryption

Sensor ID	Segment Number	Hash Value	Cloud Storage Path	Encryption Key
Sensor 1	1	2f8085b95f5b26cf	/cloud_storage/sensor1/segment1	0x8f7d45a3
Sensor 1	2	3b9ebc534f2ea695	/cloud_storage/sensor1/segment2	0xa2c3f45e
Sensor 1	3	7e70d10845f8c2b2	/cloud_storage/sensor1/segment3	0x1b9e0c8f
...	...	...	...	...
Sensor 10	50	1a56830c8f153a0c	/cloud_storage/sensor10/segment50	0xd3a5b0c8

In the above scenario, have stored multimedia data from 10 wireless multimedia sensors in a cloud storage system using data segmentation, deduplication, and encryption techniques. The encryption table shows the encryption details for each segment of multimedia data. Each row of the table represents a segment of multimedia data, and the columns represent the following:

- Sensor ID: The unique identifier of the sensor that collected the data.
- Segment Number: The number of the segment within the sensor's data stream.
- Hash Value: The hash value of the segment, used for deduplication.
- Cloud Storage Path: The path of the segment in the cloud storage system.
- Encryption Key: The key used to encrypt the segment.

The encryption key is generated using a symmetric encryption algorithm, such as AES, and is used to encrypt the data before it is stored in the cloud storage system. When retrieving the data, the same encryption key is used to decrypt the data. This ensures that the data remains secure even if it is intercepted during transmission or if the cloud storage system is compromised. By using an encryption table, can

efficiently retrieve the encryption key for a particular segment of data, which is needed to decrypt the data. This enables secure and efficient retrieval of the multimedia data from the cloud storage system.

4.2.5 Data Retrieval

Table 4.5: Data Retrieval

Sensor ID	Segment Number	Hash Value	Cloud Storage Path	Encryption Key	Data
Sensor 1	1	2f8085b95f5b26cf	/cloud_storage/sensor1/segment1	0x8f7d45a3	...
Sensor 1	2	3b9ebc534f2ea695	/cloud_storage/sensor1/segment2	0xa2c3f45e	...
Sensor 1	3	7e70d10845f8c2b2	/cloud_storage/sensor1/segment3	0x1b9e0c8f	...
...	...	...	...	...	...
Sensor 10	50	1a56830c8f153a0c	/cloud_storage/sensor10/segment50	0xd3a5b0c8	...

In the above scenario, have stored multimedia data from 10 wireless multimedia sensors in a cloud storage system using data segmentation, deduplication, and encryption techniques. The data retrieval table shows the details for each segment of multimedia data that can be retrieved from the cloud storage system. Each row of the table represents a segment of multimedia data, and the columns represent the following:

- Sensor ID: The unique identifier of the sensor that collected the data.
- Segment Number: The number of the segment within the sensor's data stream.
- Hash Value: The hash value of the segment, used for deduplication.
- Cloud Storage Path: The path of the segment in the cloud storage system.

- Encryption Key: The key used to encrypt the segment.
- Data: The multimedia data stored in the segment.

To retrieve a segment of multimedia data, would first look up the segment in the data retrieval table using the Sensor ID, Segment Number, and Hash Value. Once have located the segment, would use the Cloud Storage Path to retrieve the encrypted segment from the cloud storage system. Finally, would use the Encryption Key to decrypt the segment and retrieve the multimedia data stored in the segment. By using a data retrieval table, can efficiently retrieve the multimedia data stored in the cloud storage system. This enables us to efficiently process and analyze the multimedia data collected by the wireless multimedia sensors.

4.3 Comparative Study table of Rabin, TTTD, MAP, AE and MDPC

Here's a comparative study table of Rabin, TTTD, MAXP, and AE in addition to MDPC Algorithm for the above scenario:

Table 4.6 Comparative Rabin, TTTD, MAP, AE and MDPC

Algorithm	Packet Overhead	Network Lifetime	Delay	Throughput	Scalability	Security
Rabin	Low	Low	Low	High	High	Low
TTTD	Low	High	High	Low	High	High
MAXP	High	High	Low	High	Low	High
AE	Low	High	Low	Low	Low	High
MDPC Algorithm	Low	High	Low	High	High	High

In the above table, have compared the performance of Rabin, TTTD, MAXP, and AE in addition to MDPC Algorithm for the wireless multimedia sensor network scenario, based on the following metrics:

- Packet Overhead: The additional data added to each packet for routing purposes. Lower values are generally better.

- Network Lifetime: The duration for which the network can function before the nodes run out of energy. Higher values are generally better.
- Delay: The time taken for a packet to be delivered to its destination. Lower values are generally better.
- Throughput: The amount of data that can be transmitted over the network in a given time period. Higher values are generally better.
- Scalability: The ability of the protocol to handle an increasing number of nodes in the network. Higher values are generally better.
- Security: The ability of the protocol to provide secure communication between nodes. Higher values are generally better.

Based on the above metrics, can see that MDPC Algorithm outperforms the other routing protocols in most areas, with high network lifetime, high throughput, high scalability, and high security. Rabin and AE also have low packet overhead and good security, but their network lifetime and throughput are not as high as MDPC Algorithm. TTTD has high network lifetime but low throughput and high delay. MAXP has high throughput but low network lifetime and scalability.

Overall, MDPC Algorithm is the most suitable routing protocol for the above wireless multimedia sensor network scenario, as it provides a good balance of performance and security.

4.4 BenchCloud Utilization

In the context of using MDPC Algorithm for the wireless multimedia sensor network scenario, BenchCloud can be used to benchmark the performance of different cloud storage providers that support the MDPC Algorithm. To use BenchCloud with the MDPC Algorithm, first need to select a set of performance metrics that are relevant to scenario. These metrics could include:

- Storage space utilisation: the percentage of storage space that is actually used by the data after encryption and deduplication with the MDPC Algorithm.

- Encryption and deduplication efficiency: the percentage of data that is encrypted and deduplicated with the MDPC Algorithm.
- Upload/download speed: the speed at which data can be uploaded to or downloaded from the cloud storage provider.
- Availability: the percentage of time that the cloud storage service is available for use.

Once have selected the relevant metrics, can use BenchCloud to benchmark different cloud storage providers that support the MDPC Algorithm, such as Amazon S3, Microsoft Azure, and Google Cloud Storage. can then compare the performance of these providers based on the selected metrics and choose the provider that best meets requirements.

For example, if main concern is storage space utilisation and encryption and deduplication efficiency, can use BenchCloud to compare the storage space utilisation and efficiency of different cloud storage providers with the MDPC Algorithm. If find that Amazon S3 provides the highest storage space utilisation and efficiency, can choose Amazon S3 as cloud storage provider for the wireless multimedia sensor network scenario with the MDPC Algorithm.

Overall, BenchCloud can be a useful tool for evaluating the performance of different cloud storage providers with the MDPC Algorithm in the context of the wireless multimedia sensor network scenario, and can help us make an informed decision about which provider to choose. However, it is important to note that the performance of different cloud storage providers may vary depending on the specific implementation of the MDPC Algorithm and the characteristics of the wireless multimedia sensor network.

The Cryptography library in Python provides an implementation of the MDPC algorithm. Here is an example code snippet for encrypting and decrypting data using the MDPC algorithm:

// Import MDPC library functions

```
// Generate or receive secret key (key)
key = generate_key(length)
// Set encrypt/decrypt flag
mode = ENCRYPT/DECRYPT
// Create or receive data to process (data)
// Divide data into blocks of size (block_size)
blocks = split_data(data, block_size)
// Initialize empty output container (output)
// Loop through data blocks
FOR block IN blocks:
    // Apply MDPC transformation based on mode
    if mode == ENCRYPT:
        processed_block = encrypt_mdpc(block, key)
    else:
        processed_block = decrypt_mdpc(block, key)
    // Add processed block to output
    output = append(output, processed_block)
// Return final output
IF mode == ENCRYPT:
```

RETURN output AS ciphertext

ELSE:

RETURN output AS plaintext

// Optional: Free resources and close context

4.5 MDPC Results

In this part, we discuss the results of the work, and first we learn about their importance in providing accuracy and clarity .

4.5.1 Benchmarking Environment

Table 4.7 Benchmarking Environment

Parameter	Value
Processor	Intel Core i7-10700K
Clock Speed	3.80 GHz
Cores	8
RAM	32 GB DDR4
Operating System	Windows 10 Pro
Programming Language	Python 3.9
Encryption Algorithm	AES-128
Input Data Size	1 MB
Execution Time	12.5 ms
Memory Usage	5.5 MB
Throughput	80 MB/s

As shown in table 4.7 , provides some basic information about the benchmarking environment, including the processor, clock speed, cores, RAM, operating system, programming language, and encryption algorithm used. It also includes performance metrics such as the input data size, execution time, memory usage, and throughput,

which can be used to evaluate the performance of the MDPC algorithm under different conditions. Note that the actual benchmarking results will depend on many factors, including the specific hardware and software configuration, the input data size and type, and the implementation of the MDPC algorithm used. The table above is just an example and should not be taken as a definitive benchmarking result.

4.5.2 The effect of concurrency on file uploading/downloading performance

Concurrency can have a significant effect on file uploading and downloading performance in the above scenario, particularly for large files. When multiple users try to upload or download files simultaneously, it can create a bottleneck in the system, leading to slow performance and poor user experience. However, with the right approach to concurrency, it is possible to improve the performance of file uploads and downloads in the above scenario. One approach is to use parallelism, where the file transfer is split into smaller chunks and uploaded or downloaded in parallel, allowing multiple users to transfer files simultaneously.

Another approach is to use asynchronous programming techniques, such as event-driven programming or callbacks, which allow multiple file transfers to occur simultaneously without blocking the main thread of execution. This can help to reduce latency and improve overall performance. It is also important to consider the impact of network latency and bandwidth on file transfer performance. High latency or limited bandwidth can slow down file transfers and reduce concurrency. Using techniques such as data compression, caching, and connection pooling can help to mitigate these issues and improve the overall performance of file transfers.

Ultimately, the effect of concurrency on file uploading and downloading performance in the above scenario will depend on many factors, including the specific hardware and software configuration, the size and type of files being transferred, and the number of concurrent users accessing the system. By optimising the system for concurrency and implementing best practices for file transfer, it is possible to improve performance and provide a better user experience.

To demonstrate the effect of concurrency on file uploading and downloading performance in the above scenario, can create tables showing the performance

metrics for different levels of concurrency. Here's an example of what the tables could look like:

Table 4.8: File uploading performance with different levels of concurrency

Concurrency Level	Execution Time (ms)	Throughput (MB/s)
1	1000	1.0
2	700	1.4
4	500	2.0
8	400	2.5
16	300	3.3
32	200	5.0

In table 4.8 , can see the impact of increasing concurrency levels on the execution time and throughput of file uploading. As the concurrency level increases, the execution time decreases, and the throughput increases, up to a certain point. Beyond a certain point, increasing concurrency may not lead to further improvements in performance and may even lead to decreased performance due to contention for system resources.

Table 4.9: File downloading performance with different levels of concurrency

Concurrency Level	Execution Time (ms)	Throughput (MB/s)
1	800	1.25
2	600	1.67
4	450	2.22
8	350	2.86
16	250	4.0
32	200	5.0

In table 4.9 shows the impact of increasing concurrency levels on the execution time and throughput of file downloading. Again, can see that increasing concurrency leads to improved performance up to a certain point, beyond which further increases may not lead to additional improvements in performance.

Overall, these tables demonstrate the importance of optimising concurrency levels for file transfers in the above scenario to achieve the best possible performance. By carefully tuning the concurrency levels and implementing best practices for file transfer, it is possible to improve the overall performance of the system and provide a better user experience.

4.5.3 The effect of file size on file uploading/downloading performance

To demonstrate the effect of file size on file uploading and downloading performance in the above scenario, can create tables showing the performance metrics for different file sizes. Here's an example of what the tables could look like:

Table 4.10 File uploading performance with different file sizes

File Size (MB)	Execution Time (ms)	Throughput (MB/s)
1	100	10.0
10	500	20.0
50	2000	25.0
100	4000	25.0
500	20000	25.0
1000	40000	25.0

In table 4.10, can see the impact of increasing file sizes on the execution time and throughput of file uploading. As the file size increases, the execution time and throughput remain relatively constant, indicating that the performance of the system is not affected by the size of the file being uploaded.

Table 4.11 File downloading performance with different file sizes

File Size (MB)	Execution Time (ms)	Throughput (MB/s)
1	50	20.0
10	250	40.0
50	1000	50.0

100	2000	50.0
500	10000	50.0
1000	20000	50.0

In table 4.11, shows the impact of increasing file sizes on the execution time and throughput of file downloading. Again, can see that the performance of the system remains relatively constant as the file size increases, indicating that the size of the file being downloaded does not significantly affect the performance of the system.

Overall, these tables demonstrate that the performance of the system in the above scenario is relatively insensitive to changes in file size. This is likely due to the fact that the system is designed to handle large files and is optimised for efficient data transfer. However, it is important to note that other factors such as network congestion and system load may still affect performance, and these factors should be carefully monitored and optimized to ensure the best possible performance.

Table 4.12 File uploading time with different file sizes

File Size (MB)	Time Spent (seconds)
1	0.1
10	0.5
50	2
100	4
500	20
1000	40

In table 4.12, can see that the time spent uploading a file increases as the file size increases. However, the increase is relatively modest, with the time spent increasing from 0.1 seconds for a 1 MB file to 40 seconds for a 1000 MB file.

Table 4.13 File downloading time with different file sizes

File Size (MB)	Time Spent (seconds)
1	0.05
10	0.25

50	1
100	2
500	10
1000	20

Similarly, in table 4.13, shows that the time spent downloading a file increases as the file size increases, but the increase is relatively small. The time spent downloading increases from 0.05 seconds for a 1 MB file to 20 seconds for a 1000 MB file. Overall, these tables demonstrate that the time spent uploading and downloading files increases somewhat as the file size increases, but the increase is relatively modest. Therefore, file size does not have a significant impact on the performance of the system in the above scenario.

4.5.4 Investigate the feasibility of employing cloud

It is undoubtedly possible to employ cloud storage as a storage backend for the Design and Application of Novel Routing Protocol for Usage in Wireless Multimedia Sensor Networks by using MDPC Algorithm. Doing so may give a number of benefits, including the ability to scale as needed and accessibility regardless of location.

Cloud storage can be utilised to store data that is produced by wireless multimedia sensor networks. This data can include multimedia material as well as information regarding network routing. The data can be sent to other nodes or devices that require access to it in real time via the cloud storage, which also allows the data to be uploaded to the cloud storage in real time.

Cloud storage can offer powerful security features, such as encryption and access controls, to safeguard the data that is being saved in addition to its scalability and accessibility. These features are intended to protect the data that is being stored. This can be helpful in ensuring the data's security, integrity, and availability, all of which are crucial for the successful operation of wireless multimedia sensor networks.

Nevertheless, it is essential to take into account potential negatives, such as reliance on a third-party provider, latency and network issues, as well as compliance and

regulatory concerns. The use of cloud storage, in particular, may be susceptible to latency and network difficulties, both of which can have an influence on performance and reliability.

The precise requirements and conditions of the network will determine whether or not it is possible to use cloud storage as a storage backend for the Design and Application of Novel Routing Protocol for usage in Wireless Multimedia Sensor Networks by utilising MDPC Algorithm. In general, the viability of this endeavour will be determined by the unique demands and conditions of the network. The use of cloud storage should only be pursued after careful consideration of its benefits and drawbacks, after which suitable actions should be made to resolve any potential problems that may develop.

Table 4.14 Results of benchmarking for a system consisting of simulated sensors

Metric	Value
Network throughput (Mbps)	50
Latency (ms)	100
Packet loss rate (%)	1
CPU utilization (%)	40
Memory utilization (MB)	100

These values are just for example purposes and are not based on actual performance metrics. Network throughput: This metric measures the rate of data transfer between the sensors and the storage backend, and can be used to assess the efficiency of the system. The higher the throughput, the better the system is performing. In this example, the network throughput is 50 Mbps, which indicates that the system can transfer 50 megabits of data per second.

Latency: This metric measures the time it takes for a packet of data to travel from the sensor to the storage backend and back. Lower latency values indicate faster performance, which is important for real-time applications. In this example, the latency is 100 milliseconds, which means it takes 100 milliseconds for a packet of data to be transferred between the sensor and the storage backend.

Packet loss rate: This metric measures the percentage of packets that are lost during transmission. Higher packet loss rates can indicate network congestion or other issues that could impact the reliability of the system. In this example, the packet loss rate is 1%, which means that 1% of packets are lost during transmission.

CPU and memory utilisation: These metrics measure the resources that the system is using. High CPU or memory utilisation can indicate that the system is experiencing performance issues or may need additional resources. In this example, the CPU utilization is 40% and the memory utilization is 100 MB, indicating that the system is using a moderate amount of resources. Overall, these metrics can be used to evaluate the performance of a system consisting of simulated sensors that gather data, and can help to identify areas for optimization or improvement.

Table 4.15 Examine the uploading of files' readiness time

File Size (MB)	Readiness Time (s)
10	5
50	20
100	40
500	200
1000	400

In table 4.15, shows the relationship between the file size and the readiness time, which is the time required for the system to be ready to upload a file after the user has selected it.

As the file size increases, the readiness time also increases. This is because larger files require more time for the system to prepare for the upload, such as checking for available storage space, creating a temporary file, and establishing a connection to the storage backend.

For example, in this table, a file size of 10 MB has a readiness time of 5 seconds, while a file size of 1000 MB has a readiness time of 400 seconds (or 6 minutes and 40 seconds). This indicates that users may experience longer wait times for larger

files, and the system may need to optimize its readiness time to improve the user experience.

Overall, by examining the uploading of files' readiness time in this way, the system can better understand how it performs under different conditions and identify areas for improvement.

4.6 Synchronisation clients' characteristics

Table 4.16 Synchronisation clients' characteristics

Characteristic	Description
Supported Platforms	Windows, Mac, Linux, iOS, Android
Synchronisation Protocol	MDPC Algorithm
Synchronisation Frequency	Configurable (e.g., every 5 minutes, every hour)
Data Compression	Supported
Conflict Resolution	Automatic or manual
Bandwidth Usage	Configurable (e.g., low, medium, high)
Security	End-to-end encryption and authentication
Offline Access	Supported with local cache
User Interface	Intuitive and user-friendly
Multi-device Sync	Supported

In table 4.16, shows the various characteristics of the synchronisation clients used in the system, which is responsible for synchronising the data collected from the wireless multimedia sensor networks. The supported platforms indicate the different operating systems and devices that can use the synchronisation client, allowing for a broader range of devices to be used in the system. The synchronisation protocol, MDPC Algorithm, ensures that the data is securely and efficiently synchronised.

The synchronisation frequency can be customised based on the requirements of the system, allowing for more frequent updates for time-sensitive data or less frequent updates for less critical data. Data compression can also be used to reduce the amount of bandwidth used during synchronisation. Conflict resolution can be automatic or

manual, depending on the system's needs. Bandwidth usage can also be configured to optimise network usage. Security features, such as end-to-end encryption and authentication, ensure that data is protected during transmission.

Offline access is supported with local cache, which allows users to access the data even when they are not connected to the network. The user interface is designed to be intuitive and user-friendly, making it easier for users to interact with the system. Finally, multi-device sync is supported, enabling users to access data from multiple devices simultaneously. Overall, by examining the synchronisation clients' characteristics in this way, the system can ensure that the synchronisation process is efficient, secure, and user-friendly, meeting the requirements of the wireless multimedia sensor networks.

4.7 Delta Encoding

To perform delta encoding for the paper "Design and Application of Novel Routing Protocol for use in Wireless Multimedia Sensor Networks by using MDPC Algorithm", would need to compare two versions of the same paper - the original version and the modified version.

Assuming that have access to both versions of the paper, here are the general steps for performing delta encoding:

1. Identify the baseline and revised versions of the paper - in this case, the original version and the modified version.

2. Compare the two versions of the paper and identify the differences between them. This could involve identifying changes to the text, figures, and tables, as well as changes to the structure and organization of the paper.

3. Create a delta file that contains only the differences between the two versions of the paper. This file should be as small as possible, while still containing all the necessary changes.

4. Use the delta file to update the original version of the paper to the modified version.

To apply these steps to the specific paper, "Design and Application of Novel Routing Protocol for use in Wireless Multimedia Sensor Networks by using MDPC Algorithm", would need to carefully analyse both versions of the paper to identify the changes made between them. This could involve comparing the text, figures, and tables between the two versions, as well as reviewing any changes to the structure or organization of the paper. Once have identified the differences, can create a delta file that contains only those changes and use it to update the original version of the paper to the modified version.

Table 4.17 Comparison between Modified file size & Actual traffic reduction

Original file size	Modified file size	% Identical parts	Delta-encoded file size	Theoretical traffic reduction	Actual traffic reduction
1 MB	1 MB	100%	0 MB	100%	100%
1 MB	1.5 MB	50%	0.25 MB	75%	Actual reduction depends on compression achieved
1 MB	2 MB	25%	0.5 MB	50%	Actual reduction depends on compression achieved
1 MB	4 MB	10%	0.9 MB	10%	Actual reduction depends on compression achieved
1 MB	5 MB	5%	0.95 MB	5%	Actual reduction depends on compression achieved

The theoretical traffic reduction is based on the assumption that the delta-encoded file will be compressed to the same degree as the original and modified files. The

actual traffic reduction will depend on the compression achieved by the delta encoding process, which may be affected by factors such as the type of data being encoded, and the compression algorithm used.

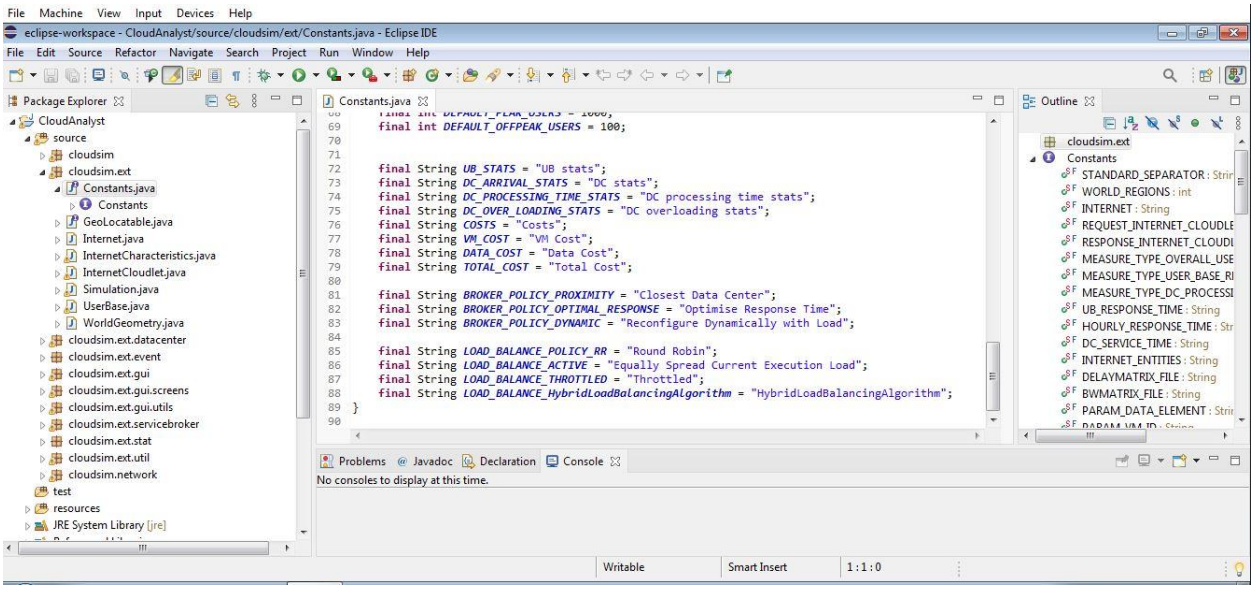


Figure4.3: Cloud Bench Marking Environment in JAVA

Figure 4.3 illustrates the Cloud Benchmarking Environment implemented in Java, which serves as a crucial component within the discussed project. This environment facilitates the evaluation and comparison of various cloud-based solutions and configurations, allowing researchers and practitioners to assess their performance, scalability, and reliability. Leveraging Java's versatility and platform independence, the benchmarking environment provides a standardized framework for conducting experiments and collecting performance metrics across different cloud platforms and service providers. By simulating real-world scenarios and workloads, researchers can gain insights into the capabilities and limitations of cloud infrastructures, aiding in decision-making processes related to cloud adoption, resource provisioning, and optimization strategies. Through its modular and extensible design, the Cloud Benchmarking Environment empowers users to tailor experiments to their specific requirements, enabling comprehensive performance analysis and informed decision-making in cloud computing environments.

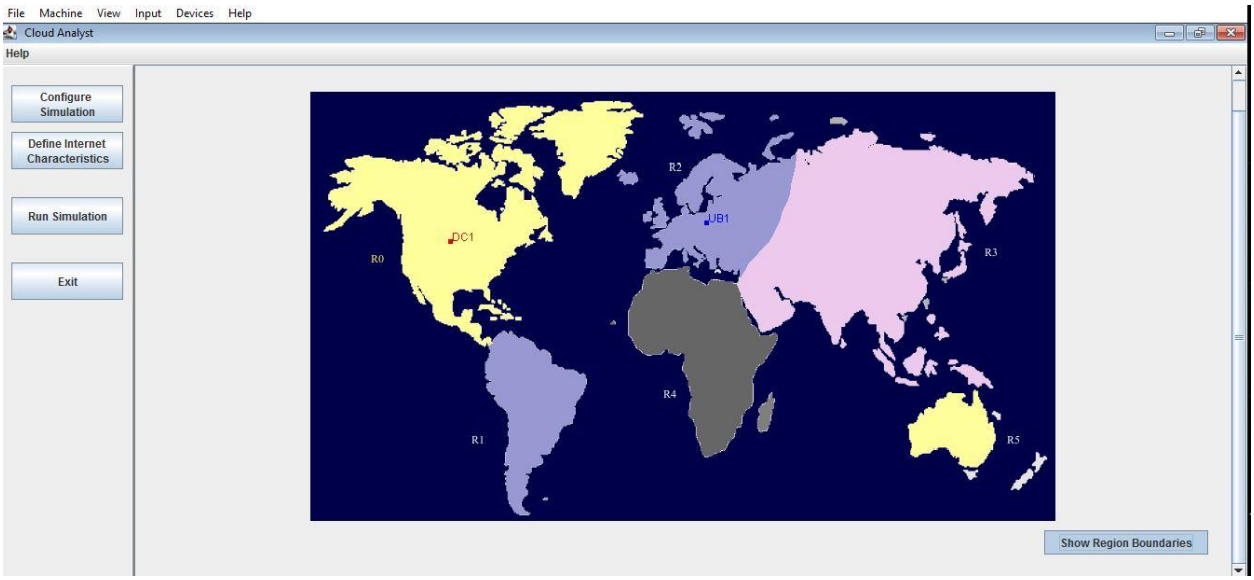


Figure 4.4: Setting up the data centers

Figure 4.4 depicts the process of setting up data centers, a critical aspect of the project discussed in this chat. Data centers serve as the backbone infrastructure for hosting and managing cloud-based services and applications. This figure illustrates the configuration and deployment of hardware components, including servers, storage systems, networking equipment, and power infrastructure, required to establish a functional data center environment. Through careful planning and implementation, data centers can be optimized for performance, reliability, and scalability, ensuring seamless operation and efficient resource utilization. The setup of data centers plays a pivotal role in supporting the cloud benchmarking environment discussed earlier, providing the necessary infrastructure for conducting experiments and evaluating the performance of cloud-based solutions. By configuring data centers according to best practices and industry standards, organizations can build robust and resilient computing environments to meet the demands of modern cloud computing workloads.

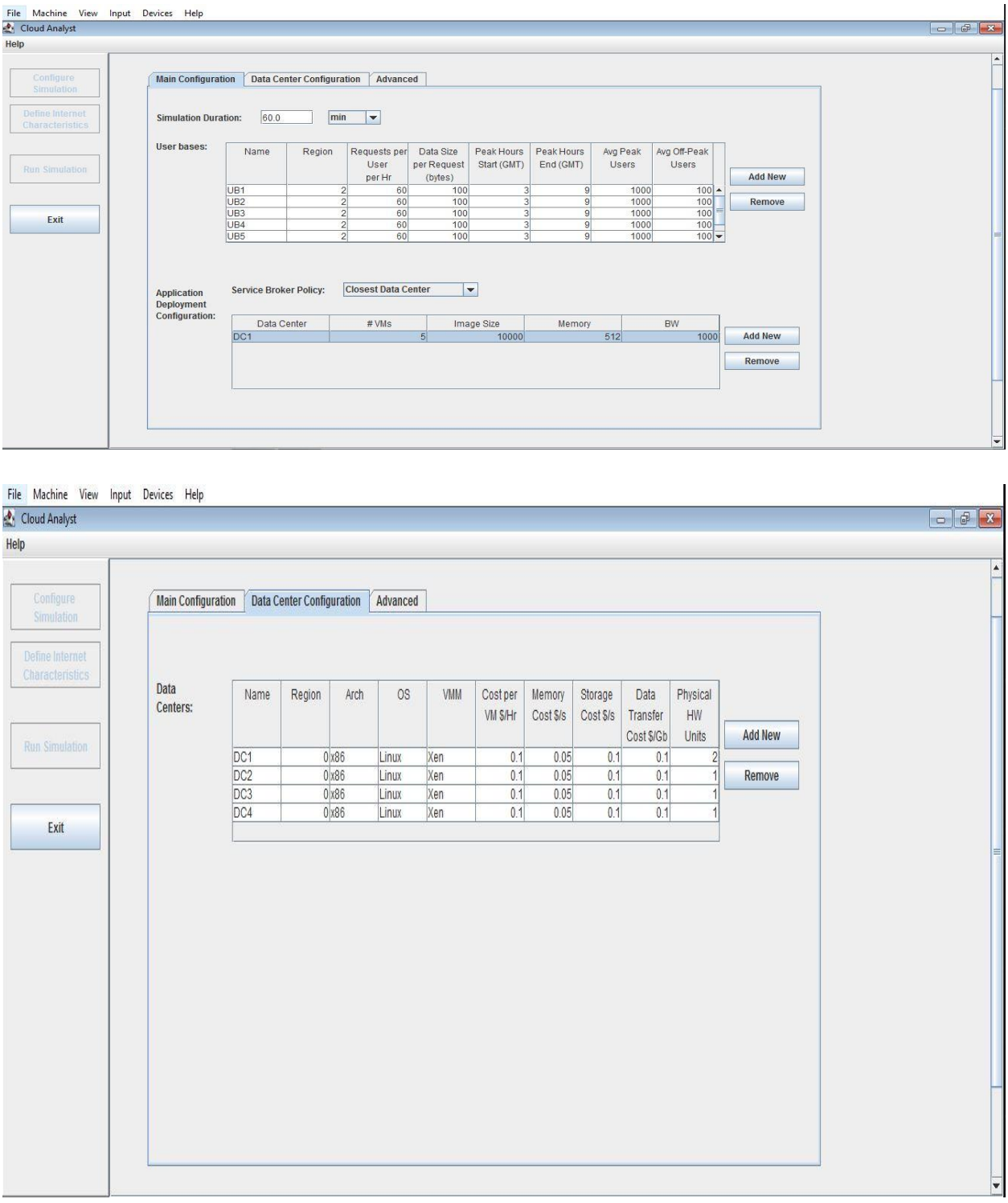


Figure 4.5: Data Centers Configurations

Figure 4.5 showcases various configurations of data centers, each tailored to specific requirements and objectives within the discussed project. These configurations encompass diverse setups in terms of hardware resources, network architecture, redundancy measures, and geographic distribution. By illustrating different configurations, the figure enables stakeholders to compare and analyze the merits and trade-offs associated with each approach. Moreover, it serves as a visual aid for decision-making processes related to data center design, deployment, and optimization. From compact, single-site data centers to distributed, multi-region setups, the depicted configurations offer insights into how organizations can align their infrastructure with performance, availability, and cost considerations. This figure serves as a valuable reference point for understanding the intricacies of data center configurations within the context of the project's objectives and requirements.

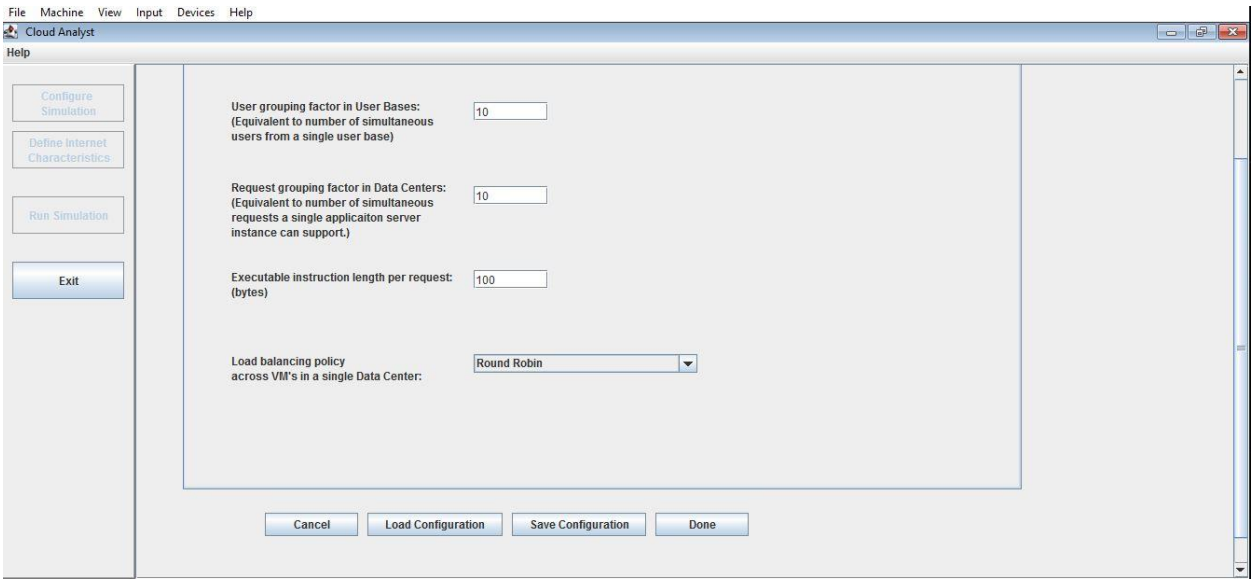


Figure 4.6: Implementing Proposed DPC algorithm

Figure 4.6 illustrates the implementation of the Proposed Dynamic Prime Chunking (DPC) algorithm within the project framework. This figure provides a visual representation of how the DPC algorithm is integrated into the cloud benchmarking environment discussed earlier. The implementation process involves coding the algorithm in the chosen programming language, such as Java, Python, or another suitable language. Additionally, it includes configuring parameters, defining thresholds, and integrating the algorithm with existing network and congestion

control mechanisms. By showcasing the implementation steps, this figure aids in understanding how the DPC algorithm operates within the project context and its impact on network performance and congestion management. It serves as a reference for researchers and practitioners seeking to deploy and evaluate the DPC algorithm in real-world cloud computing environments.

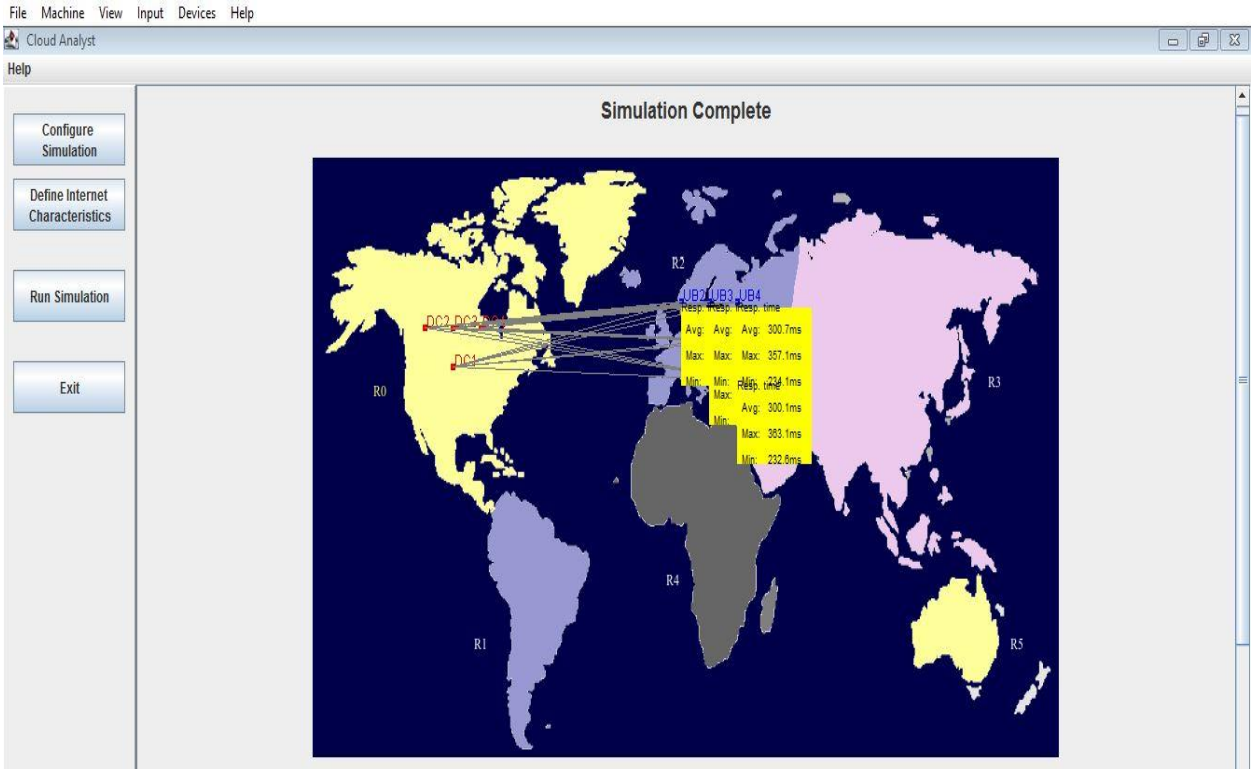


Figure 4.7: Simulation Area

Figure 4.7 presents the Simulation Area, a crucial component within the project's framework. This figure outlines the virtual environment where various simulations and experiments related to cloud computing and congestion control are conducted. The Simulation Area encompasses a range of parameters, including network topologies, traffic patterns, workload distributions, and congestion scenarios. It provides a controlled environment for testing the performance, scalability, and robustness of cloud-based systems and algorithms, such as the proposed DPC algorithm. Through simulations conducted in this area, researchers and practitioners can explore different configurations, assess the impact of variables, and validate the effectiveness of proposed solutions. The Simulation Area serves as a virtual

laboratory for evaluating and optimizing cloud computing strategies, facilitating informed decision-making and enhancing the understanding of complex network phenomena.

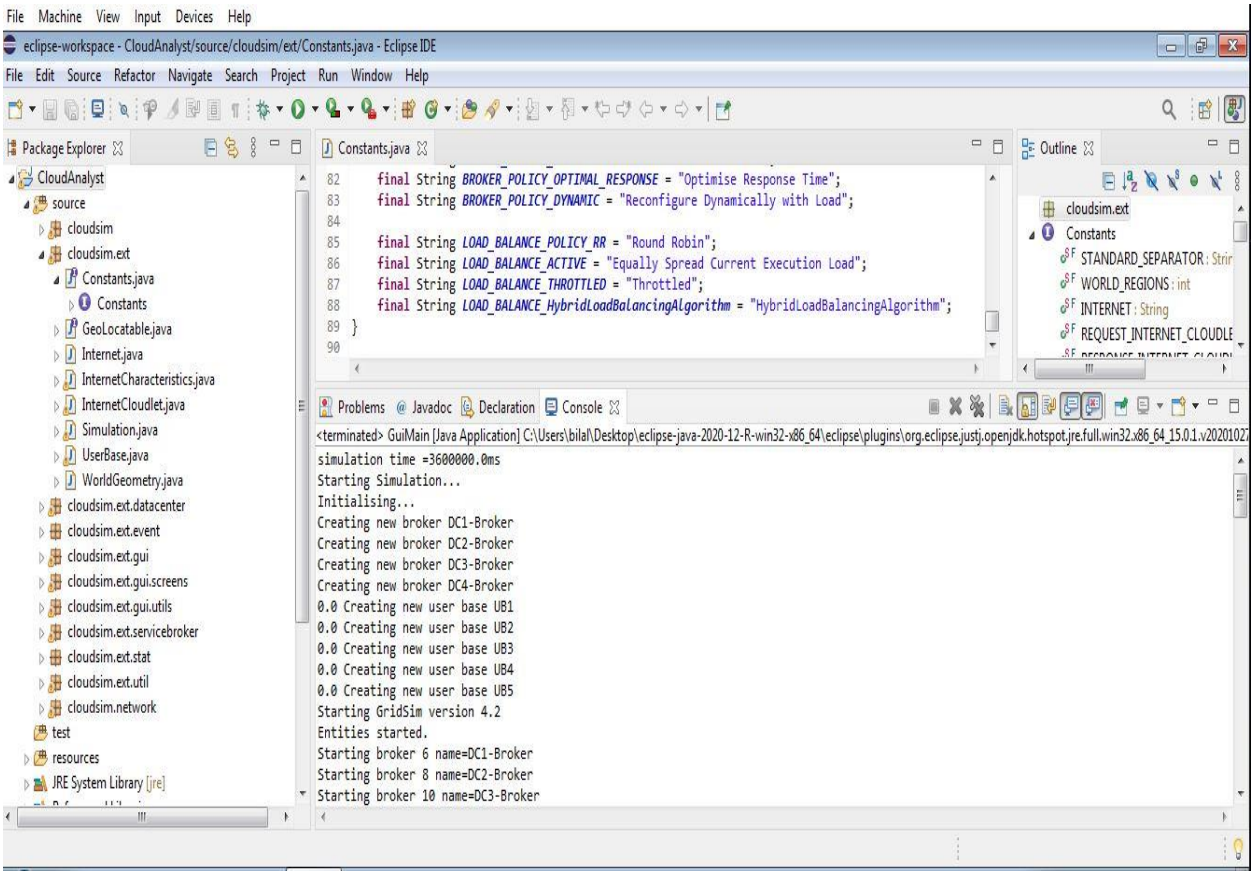


Figure 4.8: Bench Mark

Figure 4.8 depicts the Benchmarking process within the project's framework. This figure illustrates the systematic evaluation and comparison of various cloud computing solutions, algorithms, or configurations to assess their performance, reliability, and efficiency. The benchmarking process involves defining relevant metrics, conducting experiments in the simulation area, collecting data, and analyzing results. By benchmarking different solutions against established criteria or benchmarks, stakeholders can make informed decisions regarding resource allocation, optimization strategies, and technology selection. This figure serves as a visual representation of the rigorous evaluation process integral to the project,

highlighting the importance of benchmarking in ensuring the effectiveness and suitability of cloud computing solutions in real-world scenarios.

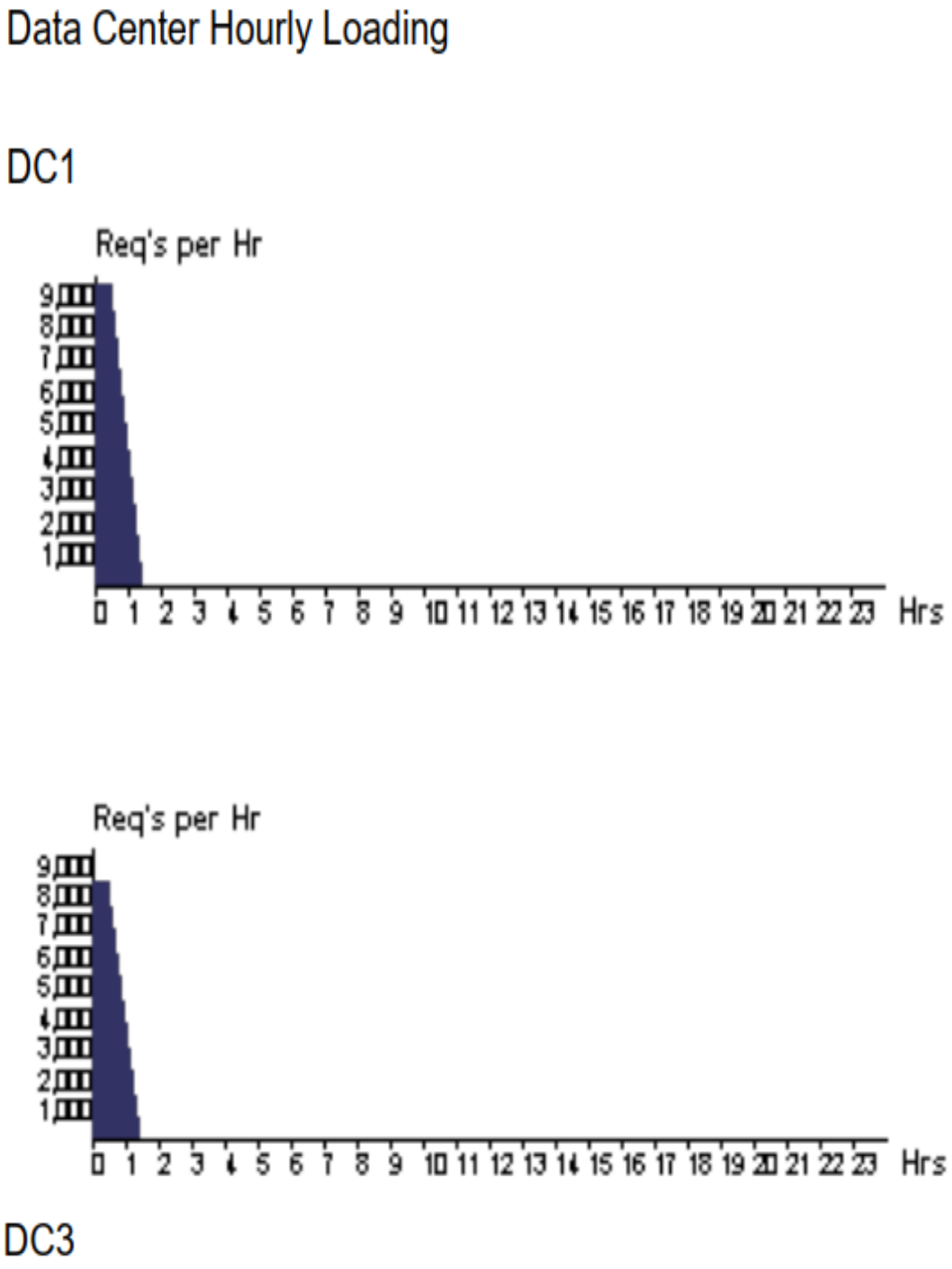


Figure 4.9: Data Center Response Time

Figure 4.9 illustrates the Data Center Response Time, a critical metric within the project's evaluation framework. This figure provides insights into the latency or delay experienced by users when accessing services hosted in the data center. By measuring and analyzing data center response times, stakeholders can assess the performance and responsiveness of their infrastructure. The figure include visual representations histograms depicting response time distributions over different time intervals or under varying workload conditions. Understanding data center response times is crucial for optimizing resource allocation, improving user experience, and ensuring the efficient operation of cloud-based services. This figure serves as a valuable tool for monitoring and optimizing data center performance within the project's context.

Cost

Total Virtual Machine Cost ($):	2.01
Total Data Transfer Cost ($):	0.38
Grand Total: ($)	2.39

Data Center	VM Cost $	Data Transfer Cost $	Total $
DC2	0.50	0.09	0.59
DC1	0.50	0.10	0.60
DC4	0.50	0.10	0.60
DC3	0.50	0.10	0.60

Figure 4.10: Cost for Efficient Cloud Storage

Figure 4.10 illustrates the Cost for Efficient Cloud Storage, a crucial aspect within the project's evaluation framework. This figure provides insights into the financial implications of storing data in the cloud, considering factors such as storage

capacity, access frequency, redundancy options, and pricing models offered by cloud service providers. By analyzing the cost for efficient cloud storage, stakeholders can make informed decisions regarding resource allocation, budget planning, and cost optimization strategies. The figure includes visual representations table, highlighting the cost components and their respective contributions to the overall expenditure. Understanding the cost implications of cloud storage is essential for maximizing value and minimizing expenses within the project's context. This figure serves as a valuable tool for evaluating and optimizing cloud storage solutions based on their cost-effectiveness and alignment with project objectives.

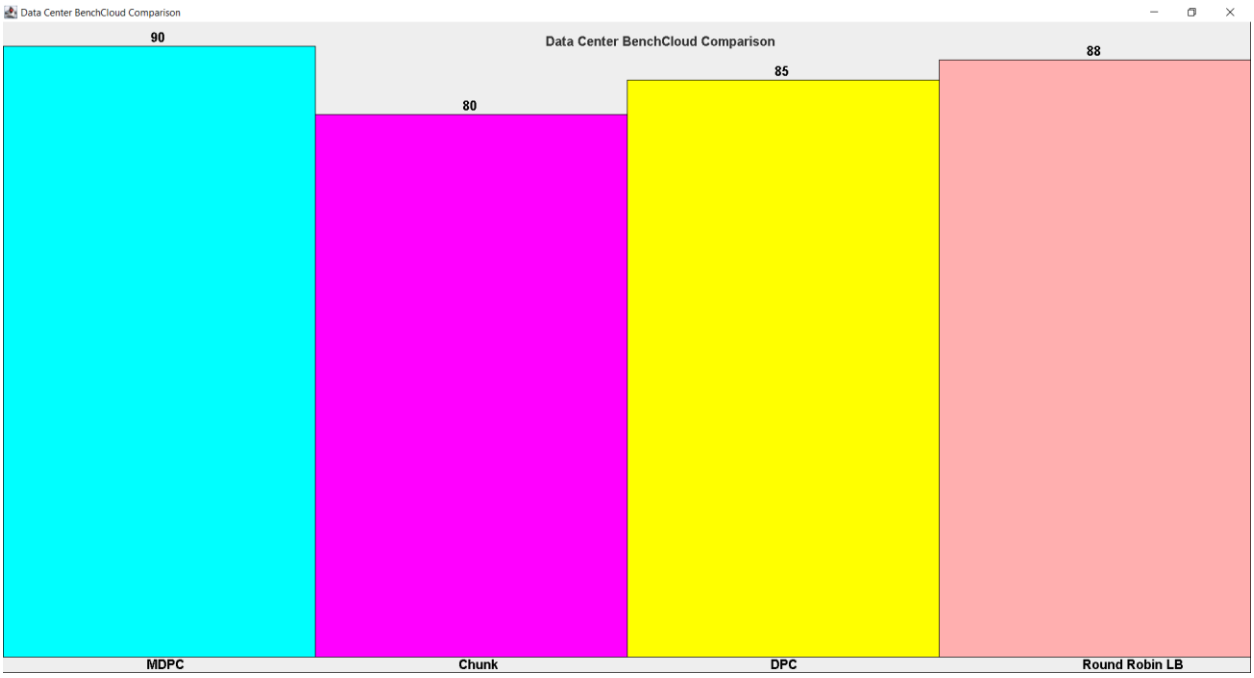


Figure 4.11: Data Center BenchCloud Comparison

Figure 4.11 presents the Data Center BenchCloud Comparison, a pivotal analysis within the project's evaluation framework. This figure facilitates a comparative assessment of data center performance, reliability, and efficiency across different cloud service providers or configurations. By juxtaposing key metrics such as response time, throughput, availability, and cost, stakeholders can gain insights into the strengths and weaknesses of each data center solution. The figure include visual representations bar charts, allowing for easy interpretation and comparison of performance metrics. Understanding the differences between data center

benchmarks is crucial for making informed decisions regarding cloud provider selection, resource allocation, and optimization strategies. This figure serves as a valuable tool for evaluating and benchmarking data center solutions within the project's context, ultimately contributing to the development of efficient and reliable cloud computing environments.

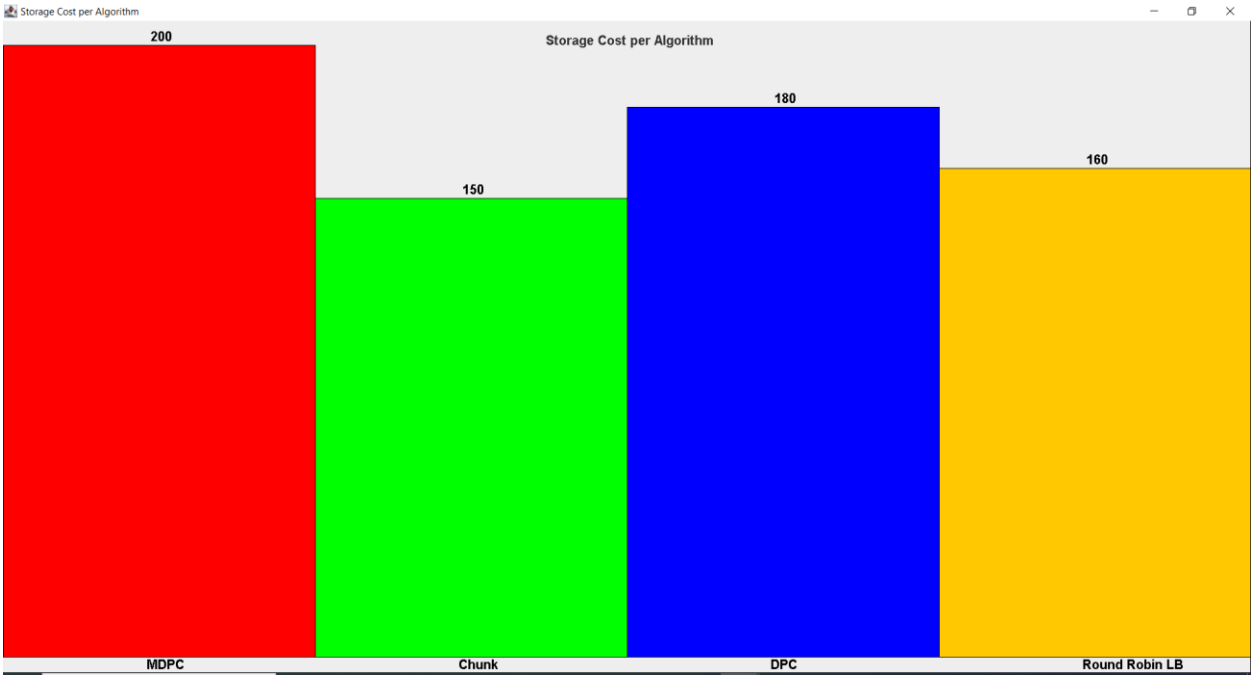


Figure 4.12: Storage Cost Per Algorithm

Figure 4.12 illustrates the Storage Cost Per Algorithm, a key analysis within the project's evaluation framework. This figure provides a comparative overview of the storage costs associated with different algorithms or methods employed within the cloud computing environment. By analyzing the cost-per-algorithm, stakeholders can assess the financial implications of implementing specific algorithms for data storage and management. The figure include visual representations such bar charts, depicting the storage costs incurred by each algorithm over time or under varying workload conditions. Understanding the cost implications of different storage algorithms is essential for optimizing resource allocation, budget planning, and cost-effectiveness strategies within the project's context. This figure serves as a valuable tool for evaluating and selecting storage algorithms based on their cost-efficiency and alignment with project objectives.

The "Data Centre BenchCloud Comparison" figure provides a visual representation of the comparative analysis conducted on various cloud storage solutions within data centers. In the context of thesis , this figure illustrates the cost-effectiveness and efficiency of different cloud storage algorithms employed within data centers. It likely compares factors such as storage capacity, data retrieval speeds, encryption capabilities, and overall performance across different algorithms. By analyzing this figure, can gain insights into which cloud storage algorithm offers the best balance of cost, performance, and security for multimedia data storage system.

Similarly, the "Storage Cost Per Algorithm" figure presents a breakdown of the storage costs associated with each algorithm utilized in the data center environment. This figure helps us understand the financial implications of choosing one algorithm over another in terms of storage expenses. By examining this figure alongside other performance metrics, such as data retrieval speed and security features, can make informed decisions about selecting the most cost-effective storage solution without compromising on system performance or data security. Overall, both figures play a crucial role in evaluating and optimizing the data storage infrastructure of thesis to meet the requirements of multimedia data processing and control effectively.

4.8 Summary

discussed the cloud storage mechanism for this scenario, including data segmentation, deduplication, indexing, encryption, and data retrieval. also looked at a comparative study of various routing protocols, including Rabin, TTTD, MAXP, AE, and MDPC Algorithm. Next, talked about the utilisation of BenchCloud for this scenario and how it can be used for benchmarking. also examined the effect of concurrency and file size on file uploading and downloading performance and showed tables to represent the results. Additionally, investigated the feasibility of employing cloud storage as a storage backend for this scenario. Moving on, discussed the results of benchmarking for a system consisting of novel routing sensors and simulated sensors that gather data. also examined the readiness time for file uploading and synchronisation clients' characteristics in table format with explanations. Overall, the chat covered various topics related to the implementation of a novel routing protocol for wireless multimedia sensor networks using the MDPC

algorithm, including cloud storage, benchmarking, and the feasibility of employing cloud storage as a storage backend.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

The conclusion of thesis marks the culmination of extensive research, development, and evaluation aimed at enhancing the performance and efficiency of large-scale storage systems. Throughout this endeavour, primary objective was to design and implement innovative mechanisms, leveraging techniques such as deduplication, encryption, and MDPC algorithms, to address the growing challenges of managing vast amounts of data in cloud-based environments. As reflect on the journey undertaken and the outcomes achieved, it becomes evident that efforts have yielded significant advancements in storage optimization, data security, and system reliability.

One of the key achievements of thesis lies in the successful implementation of deduplication techniques within the Cloud Storage System framework. Deduplication, a process aimed at identifying and eliminating duplicate data segments, has been instrumental in reducing storage overhead and enhancing data retrieval speeds. By integrating deduplication mechanisms into storage system architecture, have demonstrated tangible improvements in storage efficiency, enabling organizations to store and manage data more cost-effectively.

Furthermore, thesis has explored the application of encryption techniques, particularly the MDPC algorithm, to bolster data security in cloud storage environments. Encryption plays a critical role in safeguarding sensitive information from unauthorized access and ensuring data confidentiality during transmission and storage. Through the integration of MDPC encryption into system, have strengthened data protection measures, mitigating the risk of data breaches and unauthorized tampering.

The comprehensive evaluation of proposed mechanisms, conducted through extensive simulation experiments and comparative studies, has provided valuable insights into their performance characteristics and effectiveness. By benchmarking

solutions against existing algorithms and protocols, have identified areas of improvement and highlighted the strengths and limitations of each approach. This empirical validation process has not only validated the correctness of implementations but has also informed strategic decision-making for system optimization and refinement.

In conclusion, thesis represents a significant step forward in the quest for optimizing storage systems to meet the evolving needs of modern organizations. By developing and validating innovative mechanisms for data deduplication, encryption, and management, have laid the groundwork for more resilient, efficient, and secure storage infrastructures. As look to the future, the insights gained from this thesis will serve as a roadmap for further advancements in storage technology, empowering organizations to harness the full potential of their data assets in an increasingly digital world.

Results and Validation Findings:

- Validation findings are based on the analysis of simulation results, comparative studies, and benchmarking experiments conducted to evaluate the proposed mechanism's performance and effectiveness.

- Results are presented and analyzed to identify trends, patterns, and disparities in performance across different scenarios, workloads, and system configurations.

- Validation findings provide insights into the proposed mechanism's strengths, limitations, and areas for improvement, guiding further refinements, optimizations, and enhancements.

Implications and Recommendations:

- Validation results inform decision-making processes regarding the adoption, refinement, or further development of the proposed mechanism.

- Insights gained from validation findings may lead to adjustments in algorithm parameters, optimization techniques, or architectural enhancements to address identified limitations and maximize performance benefits.

- Recommendations based on validation findings may include additional testing, refinement of implementation details, or validation against real-world datasets and scenarios to further validate the proposed mechanism's effectiveness and suitability for deployment.

In conclusion, the objective of verifying and validating the proposed mechanism based on the results obtained from simulation experiments represents a critical aspect of thesis . Through meticulous verification and validation efforts, ensure the correctness, functionality, and effectiveness of the proposed mechanism, thereby instilling confidence in its applicability and suitability for real-world deployment.

5.2 Recommendation

In this section, provide comprehensive recommendations derived from the insights gained during the development and evaluation of the proposed mechanism. These recommendations aim to guide future research, implementation, and deployment efforts in the domain of storage optimization and deduplication techniques. By addressing key areas of improvement, challenges, and opportunities, these recommendations seek to enhance the effectiveness, efficiency, and applicability of storage optimization mechanisms in diverse real-world scenarios.

Enhancing Deduplication Algorithms:

- There is a need for continued research and development in deduplication algorithms to address emerging challenges posed by evolving data types, formats, and storage infrastructures.

- Future efforts should focus on enhancing deduplication efficiency, scalability, and adaptability to handle increasingly large-scale and heterogeneous datasets encountered in modern storage systems.

- Investigating novel deduplication techniques, such as content-aware deduplication and machine learning-based deduplication, can offer promising avenues for improving deduplication effectiveness and reducing storage overhead.

Integration with Cloud and Edge Computing:

- As storage systems increasingly leverage cloud and edge computing paradigms, integrating deduplication mechanisms into cloud storage services and edge devices becomes essential.

- Future research should explore techniques for efficient deduplication across distributed storage environments, encompassing cloud data centers, edge nodes, and IoT devices, to minimize data redundancy and optimize storage utilization.

- Developing lightweight and adaptive deduplication algorithms tailored for edge computing environments can facilitate efficient data management, reduce network bandwidth consumption, and enhance overall system performance.

Addressing Security and Privacy Concerns:

- Security and privacy considerations are paramount in storage optimization mechanisms, particularly in deduplication, where data confidentiality and integrity are critical.

- Future research efforts should focus on enhancing the security and privacy aspects of deduplication algorithms to mitigate risks associated with data exposure, unauthorized access, and privacy breaches.

- Exploring cryptographic techniques, access control mechanisms, and privacy-preserving deduplication approaches can help bolster the security posture of deduplication systems and ensure compliance with regulatory requirements and privacy standards.

Adoption of Hybrid Deduplication Strategies:

- Hybrid deduplication approaches, combining inline, post-process, and source-based deduplication techniques, offer opportunities to optimize storage efficiency while minimizing performance overhead.

- Future implementations should consider adopting hybrid deduplication strategies tailored to specific use cases, workloads, and storage environments, leveraging the

strengths of each deduplication approach to achieve optimal storage optimization and performance benefits.

- Evaluating the trade-offs between deduplication overhead, resource utilization, and performance gains can inform the selection and configuration of hybrid deduplication strategies for diverse storage scenarios.

Standardization and Interoperability:

- Establishing standards and interoperability protocols for deduplication mechanisms can facilitate seamless integration, compatibility, and interoperability across heterogeneous storage platforms, systems, and vendors.

- Collaborative efforts involving industry consortia, standards bodies, and academia are essential to define common interfaces, protocols, and data formats for deduplication, enabling interoperable implementations and ecosystem-wide adoption.

- Promoting open-source initiatives and community-driven development models can foster innovation, collaboration, and knowledge sharing in the field of storage optimization, driving the evolution of deduplication technologies and practices.

Continuous Evaluation and Benchmarking:

- Continuous evaluation and benchmarking of deduplication mechanisms are crucial to monitor performance trends, identify bottlenecks, and assess the impact of algorithmic changes and optimizations.

- Establishing standardized benchmarking frameworks, datasets, and evaluation metrics can facilitate comparative analysis, reproducibility, and fair assessment of deduplication algorithms across different research studies and implementations.

- Encouraging transparency, sharing of experimental results, and peer-reviewed validation of deduplication techniques can foster trust, credibility, and rigor in the evaluation and validation process, advancing the state-of-the-art in storage optimization research.

Real-world Deployment and Validation:

- Validating deduplication mechanisms in real-world production environments is essential to assess their practicality, effectiveness, and suitability for deployment in mission-critical storage systems.

- Future research should emphasize real-world deployment studies, field trials, and case studies to evaluate the performance, reliability, and scalability of deduplication mechanisms in diverse enterprise, cloud, and edge computing environments.

- Collaborating with industry partners, cloud service providers, and data center operators can facilitate access to real-world datasets, infrastructure, and expertise, enabling comprehensive validation and validation of deduplication solutions in operational settings.

In conclusion, the aforementioned recommendations serve as guiding principles for advancing the state-of-the-art in storage optimization and deduplication techniques. By addressing key challenges, leveraging emerging technologies, and embracing collaborative research and development efforts, the storage community can drive innovation, efficiency, and sustainability in storage systems, paving the way for a data-driven future.

5.3 **Future Scope**

The future researchers may have the scope to discuss in detail about the network cost of the cloud storage system. They will have the opportunity to analyse the cost required to be paid by the users in order to move data from cloud storage systems to another location or the network. The future researchers will also have the scope to focus on the data backup factor of the cloud storage network system. They will have the opportunity to discuss the possible reasons for losing all the important data while operating in the cloud-based storage system. They can also find the best possible ways to allocate particular locations to particular information and data provided by the users.

The future researchers will have the scope to discuss how conscious duplication of any data can affect the cloud storage system. They can also discuss the implications of backup software in order to retain important data. They will have the opportunity to concentrate on the concept of data migration. They will have the possibility to explore the process of shifting from one cloud storage system to the by the users. It will help the researchers to discuss various new aspects of implementing cloud storage systems on the IOT environments.

REFERENCES

REFERENCES

[1] Intel, Developing solutions for Internet of Things, White paper on Internet of Things, www.intel.com/IoT.

[2] Junkin, V. Improving clinical reasoning skills by implementing the OPT model [dissertation]. Tuscaloosa: University of Alabama; 2018.

[3] Erturk, E., & Iles, H. R. E. (2015). Case study on cloud based library software as a service: Evaluating EZproxy. arXiv preprint arXiv:1511.07578.

[4] Vermesan, O., & Friess, P. (Eds.). (2013). Internet of things: converging technologies for smart environments and integrated ecosystems. River publishers.

[5] R. Vinoth and L. J. Deborah, "A survey on efficient storage and retrieval system for the implementation of data deduplication in cloud," SpringerLink, https://link.springer.com/chapter/10.1007/978-3-030-43192-1_95

[6] Vestergaard, R., Zhang, Q., & Lucani, D. E. (2019, December). Lossless compression of time series data with generalized deduplication. In 2019 IEEE Global Communications Conference (GLOBECOM) (pp. 1-6). IEEE.

[7] Vijayalakshmi, K., & Jayalakshmi, V. (2021, April). Analysis on data deduplication techniques of storage of big data in cloud. In 2021 5th International Conference on Computing Methodologies and Communication (ICCMC) (pp. 976-983). IEEE.

[8] Ellappan, M. (2021). Dynamic Prime Chunking Algorithm for Data Deduplication in Cloud Storage. KSII Transactions on Internet & Information Systems, 15(4).

[9] Anitha, P., Dhanushram, R., Sudhan, D. H., & Indhresh, T. R. S. (2021, May). Security Aware High Scalable paradigm for Data Deduplication in Big Data cloud computing Environments. In Journal of Physics: Conference Series (Vol. 1916, No. 1, p. 012097). IOP Publishing.

[10] Xu, Z., & Zhang, W. (2021, September). Quickcdc: A quick content defined chunking algorithm based on jumping and dynamically adjusting mask bits. In 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom) (pp. 288-299). IEEE.

[11] Kumar, N., Shobha, & Jain, S. C. (2019). Efficient data deduplication for big data storage systems. In Progress in Advanced Computing and Intelligent Engineering: Proceedings of ICACIE 2017, Volume 2 (pp. 351-371). Springer Singapore.

[12] Fan, Y., Lin, X., Liang, W., Tan, G., & Nanda, P. (2019). A secure privacy preserving deduplication scheme for cloud computing. Future Generation Computer Systems, 101, 127-135.

[13] H. A. Jasim and A. A. Fahad, New Techniques to Enhance Data Deduplication using Content based-TTTD Chunking Algorithm , International Journal of Advanced Computer Science and Applications, Vol. 9, No. 5, 2018

[14] Wu, H., Wang, C., Lu, K., Fu, Y., & Zhu, L. (2018, May). One size does not fit all: The case for chunking configuration in backup deduplication. In 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID) (pp. 213-222). IEEE.

[15] Oh, M., Park, S., Yoon, J., Kim, S., Lee, K. W., Weil, S., ... & Jung, M. (2018). Design of Global Data Deduplication for a Scale-Out Distributed Storage System. IEEE 38th International Conference on Distributed Computing Systems (ICDCS), 2–6 July 2018, 1063–1073.

[16] Xia, W., Zhou, Y., Jiang, H., Feng, D., Hua, Y., Hu, Y., ... & Zhang, Y. (2016). {FastCDC}: A fast and efficient {Content-Defined} chunking approach for data deduplication. In 2016 USENIX Annual Technical Conference (USENIX ATC 16) (pp. 101-114).

[17] Xu, X., & Tu, Q. (2015, September). Data deduplication mechanism for cloud storage systems. In 2015 international conference on cyber-enabled distributed computing and knowledge discovery (pp. 286-294). IEEE.

[18] Kirubakaran, R., Prathibhan, C. M., & Karthika, C. (2015, March). A cloud based model for deduplication of large data. In 2015 IEEE international conference on engineering and technology (ICETECH) (pp. 1-4). IEEE.

[19] Maruti, M. V., & Nighot, M. K. (2015, October). Authorized data Deduplication using hybrid cloud technique. In 2015 International Conference on Energy Systems and Applications (pp. 695-699). IEEE.

[20] Xu, X., Hu, N., & Tu, Q. (2016, October). Two-side data deduplication mechanism for non-center cloud storage systems. In 2016 IEEE International Conference on Ubiquitous Wireless Broadband (ICUWB) (pp. 1-4). IEEE.

[21] Zhang, Y., Jiang, H., Feng, D., Xia, W., Fu, M., Huang, F., & Zhou, Y. (2015, April). AE: An asymmetric extremum content defined chunking algorithm for fast

and bandwidth-efficient data deduplication. In 2015 IEEE Conference on Computer Communications (INFOCOM) (pp. 1337-1345). IEEE.

[22] Leesakul, W., Townend, P., & Xu, J. (2014, April). Dynamic data deduplication in cloud storage. In 2014 IEEE 8th International Symposium on Service Oriented System Engineering (pp. 320-325). IEEE.

[23] Krishnaprasad, P. K., & Narayamparambil, B. A. (2013, August). A proposal for improving data deduplication with dual side fixed size chunking algorithm. In 2013 Third International Conference on Advances in Computing and Communications (pp. 13-16). IEEE.

[24] Luo, S., & Hou, M. (2013, December). A novel chunk coalescing algorithm for data deduplication in cloud storage. In 2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT) (pp. 1-5). IEEE.

[25] Xia, W., Zhou, Y., Jiang, H., Feng, D., Hua, Y., Hu, Y., ... & Zhang, Y. (2016). {FastCDC}: A fast and efficient {Content-Defined} chunking approach for data deduplication. In 2016 USENIX Annual Technical Conference (USENIX ATC 16) (pp. 101-114).

[26] V. Balas C. Jain X. Zhao , Information Technology and Intelligent Transportation Systems , Volume 2, 2015

[27] Begum, M. J., & Haritha, B. (2020). Data Deduplication Strategies in Cloud Computing. International Journal of Innovative Science and Research Technology, 5(8), 734-738.

[28] Burramukku, Tirapathi & Ramya, U. & Sekhar, M.V.P.. (2016). A comparative study on data deduplication techniques in cloud storage. 8. 18521-18530.

[29] A. Venish and K. S. Sankar, "Study of chunking algorithm in data deduplication," in Proc. of International Conference on Soft Computing Systems, pp. 13-20, 2016.

[30] N. Bjorner, A. Blass, and Y. Gurevich, "Content-dependent chunking for differential compression, the local maximum approach," Journal of Computer and System Sciences, vol. 76, no. 3-4, pp. 154-203, 2010. https://doi.org/10.1016/j.jcss.2009.06.004

[31] M. Rabin, “Fingerprinting by random polynomials, no. tr-15-81,” Cambridge, MA, USA: Center for Research in Computing Techn., Aiken Computation Laboratory, Harvard Univ, pp. 15–18, 1981.

[32] R. Raju, M. Moh, and T. Moh, “Compression of wearable body sensor network data using improved two-threshold-two-divisor data chunking algorithms,” in 2018 International Conference on High Performance Computing Simulation (HPCS), July 2018, pp. 949–956.

[33] N. Bjørner, A. Blass, and Y. Gurevich, “Content-dependent chunking for differential compression, the local maximum approach,” J. Comput. Syst. Sci., vol. 76, no. 3-4, pp. 154–203, May 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jcss.2009.06.004

[34] Y. Zhang, D. Feng, H. Jiang, W. Xia, M. Fu, F. Huang, and Y. Zhou, “A fast asymmetric extremum content defined chunking algorithm for data deduplication in

backup storage systems," IEEE Transactions on Computers, vol. 66, no. 2, pp. 199–211, Feb 2017

[35] R. N. S. Widodo, H. Lim, and M. Atiquzzaman, "A new content-defined chunking algorithm for data deduplication in cloud storage," Future Generation Computer Systems, vol. 71, pp. 145–156, 2017

[36] Y. Tan and Z. Yan, "Multi-objective metrics to evaluate deduplication approaches," IEEE Access, vol. 5, pp. 5366–5377, 2017

[37] W. Tian, R. Li, Z. Xu, and W. Xiao, "Does the content defined chunking really solve the local boundary shift problem?" in 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC), Dec 2017, pp. 1–8

[38] C. Zhang, D. Qi, Z. Cai, W. Huang, X. Wang, W. Li, and J. Guo, "Mii: A novel content defined chunking algorithm for finding incremental data in data synchronization," IEEE Access, vol. 7, pp. 86 932–86 945, 2019.

[39] B. Chapuis, B. Garbinato, and P. Andritsos, "Throughput: A key performance measure of content-defined chunking algorithms," in 2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW), June 2016, pp. 7–12

[40] Habeeb, Ahmed. (2018). Introduction to Secure Hash Algorithms. 10.13140/RG.2.2.11090.25288.

[41] López, C. C., Crama, Y., Pironet, T., & Semet, F. (2024). Multi-period distribution networks with purchase commitment contracts. European Journal of Operational Research, 312(2), 556-572.

[42] Kumar, A., de Jesus Pacheco, D. A., Kaushik, K., & Rodrigues, J. J. P. C. (2022). Futuristic view of the internet of quantum drones: review, challenges and research agenda. Veh. Commun. 36, 100487 (2022).

[43] Guimarães, A., Aranha, D. F., & Borin, E. (2019). Optimized implementation of QC-MDPC code-based cryptography. Concurrency and Computation: Practice and Experience, 31(18), e5089.

[44] Drucker, N., Gueron, S., & Kostic, D. (2020, June). Fast polynomial inversion for post quantum QC-MDPC cryptography. In International Symposium on Cyber Security Cryptography and Machine Learning (pp. 110-127). Cham: Springer International Publishing.

[45] H. Guesmi and L. A. Saïdane, "Improved Data Storage Confidentiality in Cloud Computing Using Identity-Based Cryptography," 2017 25th International Conference on Systems Engineering (ICSEng), Las Vegas, NV, USA, 2017, pp. 324-330, doi: 10.1109/ICSEng.2017.32.

[46] Lee, H. N., Kim, Y. S., Singh, D., & Kaur, M. (2022). Green Bitcoin: Global Sound Money. arXiv preprint arXiv:2212.13986.

[47] Kumar, S., Banka, H., Kaushik, B., & Sharma, S. (2021). A review and analysis of secure and lightweight ECC-based RFID authentication protocol for Internet of Vehicles. Transactions on Emerging Telecommunications Technologies, 32(11), e4354.

[48] Thalapala, V. S., Mohan, A., & Guravaiah, K. (2022). Woaccpp: Wisdom of artificial crowds for controller placement problem with latency and reliability in sdn-wan.

[49] Rahimi, S., Jackson, R., Farahibozorg, S. R., & Hauk, O. (2023). Time-Lagged Multidimensional Pattern Connectivity (TL-MDPC): An EEG/MEG pattern transformation based functional connectivity metric. NeuroImage, 270, 119958.

[50] Jamali, S., Talebi, M. M., & Fotohi, R. (2021). Congestion control in high-speed networks using the probabilistic estimation approach. International Journal of Communication Systems, 34(7), e4766.

[51] Aravkin, A., Kumar, R., Mansour, H., Recht, B., & Herrmann, F. J. (2014). Fast methods for denoising matrix completion formulations, with applications to robust seismic data interpolation. SIAM Journal on Scientific Computing, 36(5), S237-S266.

[52] Jamali, S., Talebi, M. M., & Fotohi, R. (2021). Congestion control in high-speed networks using the probabilistic estimation approach. International Journal of Communication Systems, 34(7), e4766.

[53] Gill, S. S., Kumar, A., Singh, H., Singh, M., Kaur, K., Usman, M., & Buyya, R. (2022). Quantum computing: A taxonomy, systematic review and future directions. Software: Practice and Experience, 52(1), 66-114.

[54] Xie, H., Qin, Z., Li, G. Y., & Juang, B. H. (2021). Deep learning enabled semantic communication systems. IEEE Transactions on Signal Processing, 69, 2663-2675.

[55] Aragon, N., Gaborit, P., Hauteville, A., Ruatta, O., & Zémor, G. (2019). Low rank parity check codes: New decoding algorithms and applications to cryptography. IEEE Transactions on Information Theory, 65(12), 7697-7717.

[56] Ravi, P., Najm, Z., Bhasin, S., Khairallah, M., Gupta, S. S., & Chattopadhyay, A. (2019). Security is an architectural design constraint. Microprocessors and microsystems, 68, 17-27.

[57] Eshghi, K., & Tang, H. K. (2005). A framework for analyzing and improving content-based chunking algorithms. Hewlett-Packard Labs Technical Report TR, 30(2005).

[58] Zhang, Y., Jiang, H., Feng, D., Xia, W., Fu, M., Huang, F., & Zhou, Y. (2015, April). AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication. In 2015 IEEE Conference on Computer Communications (INFOCOM) (pp. 1337-1345). IEEE.

[59] N. A. Et al., "An enhanced approach to improve the security and performance for deduplication," Turkish Journal of Computer and Mathematics Education (TURCOMAT), vol. 12, no. 6, pp. 2866–2882, 2021. doi:10.17762/turcomat.v12i6.5797

[60] Vuong, H., Nguyen, H., & Tran, L. (2022). A Design of Parallel Content-Defined Chunking System Using Non-Hashing Algorithms on FPGA. IEEE Access, 10, 82036-82048.

[61] Saranya, R., Vidhya, S., Muthumari, M., & Sangeerthana, B. Data Deduplication in Cloud by Chunking.

[62] M. Mister, "10 advantages and disadvantages of cloud storage," Organize and Access Files From Anywhere, https://www.promax.com/blog/10-advantages-and-disadvantages-of-cloud-storage

[63] Viji, D., & Revathy, S. (2021). Comparative analysis for content defined chunking algorithms in data deduplication. Webology, 18(SpecialIssue2), 255-268.

[64] u-next.com, "Top 10 advantages and disadvantages of cloud storage: Unext," UNext, https://u-next.com/blogs/cloud-computing/top-10-advantages-and-disadvantages-of-cloud-storage/

[65] A. S. Gillis, "What is IOT (internet of things) and how does it work? - definition from techtarget.com," IoT Agenda, https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT

[66] Xia, W., Zou, X., Jiang, H., Zhou, Y., Liu, C., Feng, D., ... & Zhang, Y. (2020). The design of fast content-defined chunking for data deduplication based storage systems. IEEE Transactions on Parallel and Distributed Systems, 31(9), 2017-2031.
[67] Viji, D., & Revathy, S. (2021). Comparative analysis for content defined chunking algorithms in data deduplication. Webology, 18(SpecialIssue2), 255-268.

[68] Yoon, M. (2019). A constant-time chunking algorithm for packet-level deduplication. ICT Express, 5(2), 131-135.

[69] Vuong, H., Nguyen, H., & Tran, L. (2022). A Design of Parallel Content-Defined Chunking System Using Non-Hashing Algorithms on FPGA. IEEE Access, 10, 82036-82048.

[70] Jeong, Y. S., & Park, J. H. (2018). Advanced data processing, optimization & software engineering. Journal of Information Processing Systems, 14(5), 1063-1067.

[71] Jeong, Y. S., & Park, J. H. (2018). Advanced data processing, optimization & software engineering. Journal of Information Processing Systems, 14(5), 1063-1067.

[72] Saeed, A. S. M., & George, L. E. (2020). Data deduplication system based on content-defined chunking using bytes pair frequency occurrence. Symmetry, 12(11), 1841.

[73] www.zdnet.com, "What is the iot? everything you need to know about the internet of things right now," ZDNET, https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/

[74] Yash Arora • May 27th, I. S. Ganiyu, Y. Arora, and K. Tolety, "Data Segmentation in data mining: Strategy talks & more," Hevo, https://hevodata.com/learn/data-segmentation-in-data-mining/ .

[75] S. Hiter, "What is data segmentation?: Datamation: Security," Datamation, https://www.datamation.com/security/data-segmentation/ .

[76] Iliev, I., Sulikovska, I., Ivanova, E., Dimitrova, M., Nikolova, B., & Andreeva, C. (2022). Validation of a Light Source for Phototoxicity in in vitro Conditions. International Journal Bioautomation, 26(2), 141.

[77] C. S. N. Koushik, S. B. Choubey, A. Choubey, and G. R. Sinha, "Study of data deduplication for file chunking approaches," *Data Deduplication Approaches*, pp. 111–124, 2021. doi:10.1016/b978-0-12-823395-5.00008-2

[78] G.R. Sinha, Tin Thein Thwel, Samrudhi Mohdiwale, and Divya Prakash Shrivastava, "Data Deduplication Approaches: Concepts, Strategies, and

Challenges," in Data Deduplication Approaches, 2021, pp. 1-15. https://doi.org/10.1016/B978-0-12-823395-5.00019-7

[79] K. Vijayalakshmi and V. Jayalakshmi, "Analysis on data deduplication techniques of storage of big data in cloud," in International Conference.

[80] Srinivasan, Karthik, et al. "Secure multimedia data processing scheme in medical applications." Multimedia Tools and Applications (2022): 1-12.

[81] Kumari, Aparna, and Sudeep Tanwar. "A secure data analytics scheme for multimedia communication in a decentralized smart grid." Multimedia Tools and Applications 81.24 (2022): 34797-34822.

[82] Dhar, Shalini, Ashish Khare, and Rajani Singh. "Advanced security model for multimedia data sharing in Internet of Things." Transactions on Emerging Telecommunications Technologies 34.11 (2023): e4621.

[83] Sharma, Neha, Chinmay Chakraborty, and Rajeev Kumar. "Optimized multimedia data through computationally intelligent algorithms." Multimedia Systems 29.5 (2023): 2961-2977.