

Scalable Metadata Storage and Retrieval Techniques for Very Large Distributed Storage Systems

A Thesis

submitted in partial fulfilment of the requirements for the award of the degree of

Doctor of Philosophy

in

Computer Science and Engineering

Submitted by:

Harcharan Jit Singh
(Registration no: 951303001)

Under the guidance of
Dr. Seema Bawa
Professor, CSE and Dean (Outreach)



Thapar Institute of Engineering and Technology

Patiala-147004, Punjab, India

September 2023

Certificate

I, Harcharan Jit Singh, Registration no: 951303001, hereby certify that the work, which is being presented in the thesis, entitled **Scalable Metadata Storage And Retrieval Techniques for Very Large Distributed Storage Systems**, in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy submitted to the Computer Science and Engineering Department at Thapar Institute of Engineering and Technology, Patiala, Punjab, India is an authentic record of my own work carried out under the guidance of Dr. Seema Bawa. The matter presented in this thesis has not been submitted to any other institute for the award of any other degree.

Place: Patiala

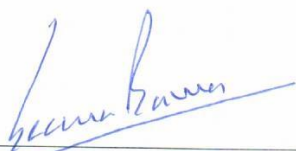
Date: 13-Sep-2023



Harcharan Jit Singh

Regn. No. 951303001

This is to certify that the statement made above by the candidate is correct to the best of my knowledge.



(Dr. Seema Bawa)

Professor and Dean (Outreach)
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
Patiala-147004
Punjab (India)

DEDICATED TO MY FATHER

Abstract

The rapid proliferation of global digital data has reached unprecedented levels, with a projected increase to a staggering 175 ZB by 2025, according to forecasts by the International Data Corporation (IDC). This exponential data growth presents immense challenges for storage and processing capabilities. Over the past three decades, parallel and distributed techniques have emerged as viable solutions to address the storage and retrieval demands of this ever-expanding data landscape.

As digital data surges, large-scale storage systems generate massive volumes of metadata. Metadata is a crucial component of storage and file systems that contains vital information about organizational structure, block identification numbers, physical block locations, and access permissions. Effectively managing metadata of exabyte or zettabyte storage capacities has become critical for parallel and distributed storage techniques. This research delves into the challenges posed by the continuous growth of global digital data. It explores the development of parallel and distributed metadata management techniques for efficient storage and retrieval. It investigates the significance of metadata in storage systems and underscores the importance of scalable distributed metadata management to handle these colossal data volumes.

The provisioning of an efficient ultra-large scalable distributed storage system for expanding cloud applications has been a challenging endeavour for both academia and industry. In such an ultra-large-scale storage system, data are distributed across multiple storage nodes to enhance performance, scalability, and availability. Access to this distributed data is facilitated through metadata, maintained by multiple metadata servers. The metadata carries crucial information about the physical address of data and access privileges. The efficiency of a storage system heavily relies on effective metadata management.

This research presents a comprehensive systematic literature analysis of metadata management techniques in storage systems. It aims to help researchers understand the significance of metadata management and the important parameters of metadata management techniques for storage systems. It systematically examines metadata management techniques developed by various industry and research groups, identifying various taxonomies based on different metadata distribution techniques.

Furthermore, this research investigates techniques based on distribution structures and critical parameters of metadata management. It presents a balanced view of the strengths and weaknesses of individual existing techniques, assisting researchers in selecting the

most appropriate approach for specific applications. Additionally, it addresses existing challenges and significant research directions in metadata management for the benefit of researchers.

This research proposes novel locality-aware metadata management techniques (LaMeta). The proposed work aims to enhance the performance of ultra-large distributed storage systems. The LaMeta leverages the locality of Metadata Servers (MDS) in a globally distributed ultra-large storage system to enhance metadata management performance. The novel LaMeta tailored for ultra-large distributed storage systems based static subtree, dynamic subtree, hash-based, and DROP-based metadata distribution techniques outperform than their baseline techniques. LaMeta-based techniques are designed to optimize metadata operations by efficiently utilizing the proximity of MDSs within the distributed storage system. It thoroughly evaluates the proposed techniques through extensive experiments and performance comparisons using real metadata datasets.

In summary, LaMeta ensures efficient metadata operation routing in distributed storage systems. The research presented in this thesis significantly contributes to the field of metadata management for ultra-large distributed storage systems. The novel LaMeta technique opens new avenues for optimizing metadata operations, improving storage performance, and meeting the demands of next-generation distributed storage technologies.

We evaluated the performance of LaMeta-based metadata distribution techniques against contemporary methods, measuring operations per second across varying scales of Metadata Servers (MDSs) from 5 to 50. Performance gains were assessed based on the aggregate throughput of metadata operations. The LaMeta-based static subtree metadata distribution techniques show gains from 2% to 5% at the 5 MDs scale, 9% to 12% at the 10-20 MDS scale and 12% to 14% at 25 to 50 than baseline static subtree partitioning. In LaMeta-based dynamic subtree partitioning, the average throughput gains range from 4% to 9% at 5 MDS, 10% to 20% at 10-30 MDSs scale and more than 20% at 40-50 MDS scale. While in LaMeta-based hash partitioning, the gains are 1% to 9% at 5-20 MDS scale, 9% to 19% at 25-40 MDS scale and more than 24% at 50 MDS scale.

The proposed locality-aware metadata management (LaMeta) for large distributed storage systems is effective for all such latency-sensitive data storage and retrieval applications such as High-Frequency Trading (HFT), Online Gaming, Content Delivery Networks (CDNs), Cloud Computing, IoT (Internet of Things), Autonomous Vehicles, etc.

Keywords: Distributed Computing, Big Data Storage Systems, Distributed file system, Information Storage, Information Retrieval, Data-intensive Computing, Metadata Man-

agement, Scalability, load balancing, Locality, Namespace, Deep learning, Stacked AutoEncoder, LaMeta, Machine Learning, High-performance computing, Storage and retrieval systems, Storage architectures, Cloud Storage, Recommendation System, Matrix factorization techniques.

Acknowledgements

This Ph.D. thesis is the outcome of a challenging journey, upon which Waheguruji has given me wisdom and perseverance to accomplish this work successfully. First and foremost, I am eternally grateful to Sri Guru Granth Sahib Ji for his enduring blessings and unwavering support throughout this journey.

I wish to extend my heartfelt gratitude to my esteemed supervisors, Dr. Seema Bawa, Professor and Dean (Outreach), for their invaluable guidance, enthusiastic encouragement, and valuable feedback throughout the course of this research endeavor. Her constant support, encouragement, and immense knowledge provided solace during my most challenging times. I am indebted to her insightful discussions and enlightening suggestions that helped me shape the direction of this research. The meticulous and diligent proofreading of this work contributed to the successful presentation of this thesis at the desired level of quality. Her constant pursuit of perfection, guidance, and valuable suggestions have guided me at every step of my professional journey. I sincerely thank her from the bottom of my heart and will be indebted to her throughout my lifetime.

I offer my sincere gratitude to our director Prof. Padmakumar Nair, Dr. N. Tejo Prakash, Dean (Research Sponsored Projects), and Dr. Shalini Batra, Professor and Head, Computer Science and Engineering Department for providing the necessary academic and administrative assistance in the completion of this work. I sincerely thank the members of my Doctoral committee – Dr. Inderveer Chana, Dr. Rinkle Rani, and Dr. Mahesh Kumar for their academic support and invaluable comments on this research. I am also thankful to Ph.D. Coordinator Dr. Sushma Jain and all the Computer Science and Engineering Department faculty members, Thapar Institute of Engineering and Technology, Patiala for their cooperation and support.

I would also like to thank Dr Maninder Singh Dean of Academic Affairs and Head of Centre of Information Technology for their continuous motivation and moral support. I especially thank all support staff of the Centre of Information Technology, Thapar Institute of Engineering and Technology for helping and supporting me during this period.

I would like to express my gratitude to the Thapar Institute of Engineering and Technology for providing an opportunity and resources for my doctoral research.

This thesis would not have been possible without the unconditional support of my family. I want to acknowledge my pillars of strength, my parents Late S. Jaspal Singh and Smt.

Nirmal Kaur for their blessings and eternal support. I owe a lot to my parents, brother sisters for their unwavering support and guidance throughout my personal and academic life. I gratefully acknowledge the love of my family who extended their cooperation and encouragement to me.

Immense gratitude to my wife Mrs. Ravneet Kaur for her understanding, tremendous moral support, and the help he extended to me all the time. She inspired me in all dimensions of life and instilled confidence in me to complete this journey successfully. I will always be indebted to her for his love and commitment. I want to express my sincere gratitude to my in-laws for their blessings for me. These acknowledgements will remain incomplete if I do not mention the names of my daughter Gunveen Kaur and son Ranfateh Singh.

Besides this, several other friends and relatives have helped me in the successful completion of this work. I wholeheartedly thank every person not mentioned above but who contributed to my thesis directly or indirectly.

Table of Contents

Title	Page No.
Abstract	v
Acknowledgements	ix
Table of Contents	xi
List of Figures	xv
List of Tables	xvii
Chapter 1 Introduction	1
1.1 Background and Related Work	3
1.2 Evolution of Distributed Metadata Management	3
1.3 Distributed Metadata Management	7
1.4 Merits of Distributed Metadata Management	8
1.5 Demerits of Distributed Metadata Management	10
1.6 Components of Distributed Metadata Management Technique	12
1.6.1 Metadata Partition Allocation Method	13
1.6.2 Object Storage Devices	16
1.6.3 Logical Structures for Distributed Metadata Management	17
1.7 Metadata Operations	20
1.8 Parameters of Distributed Metadata Management	21
1.8.1 Decoupling of Metadata Operations	21
1.8.2 Metadata Lookup Cost	21
1.8.3 Metadata Scalability	21
1.8.4 Metadata Locality	22
1.8.5 Metadata Load Balancing	23
1.8.6 Metadata Recovery	23
1.8.7 Metadata Replication	24
1.8.8 Metadata Migrations	24
1.9 Research Motivation	26
1.10 Thesis Objectives	27
1.11 Thesis Contributions	28

1.12	Thesis Organization	29
1.13	Conclusion	31
Chapter 2	Literature Review	33
2.1	Metadata Distribution Techniques: State of Art	34
2.2	Taxonomy of Metadata Distribution Techniques based on Distribution Structures	34
2.2.1	Table-based Metadata Distribution Techniques	34
2.2.2	Hash-based Distribution Techniques	36
2.2.3	Static Subtree Partitioning Distribution Techniques	38
2.2.4	Dynamic Subtree Distribution Techniques	39
2.2.5	DHT-Based Distribution Techniques	40
2.2.6	Bloom Filter Based Distribution Technique	42
2.2.7	Other Techniques	43
2.3	Strengths and Shortcomings of Techniques	47
2.4	Taxonomy of Distributed Structures and parameters	54
2.5	Parameters of Metadata Management Techniques	55
2.5.1	Taxonomy of Techniques based on Scalability	57
2.5.2	Taxonomy of Techniques based on Lookup	59
2.5.3	Taxonomy of Techniques based on Load Balancing	60
2.5.4	Taxonomy of Techniques based on Memory Overhead	62
2.5.5	Taxonomy of Techniques based on Recovery	63
2.5.6	Taxonomy of Techniques based on Replication	64
2.5.7	Taxonomy of Techniques based on Locality	65
2.5.8	Taxonomy of Techniques based on Migration	66
2.5.9	Taxonomy of Techniques based on Directory Operations	67
2.5.10	Decoupling of Metadata	68
2.6	Conclusion	69
Chapter 3	Proposed LaMeta: Locality-Aware Metadata Management Technique for an Ultra-Large Distributed Storage System	71
3.1	Locality of metadata on MDSs	73
3.2	Locality in Metadata Distribution Techniques	74
3.3	Proposed Metadata Locality-aware Technique	76
3.4	Constituents of Proposed LaMeta Technique	78
3.4.1	MDS Path measurement for locality	79
3.4.2	MDS path ratings	81
3.4.3	MDS Path Recommendation System	82

3.4.4	Singular value Decomposition (SVD) with Bias	83
3.4.5	SVD++	84
3.4.6	Regularised Non-Negative Matrix Factorization (RNMF)	85
3.4.7	Stacked AutoEncoder for MDS path rating	86
3.5	One-Hot Encoding	87
3.6	MDS K NextHop Cache and Metadata Operation Query Routing	87
3.7	Conclusion	88
Chapter 4	Design of the Proposed LaMeta Technique	89
4.1	MDS Path Probing and Path Ratings	91
4.2	MDS Path Ratings Matrix	91
4.3	Datasets and MDS Path Ratings	94
4.4	Stacked AutoEncoder	94
4.5	Locality Aware Metadata Routing	101
4.5.1	Locality Aware Next K MDS	102
4.6	Locality Expression	106
4.7	Conclusion	108
Chapter 5	Validation of the Proposed LaMeta-based Techniques	111
5.1	Validation of LaMeta-Based Technique	113
5.2	Performance of MF based recommendation techniques	114
5.3	Performance Evaluation of LaMeta Based Metadata Distribution Techniques	120
5.4	MDS Scale and Performance Comparison	121
5.4.1	LaMeta Based Static-Subtree Partitioning	122
5.4.2	LaMeta Based Dynamic Subtree Partitioning	123
5.4.3	LaMeta Based Hash Partitioning	123
5.4.4	LaMeta Based Dynamic Ring Online Partitioning (DROP)	128
5.5	Locality Comparison	131
5.6	Conclusion	137
Chapter 6	CONCLUSION	139
6.1	Contributions	139
6.2	Conclusions	141
6.3	Future Directions	142
Chapter A	Appendixes	145
A.1	Summary of Metadata Management Techniques	145
A.2	Abbreviations List	150

List of Publications	153
References	155

List of Figures

Figure No.	Title	Page No.
1.1	Clients accessing the metadata and data from MDS and data nodes, respectively	1
1.2	Timeline of metadata management techniques.	4
1.3	Taxonomy of Metadata Partitioning Techniques	14
1.4	Taxonomy of Metadata Management Parameters	22
2.1	Taxonomy of Metadata Techniques based on Distribution Structure. . . .	35
2.2	Taxonomy of Techniques based on Scalability.	58
2.3	Taxonomy of Techniques based on Lookup time	60
2.4	Taxonomy of Techniques based on Load Balancing.	61
2.5	Taxonomy of Techniques based on Memory Overhead.	62
2.6	Taxonomy of Techniques based on Recovery.	63
2.7	Taxonomy of Techniques based on Replication.	65
2.8	Taxonomy of Techniques based on Locality.	65
2.9	Taxonomy of Techniques based on Migration.	66
2.10	Taxonomy of Techniques based on Directory Operations.	68
3.1	Static subtree metadata partition on MDSs	74
3.2	Architecture of proposed Locality Aware Metadata Management (LaMeta)	78
3.3	Represents 8 numbers of MDSs distributed globally and a matrix of with no connections	79
3.4	MDS1 connecting MDS3, MDS4, MDS5, and MDS8 are represented in green color. MDS6 connecting MDS4, MDS5, MDS7, and MDS8 are rep- resented in yellow color.	80
3.5	The green entries represent known MDS paths, and the grey represents un- known MDS paths. The diagonal MDS path are self-paths and represents with light blue color.	80
4.1	The MDS path rating matrix of 4 source MDS and 4 destinations MDS .	92
4.2	Structure of Stacked AutoEncoder (SAE)	95
5.1	Throughput of LaMeta based static subtree metadata distribution tech- niques using LiveMap dataset.	124

5.2	Throughput of LaMeta based Dynamic subtree metadata distribution techniques using LiveMap dataset.	124
5.3	Throughput of LaMeta Hash based metadata distribution techniques using LiveMap dataset.	125
5.4	Throughput of LaMeta based DROP metadata distribution techniques using LiveMap dataset.	125
5.5	Throughput of LaMeta based Static Subtree metadata distribution techniques using DevTools dataset	126
5.6	Throughput of LaMeta based Dynamic Subtree metadata distribution techniques using DevTools dataset	126
5.7	Throughput of LaMeta Hash based metadata distribution techniques using DevTools dataset.	127
5.8	Throughput of LaMeta based DROP metadata distribution techniques using DevTools dataset.	127
5.9	Throughput of LaMeta based Static subtree metadata distribution techniques using MSN dataset.	128
5.10	Throughput of LaMeta based Dynamic subtree metadata distribution techniques using MSN dataset.	129
5.11	Throughput of LaMeta Hash based metadata distribution techniques using MSN dataset.	129
5.12	Throughput of LaMeta based DROP metadata distribution techniques using MSN dataset.	130
5.13	Average percentage increase in aggregate throughput of metadata distribution techniques using all three datasets.	131
5.14	The average locality of metadata operations using the MSN dataset at hop=2.	132
5.15	The average locality of metadata operations using the MSN dataset at hop=4.	133
5.16	The average locality of metadata operations using the MSN dataset at hop=6.	134
5.17	The average locality of metadata operations using the MSN dataset at hop=8.	135

List of Tables

Table No.	Title	Page No.
1.1	Taxonomy of Distributed structures.	19
1.2	Metadata Operations.	20
2.1	Taxonomy of Various Existing Techniques.	45
2.2	Strengths and shortcomings of metadata management techniques	53
2.3	Taxonomy of Distributed on Structures and their parameters	55
2.4	Relative Parameters of Metadata Management Techniques	57
2.5	Taxonomy Based on Decoupling of Metadata	69
3.1	Metadata Distribution Techniques and their performance parameters . .	76
3.2	List of Notations	83
4.1	Training set construction	93
4.2	Dataset for experiments	94
4.3	Rating Thresholds in RTT (ms)	94
4.4	Hyper-Parameters of the Stacked AutoEncoder	98
4.5	Path rating matrix with unknown ratings	99
4.6	Input data matrix of 8x8 matrix of MDSs	99
4.7	Training set construction	100
4.8	Output path rating matrix with predicted path ratings	101
4.9	Predicted path rating matrix with 0 diagonal rating value to avoid self- routing.	104
4.10	Locality Aware Next-Hop IDs for $k = 3$ and $r \geq 3$	105
4.11	Locality Aware Next-Hop IDs for $k = 2$ and $r \geq 4$	105
5.1	Hyper-parameter of SGD used in SVD, SVD++, RNMF	114
5.2	Confusion Matrix of SVD on DS-I	115
5.3	Confusion Matrix of SVD++ on DS-I	115
5.4	Confusion Matrix of RNMF on DS-I	115
5.5	Confusion Matrix of SAE on DS-I	115
5.6	True positive, false positive, false negative and true negative values for each binary class of confusion matrices on DS-I.	116

5.7	Accuracy, precision, recall and F1 score for each binary class of confusion matrices on DS-I.	117
5.8	Confusion Matrix of SVD on DS-II	118
5.9	Confusion Matrix of SVD++ on DS-II	118
5.10	Confusion Matrix of RNMF on DS-II	118
5.11	Confusion Matrix of SAE on DS-II	119
5.12	True positive, false positive, false negative and true negative values for each binary class of confusion matrices on DS-II.	119
5.13	Accuracy, precision, recall and F1 score for each binary class of confusion matrices on DS-II.	120
5.14	Data Sources used in Experiments	121
5.15	Comparison of LaMeta metrics <i>loc</i> for different methods at Hop = 2 . . .	132
5.16	Comparison of LaMeta metrics <i>loc</i> for different methods at Hop = 4 . . .	133
5.17	Comparison of LaMeta metrics <i>loc</i> for different methods at Hop = 6 . . .	134
5.18	Comparison of LaMeta metrics <i>loc</i> for different methods at Hop = 8 . . .	135

Chapter 1

Introduction

The proliferation of global digital data has reached unprecedented levels, with a projected increase from 33 zettabytes (ZB) in 2018 to a staggering 175 ZB by 2025, according to IDC forecasts [1]. This exponential data growth across diverse digital ecosystems presents significant challenges in terms of storage and processing capabilities [2]. Over the past three decades, parallel and distributed techniques have emerged as viable solutions to address this ever-expanding data landscape's storage and retrieval requirements [3–8]. As digital data continues to grow, large-scale storage systems generate vast amounts of metadata. Metadata, a critical component of storage and file systems, contains essential information about organizational structure, block identification numbers, physical block locations, and access permissions. To effectively manage the processing of metadata at exabyte or zettabyte storage capacities, scalable distributed metadata management has become an indispensable parallel and distributed technique [9].

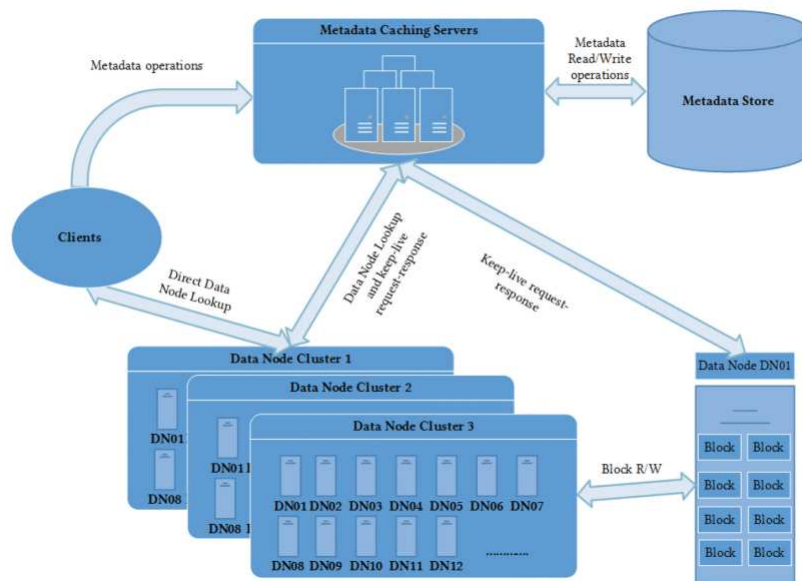


Figure 1.1: Clients accessing the metadata and data from MDS and data nodes, respectively

The performance of a large-scale file system is heavily reliant on its metadata management, encompassing tasks such as efficient storage and retrieval of metadata, decoupling metadata, and handling actual data traffic [8], [10–12]. Effective metadata management is crucial for achieving high data throughput and accommodating the workloads of millions of concurrent users. In such systems, client requests are directed to a metadata server (MDS) to access metadata of file and necessary permissions, following which the client can directly access the corresponding storage nodes to retrieve the file.

Although metadata is relatively very small compared to data, the performance of metadata operations plays a critical role in overall system throughput [10]. The memory of the Metadata Server (MDS) is used to cache metadata for fast lookup and achieve optimal performance of metadata operations. The alone MDS is unsustainable to accommodate the expanding demands of metadata caching and its operation.

This bottleneck necessitates metadata distribution across multiple metadata servers to achieve scalability, improved caching, and enhanced performance of metadata operations. Similarly, data operations are also constrained by I/O limitations, which can affect high-speed data streaming access. To overcome these challenges, parallel and distributed metadata and data management are essential to achieve scalability and high data streaming rates in a storage system. Figure 1.1 illustrates clients access distributed metadata through metadata caching servers, and multiple data nodes are utilized for data operations to achieve higher data streaming rates. The metadata distribution technique of a storage system aims to optimize metadata lookup efficiency, balance the metadata workload to prevent overloading specific nodes, minimize the cost of metadata migration, maintain high data locality to reduce latency, ensure high scalability to meet increasing demands, and ensure system availability to prevent failures

Cloud storage services are facing outages primarily due to issues with uneven load distribution, locality, and the lack of a scalable architecture for data and metadata storage and retrievals [13]. As a result, enterprises relying on these storage services are witnessing a decline in their customer satisfaction. The exponential growth in data storage demands calls for the development of a next-generation metadata management storage system that possesses seamless scalability to accommodate future growth, adaptability

to high workloads, resilience to failure, and ensures high data locality and consistent performance [14].

The researchers are proposing a distributed cloud storage allocation architecture for fair utilization of resources [15]. Cloud services are frequently dispersed across multiple data centres, necessitating the development of scalable strategies to enhance search efficiency and reduce the energy and cost associated with fulfilling requests. Multiagent-based systems have emerged as a potent methodology for enhancing distributed processing on large-scale storage systems [16].

The next section focuses on the evolution of distributed metadata management, its merits and demerits, components and parameters of distributed metadata management techniques in very large storage systems. Additionally, we delve into these areas in detail, providing a comprehensive understanding of the concepts involved.

1.1 Background and Related Work

Metadata of a file system contains crucial information about the data. In conventional file systems, metadata includes storage and access details, such as device name, device type, block size, inode number, objectid, last access, last modified, creation time, last backup time, node id, protection, and permission bits. Operations involving metadata encompass lookups, file/directory creation and deletion, and file opening and closing. To enhance metadata operation performance, memory is utilized to cache metadata. This section provides an overview of metadata storage and retrieval, traces the evolution of metadata management techniques, and discusses the advantages and disadvantages of distributed metadata management.

1.2 Evolution of Distributed Metadata Management

The metadata management has undergone significant evolution since the inception of file and storage systems, closely tied to the growth of the Internet and its services. The quest to enhance metadata management techniques has shifted from centralized approaches to

distributed metadata management for efficient caching and processing. This evolution remains an ongoing and continuous process, as depicted in Figure 1.2.

In distributed storage systems, the progress of metadata management techniques can be traced from static to dynamic metadata distribution. Initially, static distributions employed subtree-based and table-based partitioning techniques. Static subtree-based techniques were reported in the mid-1980s, coinciding with the coining of the term "Internet" and the development of the World Wide Web protocol by CERN. During this time, static subtree-based partitioning was introduced, involving child subtree distribution for metadata allocation. Although these techniques offered simplicity and scalability, they suffered from issues such as significant metadata migration and uneven distribution due to uncontrolled file system hierarchy growth and metadata operations like creations, moves, and renames. To address these challenges, table-based partitioning was introduced in the late 1990s. This period saw the emergence of online platforms like Yahoo, Google, and Ask, and managing the parent-child relationship in hierarchical file systems became a significant concern for distributing metadata management across multiple servers. The introduction of hash-based techniques proved instrumental in resolving the parent-child relationship issue in a distributed metadata management environment. Table-based partitioning techniques were also evolving during this time, providing scalability but with limitations due to uneven metadata distribution, memory overhead, and lack of flexibility.

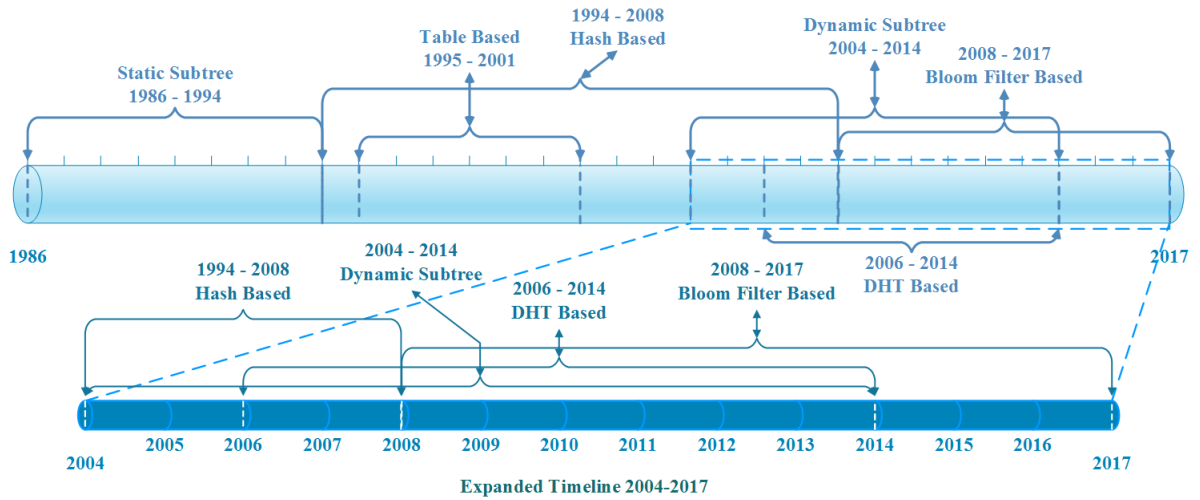


Figure 1.2: Timeline of metadata management techniques.

Between 2001 and 2010, the rapid growth of the Internet was marked by the introduction of prominent platforms such as Wikipedia, iTunes by Apple, LinkedIn, Facebook, Mozilla Firefox, Flickr, YouTube, Twitter, iPhone by Apple, and Android OS-based smartphones. This period brought about a realization of the significance of decoupling and the necessity for distributed metadata management in the realm of distributed storage systems. Consequently, hashing-based and dynamic subtree-based metadata partitioning and distribution techniques evolved.

Dynamic subtree partitioning emerged as a solution to uniformly distribute metadata and reduce metadata migration. The inherent self-healing and self-organizing characteristics were leveraged in distributed hash table (DHT)-based metadata partitioning, enabling an extensive MDS (metadata server) network for metadata management. However, even today, these approaches face challenges such as high metadata migration cost in DHT, the maintenance of DHT logical structure, and non-deterministic lookup.

To address some of these challenges, bloom-filter-based metadata management approaches incorporated bloom filter features to reduce lookup time. Nevertheless, issues like false positives, multicasting, and flooding of lookup queries continue to pose challenges in these techniques.

In conclusion, the evolution of distributed metadata management has been driven by the ever-growing landscape of the Internet and its services. From static to dynamic approaches, metadata management techniques have sought to meet the challenges posed by large-scale distributed storage systems and achieve more efficient caching and processing capabilities. In Chapter 2 of this thesis, we will discuss these techniques, analyzing their strengths and weaknesses to facilitate a comprehensive comparison.

Over the past decade, the demand for data storage has witnessed a significant surge, necessitating metadata management techniques that can scale in terms of storage while maintaining locality and consistent performance [17–20]. To address this challenge, ultra-large distributed storage techniques have been developed, leveraging features like decoupling metadata and data operations. These techniques aim to enhance performance, improve locality, increase scalability, and boost data throughput.

To achieve these objectives, files are divided into smaller units called blocks or chunks, typically ranging from 32 to 256MB. These blocks are distributed and replicated across various storage nodes known as data nodes. The distribution of data blocks effectively mitigates constraints imposed by disk I/O latency, resulting in higher data streaming rates, improved availability, and reduced network latency. Metadata is automatically generated for each data block or chunk during creation to facilitate future access and processing. Due to the extensive storage system, caching requires a substantial memory capacity. Hence, the voluminous metadata is partitioned, cached, and replicated among multiple metadata servers to optimize performance and availability. Each metadata server (MDS) is responsible for metadata caching and executing metadata operations on the designated partition [18], [20].

In the distributed storage system, data nodes store the actual data in the form of data blocks or chunks. These data nodes are responsible for sending status information of blocks and handling keep-alive requests to the designated MDS. To improve metadata lookup performance, data and metadata are placed closer to the user or access devices. The locality of data nodes, metadata servers, and metadata is crucial for the large distributed storage system, as it minimizes network congestion and enhances overall system throughput. In such a system, failures are not treated as exceptions but are anticipated as a normal part of the system's operation. Therefore, the system should be elastic enough to take auto-corrective measures when failures occur.

In this context, efficient metadata management is of utmost importance, as metadata operations typically account for over 60% of the workload [21], [22]. However, due to memory limitations, a single MDS cannot cache the entire metadata and may struggle to deliver consistent performance on metadata operations. As a result, several scalable file or storage systems have been proposed to address these challenges.

Indeed, efficient metadata management plays a critical role in the successful operation of a very large storage system. However, due to memory limitations, a single metadata server (MDS) is unable to cache the entire metadata, leading to potential challenges in delivering consistent performance for metadata operations.

In response to these considerations, numerous scalable file or storage systems have been proposed to overcome the limitations of a single MDS and enhance the overall efficiency of metadata management. These scalable systems employ innovative techniques to optimize metadata distribution, caching, and replication, thereby ensuring smooth and consistent performance for metadata-intensive operations.

By leveraging such scalable approaches, large storage systems can effectively handle the growing demands for data storage and achieve seamless metadata operations, contributing to the overall robustness and efficiency of the distributed storage environment.

The Google File System (GFS) is a highly scalable file system designed to run on commodity hardware [18]. It boasts essential properties like fault tolerance and high operational performance, making it an integral part of Google’s storage architecture. GFS handles vast amounts of data, with hundreds of terabytes stored on thousands of disks and nodes.

Another platform inspired by GFS is the Hadoop Distributed File System (HDFS), widely used for large-scale data computing and processing. HDFS utilizes parallel data processing to minimize disk I/O latency, resulting in high data streaming rates. It finds extensive application in log processing [23–25]. In HDFS, metadata is managed by interconnected namenode servers. The system employs a map-reduce programming paradigm to take advantage of distributed storage. The program is dynamically moved to the specific node where the data resides. The task is then executed on partial data present on the data node, and the results are later combined from all data nodes. Section 1.3 discusses the distributed metadata in detail.

1.3 Distributed Metadata Management

An ultra-large-scale file system, housing billions of objects or files/directories with their corresponding authorization parameters for access [26], faces significant challenges as it grows. As the file system expands, a single metadata server becomes inadequate to meet the performance requirements due to limitations in memory for caching and computational capabilities for handling metadata operation requests from users or ap-

plications [26], [27]. Consequently, a single metadata server struggles to process billions of user requests while maintaining the high data streaming rates necessary for the file system.

To address this issue, there arises a pressing need for the adoption of a distributed and decentralized approach to metadata caching and processing within the ultra-large file system, employing multiple interconnected Metadata Servers (MDSs) [28]. Numerous distributed metadata management schemes have been previously proposed for distributed file systems. These schemes or techniques can be broadly classified into table-based, hash-based, static and dynamic tree partitioning, and bloom-filter-based metadata management schemes [29]. A comprehensive discussion of these metadata management techniques will be presented in Chapter 2.

In this thesis, the terms "very large" and "ultra-large" are used interchangeably to describe storage systems encompassing vast amounts of data, ranging from hundreds of petabytes to exabytes [2], [26], [30]. While a typical storage system can accommodate data in the range of hundreds of gigabytes to a few terabytes, larger storage systems can handle hundreds of terabytes to a few petabytes of data. The efficient management of distributed metadata becomes of paramount importance in such ultra-large or very-large storage systems. This thesis first delves into the necessity of distributed metadata and its impact on the performance of large storage systems. It explores both the advantages and disadvantages associated with distributed metadata management, which will be discussed in detail in Section 1.4.

1.4 Merits of Distributed Metadata Management

This section highlights the significant benefits associated with utilizing distributed metadata in a large-scale storage system. The merits of distributed metadata are as follows:

- i *Scalability of Metadata Operations:* By distributing metadata across multiple Metadata Servers (MDSs) using metadata distribution techniques [2], [31–33], [30], the system can effectively harness additional memory, bandwidth, and CPU resources

from these MDSs. This scalability empowers the storage system to handle a higher volume of user requests efficiently. Utilizing multiple MDSs enhances the system’s memory capacity, enabling it to cache a larger amount of metadata and process a substantial number of metadata operations.

- ii *Availability of Metadata Operations:* The accessibility of metadata and data hinges upon the availability and proper functioning of multiple servers within the distributed metadata system. However, in the event of an MDS becoming unavailable or encountering failure, this accessibility is compromised. To ensure recovery and maintain high availability, the distributed metadata is replicated across multiple MDSs. Thus, the distribution of metadata serves the dual purpose of ensuring high system availability and mitigating the risk of a single point of failure (SPOF). Minimizing recovery time is crucial for a large distributed storage system, and the mean time to recovery of an MDS is influenced by the amount of metadata it caches [17–20]. During the MDS startup process, large metadata requires more time to cache compared to the smaller portion cached by multiple MDSs. Therefore, distributed metadata reduces the average mean time to recover a storage system.
- iii *Performance Enhancement of Metadata Operations:* The role of a single MDS (Metadata Server) involves monitoring data nodes’ heartbeat, caching metadata, and managing metadata operations. However, with an increased quantity of data nodes and data blocks, the performance of a single MDS is negatively impacted. To address this challenge, the distribution of metadata involves dividing the metadata load among multiple Metadata Distribution Servers (MDSs). This distributed approach generates high rates of metadata streaming and leads to significantly improved performance in handling metadata operations.
- iv *Locality of Metadata:* In a geographically distributed storage system, metadata is replicated locally. Distributed metadata offers a practical method to enhance performance by achieving better locality and reducing network latencies. The availability of locally replicated metadata reduces latency and alleviates the load on the MDS network [6], [34], [35]. As the MDS network typically comprises multiple network paths connecting to other MDSs, leveraging the improved locality in these MDSs

can further enhance overall system performance.

1.5 Demerits of Distributed Metadata Management

This section sheds light on the primary drawbacks associated with the use of distributed metadata in a large-scale storage system. The demerits of distributed metadata are as follows:

- i *Distributed Metadata Consistency:* Ensuring consistency in distributed metadata operations poses substantial complexity and requires significant resource consumption. In a large-scale distributed storage system, enforcing locks to maintain metadata consistency becomes a costly and resource-intensive endeavor. Additional CPU, memory, and network resources are essential to effectively manage and maintain the consistency of metadata operations.

To achieve consistent metadata across distributed servers, coordination and synchronization mechanisms are necessary, which can result in increased overhead and resource utilization. Balancing the need for consistency with the efficient utilization of resources presents a challenge in the design and implementation of distributed metadata management systems. As the scale of the storage system increases, the complexity of managing metadata consistency also escalates, necessitating careful consideration of resource allocation and coordination mechanisms to maintain the desired level of consistency without compromising overall system performance.

- ii *Unbalanced Metadata Distribution:* The uneven allocation of metadata can create hotspots, where certain Metadata Servers (MDSs) bear a disproportionate burden of metadata operations. This imbalance in distribution results in reduced performance and efficiency for the affected MDSs. As a consequence, the utilization of unbalanced metadata distribution techniques can lead to a deterioration in storage system performance and a decrease in overall system scalability.

To address this issue, achieving a balanced or uniform distribution of metadata becomes crucial. However, ensuring such balance requires additional resources and careful management to prevent performance degradation caused by uneven meta-

data distribution. Balancing the metadata allocation across MDSs is essential for optimizing metadata operations and overall system performance in a large-scale storage environment.

- iii *Metadata Migration:* To achieve a balanced workload distribution and minimize potential bottlenecks, metadata migration is employed to transfer metadata to additional Metadata Servers (MDSs). However, the process of recovery and replication becomes more intricate and challenging after the occurrence of metadata transfer. The migration of metadata directly impacts the overall performance of the storage system.

During metadata migration, careful planning and execution are required to ensure a seamless transfer and to avoid any adverse effects on the system's performance. The complexity of handling metadata migration lies in maintaining data integrity, handling concurrent access to the metadata, and ensuring continuous service availability during the migration process.

Efficient metadata migration strategies are essential to mitigate performance degradation and to optimize the use of available resources in large-scale storage systems. Properly managing metadata migration helps in maintaining a balanced workload distribution, minimizing bottlenecks, and ultimately ensuring a robust and efficient storage system.

- iv *Complexities and Compliances of Distributed Metadata:* Storing metadata in a distributed environment introduces various additional concerns, such as distributed caching, distributed lookup, availability, locality, migration cost, recovery, local regulatory compliance, security, and adherence to local privacy regulations. The decentralized nature of metadata distribution further complicates these issues. Additionally, different nations have distinct data regulatory compliances, especially concerning citizen data. Managing data regulatory compliances across different nations poses significant challenges for geographically distributed storage systems [14], [17], [20], [28], [29], [36].

The distribution of metadata necessitates the use of distributed structures, each of

which has its unique set of characteristics, strengths, and limitations. Balancing these characteristics and addressing their individual complexities are critical aspects of designing and operating an efficient and compliant distributed metadata management system. Addressing the complexities and compliances of distributed metadata requires careful consideration of data sovereignty, privacy laws, data residency requirements, and cross-border data transfers. Implementing robust mechanisms for distributed caching, lookup, and recovery, while ensuring data availability and security, becomes essential for the successful operation of large-scale geographically distributed storage systems.

The incorporation of distributed metadata in storage systems is of utmost importance due to its capacity to enable access to large memory caches across multiple metadata servers (MDS), thereby alleviating disk input/output (I/O) limitations. It ensures data locality, facilitates the scalability of metadata operations, and enhances the overall system availability, albeit at an additional cost. These capabilities are critical for the development of next-generation storage systems at this current time. In next Section, we delve into the metadata partitioning methods, logical structures, and parameters of a metadata management technique, providing a comprehensive discussion on this essential aspect of distributed storage systems.

1.6 Components of Distributed Metadata Management Technique

The evolution of distributed metadata management techniques has given rise to a distributed metadata management system, playing a crucial role in achieving a scalable storage system. This section emphasizes three key aspects: metadata partitioning methods, logical structure employed in metadata distribution on Metadata Servers (MDSs), and the characteristics of distributed metadata management techniques.

Metadata Partitioning Methods: To achieve scalability, metadata is divided into partitions and allocated to multiple MDSs. The choice of partitioning methods is essential for efficient distributed metadata management. Different partitioning strategies, such as

hash-based, tree-based, or table-based, have their unique merits and trade-offs. Selecting an appropriate partitioning method influences the distribution of metadata across MDSs, which impacts overall system performance.

Logical Structure for Metadata Distribution: The logical structure employed in metadata distribution across MDSs is vital in achieving balanced workload distribution and efficient metadata operations. Careful consideration of the logical organization of metadata is necessary to avoid hotspots and bottlenecks. An optimized logical structure ensures a smooth flow of metadata operations, enhancing system efficiency.

Characteristics of Distributed Metadata Management Techniques: Distributed metadata management techniques possess specific characteristics, such as fault tolerance, high availability, distributed caching, data locality, and data replication. These features are instrumental in ensuring seamless metadata access, efficient data storage, and robust system performance.

Additionally, this section delves into the discussion of a special type of object storage devices. Understanding their characteristics and functionalities is crucial as they can significantly impact distributed metadata management efficiency.

The distribution approaches, logical structures, and key performance parameters of distributed metadata management techniques are explored in detail within this section. A comprehensive understanding of these components is essential for designing and implementing efficient and scalable distributed metadata management systems.

1.6.1 Metadata Partition Allocation Method

In the distributed storage system, achieving scalability involves dividing and allocating metadata to multiple Metadata Servers (MDSs). The metadata partitioning method or technique employed exhibits varying levels of complexity and the ability to allocate metadata uniformly, statically, and dynamically. The partition allocation is crucial in large distributed storage systems, as it enables efficient caching, storage, and maintenance by the MDSs. There are primarily three types of techniques used for partition allocation to metadata servers (MDSs) as depicted in Figure 1.3.

- i *Manual Allocation:* Manual allocation is a metadata partition allocation method where the administrator manually assigns metadata to specific Metadata Servers (MDSs) using a static table. In this approach, the distribution of partitioned metadata management is defined in advance, and the allocation remains unchanged unless modified manually by administrators. This methodology is characterized by its inflexibility, as any changes in the allocation require direct intervention from administrators.

Manual allocation requires a significant allocation of human resources as administrators are responsible for defining and maintaining the metadata distribution. The process involves careful planning and decision-making to ensure an efficient allocation strategy. The effectiveness of metadata performance in a distributed storage system heavily depends on the decisions made by administrators during the manual allocation process. If the initial distribution is not well-optimized, it may result in uneven workload distribution and suboptimal system performance.

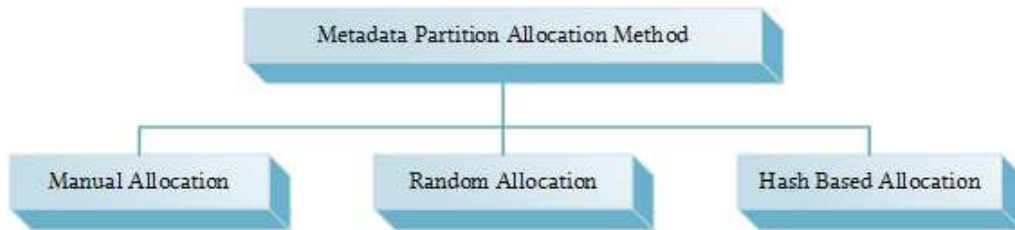


Figure 1.3: Taxonomy of Metadata Partitioning Techniques

- ii *Random Allocation:* This approach involves the utilisation of a randomised algorithm to automatically and randomly assign the partitioned metadata to the Metadata Server (MDS). The random allocation method evenly distributes metadata partitions. At the same time, the employed methods lack sensitivity towards preserving locality, managing metadata server access loads, and having excessive lookup latency. Techniques such as xFS [31], [37], HBA [32], Dynamic Dir-grain Policy (DDG) [5], G-HBA [2], [30], etc. rely on the use of random metadata partition allocation.

The random allocation method ensures that each Metadata Server (MDS) receives an approximately equal share of the partitioned metadata. This even distribution

aims to prevent the creation of hotspots and promotes load balancing across the MDSs. While random allocation achieves a balanced workload distribution, it lacks sensitivity towards preserving data locality and efficiently managing the access loads on different MDSs. As a result, certain metadata operations may experience higher lookup latencies or less efficient access to relevant data.

However, due to its limitations in preserving locality and load management, random allocation may not be the most efficient choice for all distributed storage environments. Other techniques like dynamic partitioning or hybrid approaches that combine random allocation with additional considerations may be more suitable to optimize metadata performance and overall system efficiency. The selection of the partition allocation method should align with the specific requirements and characteristics of the distributed storage system.

- iii *Hashing Based Allocation:* Hashing-based allocation is a metadata partitioning method that relies on system-defined hashing algorithms to establish mappings between metadata partitions and the Metadata Servers (MDSs). In this approach, metadata partitions are distributed among MDSs based on the hash values generated by the hashing algorithms.

The hashing-based approach demonstrates effectiveness in terms of lookup efficiency and memory usage. Hashing enables quick and direct retrieval of metadata partitions based on their hash values, leading to faster metadata operations. While hashing-based allocation offers lookup efficiency, it may face challenges in preserving the child-parent hierarchical relationship within the metadata structure. Operations involving renaming directories or modifying access permissions and ownership of directories can be time-consuming due to the lack of direct hierarchical mappings.

Several studies, such as LazyHybrid [6], [38], [39], [40], [26], HBA [32], Group-based Hash-Based Allocation (G-HBA) [2], [30], have employed the hashing-based allocation approach. These techniques leverage hashing algorithms to achieve faster metadata lookup and better memory utilization.

However, the trade-off for improved efficiency through hashing is the potential loss

of hierarchical relationships, which can result in slower operations when dealing with directory modifications. For certain distributed storage systems, where preserving the hierarchical relationship is critical, other partitioning methods that prioritize hierarchy preservation, such as tree-based partitioning, may be more suitable.

In conclusion, the choice of metadata partitioning method, should be based on the specific requirements and priorities of the storage system, such as the need for efficient lookup, memory usage, and hierarchical relationship preservation.

1.6.2 Object Storage Devices

Object storage is a technological framework that manages data in the form of discrete objects, and Object Storage Devices (OSDs) are a specialized form of storage designed to address input/output (I/O) operation delays effectively. OSDs separate metadata and data operations, enabling more efficient and flexible distribution of metadata [18], [41–47]. This separation allows clients to communicate with a metadata server for metadata operations, while direct access to OSDs is used for read and write operations, minimizing I/O constraints. Key features and benefits of OSDs include:

Variable Block Sizes: OSDs support storage blocks with variable sizes, and low-level storage devices determine the appropriate block size. This flexibility allows for better optimization of storage resources and efficient data management.

Minimized I/O Delays: By separating metadata and data operations and implementing parallel execution of operations, OSDs reduce I/O delays and enhance overall system performance, achieving significant scalability.

Equitable Workload Allocation: OSDs facilitate the equitable allocation of metadata workloads across the storage system, ensuring efficient utilization of resources and maintaining performance consistency.

Inherent System Dynamism: The architectural design of large-scale distributed storage systems, incorporating OSDs, recognizes that node failures, whether MDS or Data Nodes, are inherent to the system. This inherent dynamism allows the system to adapt to changing workloads and effectively handle metadata operations over time.

Optimal Performance: The system’s architectural design leverages available MDS resources to meet various workload requirements, resulting in optimal performance even as workloads grow incrementally.

In summary, Object Storage Devices (OSDs) play a crucial role in modern storage systems by enhancing performance through minimized I/O delays, flexible block sizes, equitable workload allocation, and inherent system dynamism. By separating metadata and data operations, OSDs contribute to achieving efficient and scalable distributed storage solutions.

1.6.3 Logical Structures for Distributed Metadata Management

The performance and features of distributed metadata management, including storage retrieval and query routing between Metadata Servers (MDSs), heavily rely on the logical structures used for interconnection. Below are brief descriptions of commonly employed logical structures for the distribution of metadata:

1. Mapping-Table Structure: In this table-based logical structure, a shared table contains information about the range of metadata keys. Each Metadata Server (MDS) is assigned object identifiers for metadata within a specific range. Regardless of the physical location or workload of a given key, the responsible MDS manages and maps it. The mapping-table structure ensures efficient partitioning and distribution of metadata.
2. Tree-Based Structure: The tree-based distributed metadata management system utilizes a parent-child relationship for partitioning. The root and its children are assigned to different Metadata Servers (MDSs). There are two classifications within this structure: static sub-tree and dynamic sub-tree.
 - (a) Static Sub-Tree: Metadata is partitioned into distinct sub-trees without overlaps, each associated with a specific MDS. The challenge here is to avoid an imbalanced distribution of metadata, where some MDSs may be overloaded while others remain underutilized. Effective load balancing and partitioning expertise are essential for implementing the static sub-tree structure.

- (b) Dynamic Sub-Tree: This approach uses hashing techniques to divide metadata within directories located near the root directory. In scenarios where an MDS experiences a heavy workload, specific entries are dynamically migrated to alternative MDS instances with lighter workloads. Although dynamic sub-tree offers adaptability to changes in workload, metadata migration adds complexity to the recovery process.
3. Peer-to-Peer (P2P) overlay structures are decentralized architectures with inherent capabilities for self-healing, self-configuration, self-optimization, and self-protection [48–52]. Free riding is a major problem in peer-to-peer networks, and reputation management systems have evolved to overcome such issues [53–55]. Distributed Hash Tables (DHTs) take advantage of these features, providing scalability, failover capabilities, and minimal administrative and maintenance needs. P2P overlay structures are employed in multi-domain systems (MDSs) to configure interconnection networks. They enable infrastructure sharing for purposes like metadata query routing and location sharing among peers.

Several networks have been developed to address these needs, including structured P2P overlay networks like Content Addressable Network (CAN) [56], Chord [56], Tapestry [57], Pastry [58], Kademlia [59], Viceroy [60], and Gnutella [61, 62]. Unstructured P2P overlay networks like Gnutella, KaZaA, BitTorrent, and Overnet use non-deterministic search mechanisms for resource discovery [61–63].

Table 1.1 provides an overview of these logical structures, along with their descriptions, advantages, and limitations. In the structure utilizing bloom filters, hashing is employed to establish metadata mapping on Metadata Servers (MDS).

The structure that relies on bloom filters uses hashing to establish metadata mapping on Metadata Servers (MDS). This approach is well-known for its high efficiency in querying the membership of an element within the MDS (Metadata Server). Bloom filters are widely used for their space efficiency and ability to provide quick query responses.

4. Bloom filters are data structures that represent sets of elements with the help of a

bit array and a series of hash functions. When an element is added to the bloom filter, its hash values are used to set the corresponding bits in the array. To check the membership of an element, the same hash functions are applied, and if all the corresponding bits are set, the element is considered to be present in the set. However, due to the nature of bloom filters, false positives can occur, meaning that an element may be reported as a member even if it is not in the set. False negatives, on the other hand, never occur.

Despite the possibility of false positives, bloom filters are widely used in distributed systems, particularly for scenarios where memory is limited, and fast query responses are crucial. They are particularly useful in cases where the occasional false positive can be tolerated, such as cache lookups and approximate membership queries. Bloom filters offer an excellent trade-off between memory efficiency and query performance, making them a popular choice for certain types of distributed metadata management systems and other applications with similar requirements.

Logical Structures	Descriptions	Strengths	Weaknesses
Table	A central table is used to distribute metadata on MDS.	Simple, achieve limited scalability, static, no metadata migration required	Hotspot due to client operation request, uneven metadata operation load, inflexible, imbalanced, memory overhead, large lookup time
Tree	Childs of tree or subtree used to distribute metadata on MDS.	Simple, keeps parent-child relationship, achieves limited scalability	Load balancing, large migration cost
Peer-to-Peer	Peer to peer protocols are used to distribute metadata on MDS.	Scalability, self-healing, and self-organizing, no memory overhead, high scalability	Large migration cost, logical structure maintenance, non-deterministic lookup
Bloom Filter Array	A probabilistic model and randomized data structure for metadata caching and distribution.	Space and time efficient for lookup queries, highly scalable	False positive ratio, small migration cost

Table 1.1: Taxonomy of Distributed structures.

These logical structures play a crucial role in the efficient distribution and management of metadata across multiple Metadata Servers (MDSs). The choice of a suitable structure depends on factors such as workload distribution, scalability requirements, and the system's ability to handle metadata operations effectively. The P2P overlay structures provide diverse solutions for metadata management and query routing, making them valuable in large-scale storage systems and distributed environments. Each structure has unique strengths and limitations, and their implementation depends on specific system requirements and goals.

1.7 Metadata Operations

Metadata operations play a crucial role in storage systems, especially in highly scalable distributed storage systems. The efficiency of metadata lookup operations directly impacts the overall storage system's performance. The retrieval or lookup operation of metadata is widely used within storage systems. Table 1.2 presents metadata operations. Efficiently handling these metadata operations is vital to ensure smooth and high-performance data access and management within the storage system.

Metadata Operations	Descriptions
Create	At the time of creation of object or file/directory, a corresponding metadata of that object or file/directory is created having information like name, parent directory, owner, group, access permissions, physical block locations of data, etc.
Lookup or Search	Lookup or search of metadata of a file system object for access.
Rename	Renaming of an object or file/directory of metadata
Delete	Delete metadata entry of object or file/directory
Close	Close file/directory operation
Changing permission	Changing access permission, owner, group, etc
Stat	Create, access and modification date time

Table 1.2: Metadata Operations.

1.8 Parameters of Distributed Metadata Management

The large-scale distributed storage system implements metadata distribution to achieve better scalability and optimize the performance of metadata operations. Additionally, caching mechanisms are utilized to further enhance the efficiency of these operations. It is crucial to decouple metadata operations from data operations in order to ensure the efficient functioning of a very large storage system [5]. This separation leads to improved performance, scalability, and enhanced data throughput. Alongside the decoupling, there have been discussions regarding the objectives and challenges of growing distributed storage systems on a significant scale. In this section, we have identified and discussed the key metadata parameters. Figure 1.4 presents a taxonomy based on these metadata management parameters.

1.8.1 Decoupling of Metadata Operations

The execution of metadata operations is conducted in an autonomous manner and exhibits little dependence on data operations that are constrained by slow I/O bound disk operations. Hence, the decoupling of metadata operations and data operations yields enhanced performance, scalability, and overall data throughput as metadata is cached for fast metadata operations.

1.8.2 Metadata Lookup Cost

The retrieval or lookup querying metadata operation pertaining to an object is the most important operation in a very large distributed storage system. Numerous distribution logical structures have been proposed to optimise metadata indexing and enhance the performance of metadata lookup queries. These structures are accompanied by various indexing and caching techniques.

1.8.3 Metadata Scalability

The ability to scale storage is a crucial attribute for future storage systems due to the exponential growth of data. It refers to the system's ability to efficiently manage and pro-

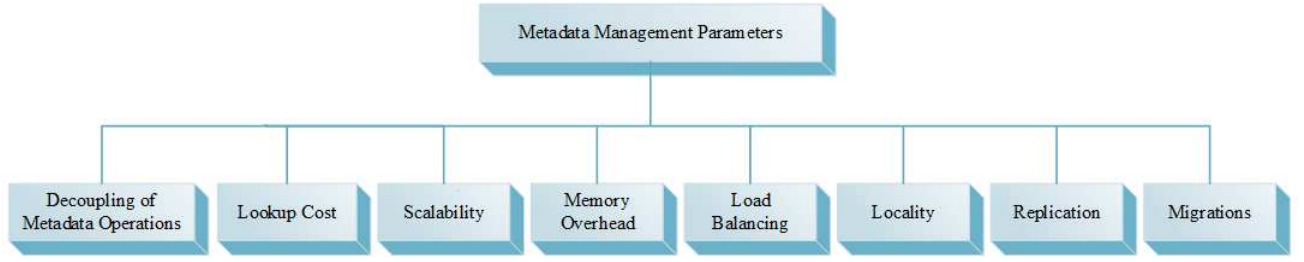


Figure 1.4: Taxonomy of Metadata Management Parameters

cess a significant amount of metadata and data. Scalability is essential to ensure that user metadata operations can be executed at a high rate while maintaining consistent performance levels. The storage system’s inherent capability allows it to handle an increasing volume of metadata and associated operations without compromising performance. The distribution of metadata is a key approach to achieve this enhanced scalability.

1.8.3.1 Memory Overhead

Memory overhead in a very large distributed storage system is associated with the indexing and management of metadata. Metadata is indexed and cached to facilitate real-time lookup and retrieval, thereby reducing excessive latencies caused by disk I/O operations. However, managing extensive metadata within limited memory resources poses a significant challenge in metadata management techniques. Ensuring efficient memory utilization is crucial for achieving optimal performance in large-scale storage systems.

1.8.4 Metadata Locality

Metadata locality refers to the optimization of metadata operations by utilizing local caching and placing metadata in nearby locations. This strategy effectively reduces the negative impact of excessive network routing latency on metadata operation performance. By performing metadata operations locally or in nearby locations, the system can minimize network bandwidth usage and improve the overall throughput of the distributed storage system. The emphasis on metadata locality is essential to achieve efficient and responsive metadata management in large-scale distributed storage systems.

1.8.5 Metadata Load Balancing

Metadata load balancing is a critical aspect of efficient metadata management in large-scale distributed storage systems. Due to the significant amount of metadata caching and its operational demands, it becomes essential to distribute metadata among multiple Metadata Servers (MDSs) in a balanced manner. The distribution of metadata and its operations should be even and adaptable to dynamic changes in workload to fully leverage the advantages of multi-server metadata management.

Load balancing in metadata management aims to ensure that the workload is evenly distributed across the Metadata Servers (MDSs) and adapts to changes in metadata and its operational behavior over time. By achieving load balancing, the system can effectively prevent bottlenecks and hotspots, ensuring consistent and optimal performance for metadata operations. A well-balanced distribution of metadata workload among MDSs is crucial for maintaining a high level of efficiency, responsiveness, and overall system scalability in large distributed storage environments.

1.8.6 Metadata Recovery

Metadata recovery is a crucial aspect of ensuring the reliability and fault tolerance of a distributed storage system. In a large-scale distributed system, failures are considered inherent rather than exceptional circumstances. Therefore, the primary objective of an extensive file system is to reliably store and cache metadata information while being able to recover from failures at the level of Metadata Servers (MDS) and storage nodes. To achieve effective recovery, a storage system must have a comprehensive strategy in place for handling failures, ranging from individual metadata server failures to failures in metadata operations. This involves implementing backup procedures to ensure critical metadata operations can be recovered in case of failures.

To support recovery, the metadata is duplicated and spread across multiple geographically distributed Metadata Servers (MDSs). By having multiple replicas of metadata, the system can ensure redundancy and fault tolerance, allowing it to continue functioning even in the event of a failure. However, the recovery feature adds complexity to the

system, particularly with regards to maintaining consistency and managing the additional load of replicas. Therefore, efficient techniques are necessary to manage the failover of metadata operations and provide a high level of reliability and availability to the storage system. Thus, efficient techniques are required to cater the failover of metadata operations and provide high reliability to the storage system.

1.8.7 Metadata Replication

Metadata replication is an essential protective measure in a distributed storage system aimed at mitigating the potential impact of failures within the Metadata Servers (MDS) or the interconnecting structure. In any distributed system, failures or disconnections are inevitable. By replicating metadata, the system ensures that multiple copies of critical metadata are available across different MDSs, allowing the system to continue executing user requests for metadata operations even in the presence of failures.

The strategic placement of metadata replicas is crucial to ensure redundancy and fault tolerance. By distributing replicas across various MDSs, the system can safeguard against the loss of metadata due to failures in specific servers. Additionally, ensuring the uniformity of replicas is vital to maintain data consistency and coherence throughout the system. Managing the replication factor is another critical aspect of metadata replication. The replication factor refers to the number of copies or replicas maintained for each piece of metadata. A well-designed replication strategy takes into account factors such as system reliability, performance, and resource constraints to determine the appropriate replication factor.

Overall, metadata replication plays a vital role in enhancing the reliability, fault tolerance, and availability of a large distributed storage system, enabling it to continue functioning seamlessly even in the presence of failures [4].

1.8.8 Metadata Migrations

Metadata migrations are essential for carrying out metadata operations involving parent-child relationships. Operations such as renaming and moving directories require the

migration of metadata across Metadata Servers (MDSs) to ensure data integrity and consistency.

The migration of metadata is an ongoing and automatic process that occurs between multiple MDSs. By transferring metadata among MDSs, the system aims to achieve an even distribution of the workload and prevent the occurrence of hotspots where specific MDSs experience excessive load. However, the presence of network latency and the complexity of the migration process can impact the overall performance of storage systems. Metadata migration introduces additional intricacies to recovery and failover procedures, making system maintenance and management more challenging.

In an ideal scenario, a large-scale storage system should aim to minimize or eliminate the need for metadata migration. By implementing efficient partitioning and load balancing strategies, the system can achieve better performance and reduce the need for frequent metadata migration, leading to enhanced scalability and optimal functioning.

The field of metadata management presents several complex challenges that researchers need to address to achieve optimal performance and scalability in large-scale distributed storage systems. While various techniques and approaches have been proposed to enhance metadata distribution, caching, locality, scalability, and other key parameters, a complete and ideal solution is yet to be discovered.

The pursuit of a comprehensive metadata management technique that effectively balances scalability, performance, metadata operations efficiency, and fault tolerance remains an ongoing challenge. Researchers must continue to explore innovative approaches, algorithms, and architectural designs to overcome the limitations and trade-offs associated with current metadata management techniques. Collaborative efforts among researchers from various domains, such as distributed systems, data management, and storage technologies, will be crucial in pushing the boundaries of metadata management and addressing the complexities posed by ultra-large-scale storage systems.

The main tradeoff between locality and consistency in the caching metadata revolves around balancing the benefits of quick, local access with the need to maintain accurate and up-to-date metadata. Storage systems that require strong consistency may experience

higher latency and overhead due to the need for frequent synchronization and coordination between caches and the central source of metadata. Storage systems that can tolerate eventual consistency might allow caches to have stale metadata for a while, benefiting from lower latency and reduced overhead at the cost of having potentially outdated information temporarily. Some systems implement hybrid approaches, where metadata is cached locally and updated periodically or based on specific events, aiming to balance the benefits of low-latency access and acceptable levels of consistency. Tight cache coherence needs immediate cache consistency on its backup distributed cache of metadata. In large distributed metadata, the cache is stale for a while to improve the performance. Generally, these storage systems are not real-time transaction systems. The tradeoff between locality and consistency in locally cached metadata is a balance between the performance gains from reduced latency and the resource overhead associated with maintaining consistency across distributed caches.

1.9 Research Motivation

After conducting a comprehensive survey on metadata storage and retrieval schemes, it has been observed that metadata storage and retrieval for next-generation storage systems is a promising field of research. This research area is expected to play a significant role in developing distributed and scalable storage systems. The main focus resides on the scalability of the storage system, which will ultimately result in establishing an ultra-large-scale storage system. It has been identified that the main challenge is related to metadata distribution on multiple servers. To enhance the scalability, performance, locality, and high availability of the storage system, it is recommended to distribute the metadata across multiple Metadata Servers (MDS).

The issue relates to the trade-off between achieving metadata distribution and maintaining the locality of metadata has been identified. The balanced metadata distribution ensures an even workload distribution across multiple metadata servers (MDS). Simultaneously, it is crucial to maintain the locality of metadata to provide local access to metadata. Geographic-based metadata distribution schemes or local redundancy are employed to guard against failures. The presence of an imbalanced distribution of metadata

can lead to the formation of hotspots in the metadata storage system, thereby impacting the performance of the storage infrastructure, as it heavily relies on these overloaded metadata servers. Most scholarly investigations have prioritised the optimisation and balanced distribution of metadata, often overlooking metadata locality.

The problems analyzed above necessitate devising an efficient locality-aware metadata management technique for an ultra-large distributed storage system. Such a distributed system will improve the locality and performance of baseline metadata distribution metadata operation throughput and locality of metadata operations.

It is noteworthy that while many research efforts have focused on optimizing metadata distribution and workload balancing, the aspect of metadata locality has not received as much attention. Overlooking metadata locality can lead to performance issues and overloaded metadata servers, ultimately affecting the overall storage infrastructure's efficiency.

The problems analyzed above necessitate devising an efficient locality-aware metadata management technique for ultra-large distributed storage systems. Such a distributed system will improve the locality and performance of baseline metadata distribution metadata operation throughput and locality of metadata operations.

1.10 Thesis Objectives

The objectives of this research work are:

1. To study, analyze and explore the latest metadata storage and retrieval schemes in a very large-scale storage system.
2. To propose scalable metadata storage and retrieval techniques for large storage systems.
3. To implement the proposed techniques for scalable metadata storage and retrieval in a simulated environment.
4. To validate the implemented techniques by performing comparative analysis.

1.11 Thesis Contributions

1. A comprehensive survey of various metadata management techniques was conducted, exploring the need for distributed metadata management and analyzing its advantages and disadvantages. The study covered different distributed metadata management schemes, providing detailed insights into their significance.
2. Contemporary metadata management techniques were compared and classified based on taxonomies, including static subtree-based, table-based, hash-based, dynamic subtree-based, distributed hash table-based, and bloom filter-based techniques. Key performance parameters like metadata lookup time, load balancing, scalability, memory overhead, metadata migration, recovery, locality, and replication were analyzed and discussed.
3. The research gaps and future directions in the field of distributed metadata management were identified and presented based on an exhaustive survey of existing techniques.
4. A novel metadata locality-aware technique (LaMeta) was proposed, leveraging MDS locality in a globally distributed ultra-large cloud storage system to enhance metadata management performance. LaMeta-based static subtree, dynamic subtree, hash-based, and DROP-based metadata distribution techniques for ultra-large distributed storage systems were introduced.
5. The performance of collaborative filtering-based MF techniques (SVD, SVD++, RNMF) and Stacked AutoEncoder was evaluated the predicting path for metadata query routing in distributed storage systems. LaMeta based on Stacked AutoEncoder (SAE) structure was proposed for efficient and intelligent MDS locality prediction in metadata distribution techniques.
6. A locality-aware next k MDS algorithm for metadata query routing was proposed, defining the metadata operation's locality to compute optimal metadata operation routing. The MDSs utilize recommended top k MDS with optimum path ratings for the next hop in the metadata distribution structure.

7. We evaluated the performance of LaMeta-based metadata distribution techniques against contemporary methods, measuring operations per second across varying scales of Metadata Servers (MDSs) from 5 to 50. Performance gains were assessed based on the aggregate throughput of metadata operations. The LaMeta-based static subtree metadata distribution techniques show gains from 2% to 5% at the 5 MDS scale, 9% to 12% at the 10-20 MDS scale and 12% to 14% at 25 to 50 than baseline static subtree partitioning. In LaMeta-based dynamic subtree partitioning, the average throughput gains range from 4% to 9% at 5 MDS, 10% to 20% at 10-30 MDSs scale and more than 20% at 40-50 MDS scale. While in LaMeta-based hash partitioning, the gains are 1% to 9% at 5-20 MDS scale, 9% to 19% at 25-40 MDS scale and more than 24% at 50 MDS scale.
8. Extensive experiments and performance evaluations of LaMeta-based static subtree, dynamic subtree, hash-based, and DROP-based metadata distribution techniques were conducted using real metadata datasets. The results demonstrated that the proposed LaMeta-based techniques significantly improved the locality and performance of baseline metadata distribution in terms of aggregate throughput and metadata operation locality.

1.12 Thesis Organization

This thesis is organized into six chapters as follows:

Chapter 1: Introduction

In this chapter, the background of metadata management is presented, with a specific focus on the necessity of distributed metadata management for very large distributed storage systems. The main emphasis is on discussing various distributed metadata management techniques for ultra-large storage systems, their evaluation, merits, and demerits. Additionally, the chapter explores the logical structures used in distributed metadata management and delves into the performance parameters associated with this field. Based on the gaps identified in the existing literature, the problem formulation and objectives of the thesis are also described in this chapter. Finally, the structure of the thesis is

outlined, including its organization and the different contributions made throughout the research work.

Chapter 2: Literature Review

This chapter provides a detailed survey of existing metadata management techniques employed in storage systems. It compares contemporary metadata management techniques and classifies them based on various distribution techniques and key performance parameters. The existing metadata distributed techniques are categorized as static subtree-based, table-based, hash-based, dynamic subtree-based, distributed hash table-based, and bloom filter-based. The key performance parameters such as lookup time, load balancing, scalability, memory overhead, metadata migration, recovery, locality, and replication in distributed metadata management are thoroughly discussed. Moreover, the chapter highlights the strengths and weaknesses of contemporary metadata management techniques in storage systems. the chapter comprehensively investigates the distributed metadata management techniques, and their significance is explained in detail.

Chapter 3: Proposed LaMeta: Locality-Aware Metadata Management Technique for an Ultra-Large Distributed Storage System

This chapter addresses the problem of locality of metadata servers in metadata distribution techniques, aiming to improve metadata query routing and enhance the performance of the distributed storage system. It introduces the architecture of the proposed locality-aware metadata management technique (LaMeta). Further, the chapter describes the MDS path measurement for locality and the need for metadata server locality for performance enhancement. It discusses collaborative filtering-based MF techniques, including Singular Vector Decomposition (SVD), SVD++, Regularized Non-Negative Matrix Factorization (RNMF), and CNN-based Stacked AutoEncoder for predicting path ratings.

Chapter 4: Design of the Proposed LaMeta-based Techniques

This section provides insights into the design and implementation aspects of the proposed

LaMeta-based technique. It explains the deep neural network structure of Stacked AutoEncoder used to predict the MDSs' path ratings. The chapter presents the structure of Stacked AutoEncoder (SAE) employed and the algorithm of the intelligent Locality-aware metadata routing technique. Additionally, a locality function is proposed to measure the relative locality of metadata operations.

Chapter 5: Validation of the Proposed LaMeta-based Techniques

The chapter begins by describing the experimental environment, scientific packages and datasets used for the implementation and performance comparison of our proposed LaMeta-based techniques. It discusses the performance evaluation of collaborative filtering-based MF techniques (SVD, SVD++, RNMF) and CNN-based Stacked AutoEncoder for predicting the path using three datasets. The results of extensive experiments and performance evaluations conducted on LaMeta-based static subtree, dynamic subtree, hash-based, and DROP-based metadata distribution techniques using three real metadata datasets are then presented. The proposed LaMeta-based techniques have demonstrated improvements in locality and performance compared to baseline metadata distribution in terms of aggregate throughput and locality of metadata operations.

Chapter 6: Conclusions and Future Scope

The final chapter summarizes the research work presented in the thesis, highlighting the achievements in locality-aware metadata distribution. It outlines several directions for future research on scalable distributed storage systems to inspire researchers to further explore and develop innovative solutions for managing metadata in next-generation storage systems.

1.13 Conclusion

In conclusion, this chapter has provided a comprehensive overview of metadata management for large-scale storage systems. Through an in-depth exploration of the concepts, we have gained valuable insights into the challenges and opportunities in the field of

metadata management for large storage systems.

The existing literature and research analysis has highlighted the shortcomings of current scalable distributed metadata management techniques. Inadequate metadata load distribution, data locality issues, and the lack of scalable architectures have resulted in outages and reduced customer satisfaction. Developing a next-generation metadata management storage system is imperative to address these challenges and accommodate the exponential growth in data storage demands. Such a system should exhibit seamless scalability, high workloads adaptability, failure resilience, data locality, and consistent performance. Additionally, the chapter outlines the problem formulation and objectives of the thesis. Finally, the structure of the thesis is outlined, including its organization and the different contributions made throughout the research work.

Moving forward, the next chapter presents a detailed survey of the existing metadata management techniques used in storage systems. It categorizes these techniques based on various distribution methods and critical performance parameters. The discussion includes an examination of static subtree-based, table-based, hash-based, dynamic subtree-based, distributed hash table-based, and bloom filter-based metadata distributed techniques. Crucial performance parameters such as lookup time, load balancing, scalability, memory overhead, metadata migration, recovery, locality, and replication in distributed metadata management are thoroughly explored.

Furthermore, the strengths and weaknesses of contemporary metadata management techniques in storage systems are analyzed. By conducting a comprehensive investigation of distributed metadata management techniques, we can better understand their significance and identify areas for improvement in the proposed research work.

Chapter 2

Literature Review

The previous chapter provided a comprehensive overview of the evolution of metadata management and its significance in distributed storage systems. It highlighted the importance of distributed metadata management and its various categorizations, which offer insights into the techniques used in large distributed storage systems. The previous chapter also established the research motivation and objectives, concluding with a summary of the research contributions and the organization of the thesis.

This chapter aims to conduct a comprehensive review of the existing literature, examining the current state of distributed metadata management techniques in large and scalable storage systems. The existing research on distributed metadata management is categorized based on the distribution structure and various key performance metrics.

Section 2.1 of this chapter outlines the current scenario of metadata distribution techniques, setting the stage for further exploration. Subsequently, Section 2.2 presents a taxonomy of metadata distribution techniques based on the distribution structure, providing a systematic classification. In Section 2.3, the strengths and shortcomings of metadata distribution techniques are thoroughly discussed, shedding light on their respective advantages and limitations. Section 2.4 delves into the taxonomy of distributed structures and their associated parameters, offering a deeper understanding of their characteristics. Section 2.5 further analyzes and compares metadata management techniques based on various parameters, providing valuable insights into their performance and capabilities. In conclusion, this chapter serves as a comprehensive guide to the current landscape of distributed metadata management techniques. It highlights the advancements and challenges faced in the field, paving the way for subsequent research and analysis.

2.1 Metadata Distribution Techniques: State of Art

In this chapter, we conduct a comprehensive survey of metadata management techniques used in storage systems. We delve into a detailed analysis, categorizing these techniques based on distribution structures and discussing their key performance parameters. The review covers various categories, such as static subtree-based, table-based, hash-based, dynamic subtree-based, distributed hash table-based, and bloom filter-based methods. We thoroughly examine critical performance metrics like lookup time, load balancing, scalability, memory overhead, metadata migration, recovery, locality, and replication. Additionally, we provide an in-depth analysis of the strengths and weaknesses of the current metadata management techniques employed in storage systems.

2.2 Taxonomy of Metadata Distribution Techniques based on Distribution Structures

The distribution structures used in metadata management of storage systems have evolved over time to overcome scalability limitations and improve overall performance. Each structure offers unique advantages and disadvantages. Various metadata attributes, including file or directory sizes, file type and content organization, namespace usage, and differences among file systems influence the relationship between metadata caching and distributed techniques [64]. These attributes are broadly classified into six taxonomies: table-based, hash-based, static subtree-based, dynamic subtree-based, DHT-based, and bloom-filter based. The figure below illustrates these taxonomies and are presented in Figure 2.1. These attributes impact the efficiency of metadata retrieval and storage.

2.2.1 Table-based Metadata Distribution Techniques

In metadata distribution for large-scale storage systems, a central table known as a "mapping table" plays a crucial role. This table contains information about the metadata key ranges and the corresponding Metadata Servers (MDSs) identifiers. This central table is shared amongst the MDSs. Each MDS organizes and caches the assigned pre-defined

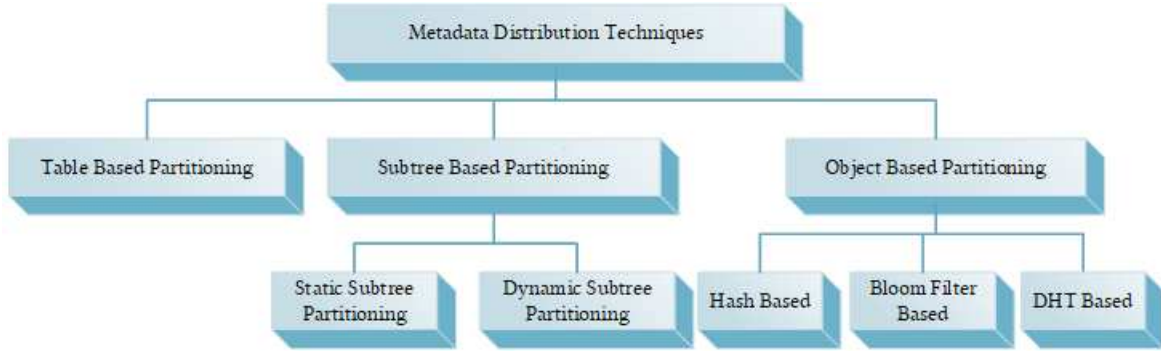


Figure 2.1: Taxonomy of Metadata Techniques based on Distribution Structure.

fixed set of metadata keys. The metadata keys falling in the specified range are processed by MDS regardless of locality and existing load. In the key-lookup operation, the key partitioned table is referred to reach the corresponding MDS.

Several table-based metadata mapping schemes, such as xFS [37] and zFS [41], have been proposed. In xFS, a manager map table is shared and replicated among clients, containing information about the manager node and groups of index numbers of files. This table defines the mapping of index number groups to manager nodes. B+ trees are used for local indexing on individual manager nodes. These schemes do not require metadata migration, providing them with a failover capability. However, in ultra-large distributed file systems, the maintenance of a centralized mapping table can lead to a degradation in overall system performance. Additionally, table-based schemes have a high memory overhead and limited scalability.

Facebook utilizes an optimized Haystack storage system for photos, running on top of extent-based xFS [37]. The High Performance, Reliable, MRAM-Enabled Storage (HeRMES) leverages magnetic RAM (MRAM) to enhance metadata request efficiency [65]. Non-volatile MRAM is utilized in HeRMES to improve file system data streaming and eliminate disk I/O for metadata accesses.

A decentralized zFS proposed on commodity hardware is interconnected with a high-speed network and uses a cooperative cache of all participating machines, a distributed transactions server, and direct access to OSD (Object Storage Device) disks [66,67], [41]. A lease manager is responsible for managing the lock of transactions to ensure data integrity on OSDs. In zFS, FMGR tables provide access to files and directories for

clients.

These table-based metadata distribution techniques present diverse approaches to managing metadata in large-scale storage systems, each with its unique advantages and challenges. Researchers continue to explore optimization strategies to enhance performance, scalability, and data integrity in ultra-large distributed file systems.

2.2.2 Hash-based Distribution Techniques

The hash-based techniques exploit hashing techniques to distribute metadata objects among multiple Metadata Servers (MDSs) in large-scale storage systems. One of the main advantages of hashing is load balancing, which ensures an even distribution of metadata among multiple servers. Several file systems, such as Lustre [68], RAMA [69], and Vesta [70, 71], utilize hash-based mapping for metadata distribution and lookup operations. Hashing provides efficient lookup operations and has low memory overheads.

One specific approach, pathname hashing, is proposed in Lazy Hybrid (LH) for hierarchical directory management [6, 38–40]. However, this scheme requires metadata migration and incurs overheads. Additionally, maintaining the parent-child relationship in LH is costly and requires rebuilding the metadata mapping and association with MDS during rename operations. While hashing eliminates memory overheads, it introduces additional overheads in terms of metadata migration, reconstruction, and rebalancing.

AbFS is another technique presented for a high-performance metadata management file system, utilizing various structures for organizing and caching metadata. It replicates volume and delegation tables among nodes and partitions access tables among MDSs. Each MDS maintains a volume table mapping volumes and metadata servers. First, the distribution among volumes is hashed using a hash function. Then, these hash keys are distributed on MDS with the help of a hash table. AbFS effectively reduces disk I/O by caching all metadata on each MDS and optimizing client queries. Each MDS maintains a volume table mapping on metadata servers. The proposed file system design demonstrates performance in organizing files and directories.

Lustre, a parallel open-source file system, employs multiple MDSs connected to Meta-

data Targets (MDTs) for storing and processing metadata operations [68]. The authors propose an active storage strategy for Lustre to optimize split writing and hierarchical directory striping [72]. They also introduce the concept of Object Storage Device (OSD+) to handle data and metadata operations on improved OSD devices [73, 74].

Dynamic hashing and network-commit protocols are utilized in Fusion Parallel File System (FPFS) design to distribute directory objects among MDSs uniformly. Dynamic hashing [45, 75] is used to distribute directory objects among MDS uniformly, and a network-commit protocol is used for coordination among OSDs. The performance of FPFS is compared with Lustre, demonstrating improved metadata server throughput and scalability [68].

Hyper Virtual File System (HVFS) is proposed for PetaFLOPS Supercomputer, utilizing distributed metadata management to enhance processing capacity and memory caching size [76]. HVFS stores data on a couple of system file systems. The distributed metadata management is exploited to improve the processing capacity, memory caching size and I/O size of MDS. The HVFS has a two-level directory partitioning. The first level partition divides each directory and distributes each partition on servers. The second level partition controls the partition chunk size. The results show that average read and write throughput is improved by adding storage servers.

Another approach, Directory-Hashing (DH), exploits pathname hashing for efficient metadata distribution [77]. DH introduces a Directory storage unit Index Management Server (DIMS) to handle client requests and utilizes pipelining and pre-fetch technology to overcome delayed metadata accesses. Therefore, metadata accesses are delayed; pipelining and pre-fetch technology are used to overcome this effect. DH algorithm improved the scalability of renaming directory operations, changing permission and churning of MDS from the cluster. Similarly, the directory-level-based hash (DLBH) architecture is proposed for metadata management, showing fast listing, building, and rebuilding of hierarchical directories on metadata changes [78].

These hash-based metadata distribution techniques present various strategies for efficiently managing metadata in large-scale storage systems, each offering its own benefits

and performance improvements. Researchers continue to explore and refine these techniques to enhance scalability, locality, performance, and consistency in distributed file systems.

2.2.3 Static Subtree Partitioning Distribution Techniques

Static subtree partitioning techniques involve dividing metadata into non-overlapping subtrees and then mapping them to respective Metadata Servers (MDS). However, this approach comes with its own challenges, primarily related to an imbalance in metadata distribution. As a result, some servers may be heavily loaded while others remain underutilized. Prominent file systems like NFS [79], AFS [80], Coda [81, 82], and Sprite [83] have adopted subtree partitioning techniques.

One notable example is the Storage Resource Broker (SRB), which aims to provide reliable storage and efficient query handling for large files distributed across geographically diverse locations [84]. The scheme uses geographic partitioning, where both storage and metadata are divided into zones, each equipped with its own Metadata Catalog (MCAT). While this approach ensures fast directory operations and eliminates the need for metadata migration, it inherently faces issues due to the uneven distribution of metadata load. The static tree partitioning scheme requires the knowledge and experience of an administrator for partitioning and balancing metadata and its operations.

Another proposal, the Metadata Replication File System (MRFS), focuses on hierarchical efficiency in distributed metadata management [85]. The architecture involves multiple MDSs and an additional namespace server (NS). Each MDS stores non-overlapping metadata and handles metadata creation operations. The NS maintains a global and consistent directory tree in the main memory, achieving high scalability and low latency for metadata operations. To avoid hotspot issues and flash crowds, MRFS employs metadata replication among MDSs.

Gfarm, a Grid file system designed for large data sharing and computing, utilizes file-system federation on computing nodes to achieve high data streaming and mitigate hotspots [86]. The proposed data locality processing further enhances the streaming rate.

In the context of subtree partitioning, a Replica-Based Distributed Metadata Management (RBMMS) introduces a spares graph to map replica structures and their associations, ensuring high efficiency and availability [87]. Another approach, the Global Distributed Storage System (GDSS), adopts subtree partitioning to facilitate efficient information management and transfer of information [88]. The proposed Metadata Controller (MDC) based on MatchTable enhances metadata lookup and reduces the load on the root directory server.

A Replica-Based Distributed Metadata Management (RBMMS) proposes subtree partitioning [87]. RBMMS maintains a spares graph for mapping the replica structure and its replica association to achieve high efficiency and availability. Another approach called Global Distributed Storage System (GDSS) proposes subtree partitioning for efficient management and transfer of information [88]. A Metadata Controller (MDC) based on MatchTable is proposed to achieve scalability. The approach improves metadata lookup and reduces the load on the root directory server. Andrew [80] and Sprite [83] also exploit static subtree partitioning techniques for metadata distribution.

In summary, static subtree partitioning techniques offer valuable solutions for distributing metadata in large-scale storage systems. However, the challenge lies in maintaining a balanced distribution of metadata to ensure optimal performance and efficiency. As researchers continue to explore and refine these techniques, the goal is to address limitations and enhance the scalability and effectiveness of metadata distribution in distributed file systems.

2.2.4 Dynamic Subtree Distribution Techniques

Dynamic subtree-based distribution techniques employ a hashing mechanism to divide metadata, focusing on directories near the root directory [6]. The goal is to alleviate the excessive load on Metadata Servers (MDS) caused by metadata access requests. To achieve this, metadata is dynamically migrated to the most suitable MDS with negligible load. CEPH [44], for instance, dynamically distributes subtrees of the directory hierarchy to multiple MDSs based on the current workload. Highly used directories are hashed across multiple MDSs to achieve load balancing.

The Dynamic Dir-Grain policy (DDG) introduces a technique for balanced metadata distribution. DDG utilizes dynamic subtree partitioning, ensuring a more efficient metadata allocation [5]a. However, one drawback of dynamic subtree-based partitioning is the need for rebuilding and reallocating metadata when MDSs join or leave the system. This redistribution process introduces extra overhead.

MilkyWay-2 supercomputer presents a scalable and high-data streaming rate solution with a user-level virtualized file system collaborating with local and shared storage [89]. This system achieves improved performance and scalability by optimizing the data streaming rate in data-intensive applications. The hybrid storage devices, SSDs and disk arrays, are strategically layered by I/O Nodes (ION) and storage servers to achieve high I/O burst performance. The storage system shows better performance and scalability than Luster and H2FS Local file systems.

Other approaches, such as FARSITE [90], Kosha [35], GIGA+ [26], and OBFS [35], also leverage dynamic subtree distribution techniques for efficient metadata management.

As researchers continue to explore dynamic subtree-based distribution techniques, the aim is to enhance load balancing, scalability, and overall efficiency in metadata management. By addressing the challenges and limitations associated with dynamic metadata migration, these techniques hold the potential to further optimize distributed file systems and improve their performance in handling large-scale data processing tasks.

2.2.5 DHT-Based Distribution Techniques

DHT-Based distribution techniques have been proposed to address the challenges of efficient metadata storage and retrieval in large-scale distributed storage systems. Grannary [91] introduces a sharing-oriented storage system that utilizes a distributed hash table (DHT) approach for storing metadata. Their work proposes efficient storage and recovery strategy. Their adaptive DHT-based mapping algorithm ensures availability and consistency while a locality-aware and popularity-aware replication mechanism improves retrieval performance and reduces network traffic.

Cassandra is another example of a DHT-based storage system used for managing a vast amount of structured data [92]. Facebook uses Cassandra, a distributed storage to store semi-structured data. It leverages consistent hashing to distribute metadata load across multiple commodity machines, providing high availability and fault tolerance against single points of failure. This distributed data provides high read and write throughput.

DCache [93], on the other hand, is a metadata management scheme that exploits client cache along with distributed MDS using an improved chord (IChord) DHT structure. This approach achieves scalability and high performance for metadata storage and processing.

The DROP [26] approach uses dynamic ring online partitioning and locality-preserving hashing to distribute metadata load among MDSs, ensuring namespace locality is maintained. It employs Histogram-based Dynamic Load Balancing (HDLB) for load balancing and replicates metadata for failover protection. It replicates metadata on the next two available nodes for failover. This approach requires metadata migration for balancing and avoiding hotspots in metadata operations.

Other works, such as Ring File System (RFS) and P2P combined indexing, have also demonstrated the potential of DHT-based techniques in achieving scalability, fault tolerance, and improved throughput in data-intensive applications [94].

In P2P combined indexing [28], a combined index metadata architecture has been proposed. This architecture aims to enhance the extensibility and interoperability of metadata by classifying it accordingly. The design takes into account the possibility of metadata servers leaving and joining the system due to failures, ensuring the system's resilience. The peer-to-peer techniques discussed in related works [48, 49, 56–63, 95] are based on the leave/join operation of P2P protocols, contributing to the adaptability and scalability of the system.

In CFS [96], a novel approach is employed where each server is assigned virtual nodes based on their size, enabling efficient load balancing. Another innovative solution, the Peer-to-Peer Metadata Service (PPMS) [97], combines a hierarchy structure with peer-to-peer principles. In this architecture, the Metadata Index Server (MIS) serves as the first

layer, responsible for managing the metadata associated with directories. The MDSs, forming the lower layer, handle the metadata of files. A prototype file system, PPMS, was developed following a similar pattern as PPFS. The test results demonstrate that PPMS operation efficiency outperforms MooseFS.

While DHT-based approaches offer promising results, some challenges remain, such as metadata migration for load balancing and avoiding hotspots during metadata operations. Nevertheless, the combination of DHT-based structures with innovative load-balancing strategies holds the potential to further optimize the performance and scalability of distributed file systems.

2.2.6 Bloom Filter Based Distribution Technique

The Bloom Filter [98, 99] is a mapping technique that employs hashing to efficiently query the membership of any element on MDS. Despite the possibility of false positives in membership queries, Bloom filters are widely used due to their space efficiency and fast lookup query response. In the context of large storage systems, the Hierarchical Bloom Filter Array (HBA) scheme is proposed [32]. This design utilizes two or more hierarchies of bloom filter arrays, with the first array representing recently used metadata buffers and the second maintaining all metadata allocation details for memory efficiency. While space-efficient, this approach incurs network traffic overhead due to multicasting.

To address multicasting issues, the Group-based Hierarchical Bloom-Filter Array (G-HBA) [2, 30] is introduced as an adaptive metadata management scheme. G-HBA dynamically aggregates metadata servers into groups, reducing metadata migration and multicast network traffic. Scalability is achieved by adding MDSs within groups with minimal migration.

For cloud storage services, Metadata as a Service (MaaS) [100] proposes a bloom filter-based solution to reduce data retrieval latency. The Cloud bloom filter consists of a global and a local bloom filter, along with a Bloomier Matrix Filter. The global bloom filter handles metadata creation and placement tasks, while the local bloom filter manages metadata update queries. Recurrent operations on metadata files are tracked and recorded in the Bloomier Matrix Filter, enhancing metadata operation performance. The

research also introduces a cloud data storage model with cipher key generation based on metadata characteristics, ensuring secure access to encoded data using cryptography. The model allows for key revocation with owner and metadata server consent, maintaining throughput and scalability while ensuring data security.

A load-aware placement policy is proposed to distribute and migrate data based on load [20]. The measuring of load factor is a combination of data stream load and disk consumption. Based on these factors and migration, the probability load-aware data placement method improves the utilization and throughput of server disk bandwidth. The idea behind the Semantic-based distribution technique is to control the lookup query in a single group of semantically correlated groups and avoid search in the whole system. SmartStore [101] exploits file semantic information to provide scalability and efficiency of lookup. It supports complex range and k-top lookup queries. The FAST [102] is another effort to address the necessity of real-time semantic search using correlation-aware hashing.

In summary, Bloom filter-based distribution techniques offer efficient mapping and querying of metadata, contributing to improved performance and scalability in large storage systems and cloud environments. The classification of various existing techniques is listed in Table 2.1.

2.2.7 Other Techniques

The DualFS technique utilizes two different file systems for data and metadata management, respectively

Distribution	Technique Name	Method	Article Code
Table-based Metadata Distribution	xFS [37]	B+ Trees, Manager-Map Table for distribution and sharing of resources	R6
	HeRMES [65]	B+ Trees , Hashing, MRAM	R8

Distribution	Technique Name	Method	Article Code
	zFS [41]	Object Storage Devices, Cooperative Cache, Distributed Transactions	R10
Hash-based Distribution	Vesta [70, 71]	Parallel View, Hashing	R4
	Lustre [68]	Interleaved Hash	R11
	RAMA [69]	Hashing	R7
	AbFS [103]	Two-level Hash/ Table	R30
	FPFS [73, 74]	OSD+	R24
	HVFS [76]	2-level Directory partitioning, Extendable Hashing	R25
	DIMS [77]	Directory Hashing, Sub-tree Partitioning	R19
	DLBH [78]	Directory Level based hash	R26
Static Subtree Partitioning Distribution	NFSv3 [79]	Distributed file system	R5
	AFS [80]	Distributed file system	R1
	CODA [81, 82]	Distributed file system	R3
	Sprite [83]	Distributed file system	R2
	SRB [84]	Geographical Distributed File System	R13
	MRFS [85]	Geo-Replicated Metadata	R32
	Gfarm [86]	Global Distributed File System	R21
	RBMMMS [87]	Replica, Graph	R15
	GDSS [88]	Distributed Match table	R14
	Farsite [90]	Replication	R9
Dynamic Subtree Distribution	GIGA+ [26]	Partitioning, B-link tree or extendible hashing	R18
	OBFS [12]	OSD	R12
	CEPH [44]	Decoupling, Dynamic load balancing	R16

Distribution	Technique Name	Method	Article Code
	DDG [5]	Distribution, Consistency	R27
	H2FS [89]	Hybrid Hierarchy Partitioning	R33
DHT-based Distribution	PPMS [97]	Peer-to-Peer	R34
	Kosha [35]	Peer-to-Peer	R17
	Granary [91]	DHT	R35
	Cassandra [92]	De-Centralized storage system	R22
	Dcache [93]	Caching in Client and Server side	R23
	DROP [104]	HDLB	R31
	RFS [94]	DHT Ring	R28
Bloom Filter Based Distribution	HBA [32]	Bloom Filter Array	R20
	GHBA [2, 30]	Group wise HBA	R29
	MaaS [100]	Cloud based metadata	R36
	FAST [102]	Semantic Aware Search, Dynamic and Adaptive namespace	R37

Table 2.1: Taxonomy of Various Existing Techniques.

FSMAC is a technique that leverages the benefits of Non-volatile Memory (NVM) to enhance metadata operations [105]. It separates data and metadata, placing metadata on NVM and data on disks. The byte address retrieval from the memory bus is used for efficient metadata access. FSMAC results in reduced I/O requirements, optimizing metadata management.

This technique proposes a reliable metadata allocation policy using scatter single-level cells for the multiple-level cells of NAND Flash [106]. It utilizes solid-state drives (SSD) with integrated circuits for persistent data storage, offering low access time and latency. The technique aims to optimize storage systems using byte-addressable non-volatile mem-

ory (NVM) and Flash-based SSDs for data-intensive applications [107, 108].

The flash translation layer (FTL) is proposed to be implemented on Non-volatile Memory (NVM) in solid-state drives (SSD) [109]. This design optimizes the storage system by utilizing byte-addressable NVM for efficient metadata access while storing data on SSDs.

PIMMS proposes byte-addressable Non-volatile Memory (NVM) for in-memory metadata management [109]. The actual data is stored on solid-state drives, while NVM is leveraged for fast metadata access, leading to improved performance. To reduce write amplification, non-small blocks are written separately. Parallel data streams are exploited in retrieving big files.

The regular small I/O operations are required for small files and are avoided through buffering. The data stream channel of the disk is proposed in which each data stream is attached to a non-dependent CPU core to each physical drive [110]. It exploits the multiple cores of processors and parallel data streams to provide an efficient multiple channel for key processing. In lookup, a skipping component traversal is proposed to reduce dependency on metadata caching [111]. It helps to achieve better performance highly in highly decentralized metadata.

User Group Sub-Directory (UGSD) based metadata management is an amalgamation of static subtree partitioning, hashing and table-based partitioning techniques [112]. A MDS caches UGSD metadata to retain a tree structure. A two-level mapping function creates a mapping table to locate metadata. Each MDS has complete metadata to reduce metadata migration and is responsible to process specified UGSD user requests. The periodic metadata synchronization maintains metadata consistency. The technique reduces the migration cost. In MDS failover, the user needs to be notified.

RDBMS-based metadata management is proposed in the A-HDFS framework to improve the robustness, availability, and scalability of the Hadoop Distributed File System [113]. The technique re-constructed internal components NameNode and MetaManager of HDFS. Deister proposed an improved block management scheme for data-intensive file systems such as HDFS using Intersected Shifted Declustering (ISD) [114]. In this

approach, no name node and data node autonomously is responsible for maintaining block-node mapping. ISD is a two-step block distribution algorithm and inverse function to calculate the list of blocks stored on the data node. The results show that it reduces the load of block management, memory spaces and maintenance cost and improves the scalability of metadata nodes.

An efficient metadata sync for consumer cloud storage service is proposed in UniDrive [115]. The metadata is stored and replicated in multi-cloud. The same metadata file is encrypted and syncs to client devices. UniDrive improved sync operation performance as it reduced metadata traffic between client and cloud storage. An empirical comparative study of five consumer cloud storage services, their design and performance is discussed [116]. The reasons for different performance delivery in all five different methods under various situations are presented.

In the next section, we conducted a comprehensive analysis of the various metadata distribution techniques and reported their strengths and weaknesses. Each technique has its unique advantages and limitations, and understanding these aspects is crucial for making informed decisions while implementing distributed metadata management in storage systems.

2.3 Strengths and Shortcomings of Techniques

The strengths of these techniques lie in their ability to achieve load balancing, scalability, and efficient metadata access. Some techniques, such as hashing and bloom filters, offer quick membership queries and high lookup efficiency. Others, like dynamic subtree-based distribution, adaptively distribute metadata and dynamically migrate it to balance the load across Metadata Servers (MDSs). Additionally, the table-based distribution provides failover capabilities and requires no metadata migration.

On the other hand, these techniques also have their weaknesses. For example, static subtree partitioning may lead to an imbalance in metadata distribution, resulting in some MDSs being heavily loaded while others are underutilized. Techniques involving hashing may encounter issues with false positives, leading to potential inaccuracies in metadata

retrieval. Dynamic subtree-based distribution, while providing adaptability, can incur overhead in metadata migration during MDS joining or leaving operations.

Furthermore, some techniques may suffer from network traffic overhead, multicasting complexities, and additional memory consumption. The management of metadata relationships, directory structures, and metadata consistency can also pose challenges in certain techniques.

Understanding these strengths and weaknesses allows researchers and system architects to make informed decisions while choosing the most suitable metadata distribution technique for a specific storage system. By considering the specific requirements and constraints of their application, they can optimize the performance and efficiency of distributed metadata management. The strengths and shortcomings of individual metadata management techniques are identified and are listed in Table 2.2.

Name (Code)	Strengths	Shortcomings
AFS (R1)	Time Sharing File System, Designed for sharing and communication, Caching File System, No metadata migration, Fast directory operations	Initial approach to Distributed File System, Low scalability, Hotspot, Lack of load balancing, Bottleneck issue with node managing root or sub-directory
Sprite (R2)	Achieve high performance through block caching and client caching, Low metadata migration, Achieve high recovery, Fast directory operations, Improve availability	Additional cache consistency cost, Low scalability, Hotspot, No dynamic metadata load balancing, Root or sub-directory node bottleneck issue, An initial approached to Distributed File System
CODA (R3)	Server Replication to improve availability, Disconnected operations, Low metadata migration, Achieve high recovery, Fast directory operations	Achieve low scalability, Hotspot, Uneven metadata load balancing, An initial approach to Distributed File System, Root or sub-directory node has bottleneck issue

Name (Code)	Strengths	Shortcomings
Vesta (R4)	Variable size independent partitioning of Data stored on I/O nodes, High Performance for I/O intensive Applications, Full Pathname Hashing	Hashing full pathname does not allow access to directory level permission control, Renaming of the directory is time-consuming and links are not possible, Issue of distributed cache coherence, Heavily client dependency for task completion and vulnerable to client failures
NFSv3 (R5)	Distributed File System, Designed to improve write throughput, Achieve high recovery, Fast directory operations, Improve availability	Initial approached to Distributed File System, Low scalability, Hotspot, No load balancing, Root or sub-directory node prone to bottleneck
xFS (R6)	Eliminate Central Server Bottleneck, Improved performance, Scalability, availability, Cost-effective, B+ Tree for caching, Manager-Map Table for distribution and sharing of resources, Disk write performance improvement	Excessive use of bandwidth limits the scalability, Managers nodes limit scalability in this initial technique of Decentralized File system, High lookup time, Low load balancing, Very high memory Overhead, High recovery time
RAMA (R7)	Hashing to distribute data, Improve I/O performance, Controlling disks through multiple CPUs, Improve lookup and high reliability	High metadata migration, Complex recovery operation, Slow directory operations
HeRMES (R8)	Exploits Magnetic RAM for performance improvement, Provide high-speed access to metadata and data, MRAM consistency checks, Minimize Disk Access, B+ Trees and Hash table for caching	Small MRAM not able to cache growing metadata, Prone to inconsistent metadata. Additional MRAM Consistency checks process required and limit scalability, High lookup time, Low Load balancing, Very high memory Overhead, High recovery time
Farsite (R9)	Low metadata migration, Achieve high recovery, Fast directory operations, Improve availability and reliability	Low scalability, Hotspot, do not have dynamic metadata load balancing, High parent-child relationship maintenance cost

Name (Code)	Strengths	Shortcomings
zFS (R10)	Decentralized Object Storage Devices, Cooperative Integrates memory of all participating machines into one coherent cache, Distributed Transactions	File managers, Lease Managers and Transaction servers adds complexity and limits to achieve scalability, Maintenance of File managers tables limits scalability, High lookup time, Low Load balancing, Very high memory Overhead, High recovery time
OBFS (R12)	Object-Based Storage Devices, High metadata performance, High scalability	Low metadata read performance, Memory overhead, High Migration cost, High recovery cost, Do not exploit locality
SRB (R13)	Geographical Distributed File System, Improve availability and reliability, High recovery, Fast directory operations,	Low scalability, Hotspot, do not have dynamic metadata load balancing
GDSS (R14)	Low metadata migration, Achieve high recovery, Fast directory operations, Improve availability and reliability	Low scalability, Hotspot, do not have dynamic metadata load balancing, High parent-child relationship maintenance cost
RBMMS (R15)	Low metadata migration, Achieve high recovery, Fast directory operations, Improve availability and reliability	Low scalability, Hotspot, do not have dynamic metadata load balancing, High parent-child relationship maintenance cost
CEPH (R16)	High scalability, High I/O performance, Fast directory operations, Decoupling of metadata, dynamic metadata load balancing	Migrate metadata for load balancing, high recovery cost, Parent-child relationship limit the load balancing and scalability
Kosha (R17)	Reliable, Shared File System, P2P enhancement of NFS, Load balancing	Resource consuming parent-child relationship maintenance, Large metadata migration, Slow recovery
GIGA+ (R18)	Scalability, Fast metadata indexing and lookup, High throughput, Fast directory operations	High recovery cost, Require large metadata migration for load balancing, do not exploit locality, Memory overhead
DIMS (R19)	Directory Hashing, sub-tree partitioning, improve lookup efficiency and reliability	Slow directory operations, Large metadata migration, Costly recovery operation

Name (Code)	Strengths	Shortcomings
HBA (R20)	Exploits Hierarchical Bloom filter array to reduce false positives and improve caching, High Scalability and Load balancing, Improve lookup performance	False positive of Bloom Filter, Broadcast and Flooding lookup queries impact the performance, Require extra caching to complete metadata function other than membership queries, Update and delete operations of metadata require more space and computation, Limited metadata locality, Small metadata migration required that effects recovery
Gfarm (R21)	Low metadata migration, Achieve high recovery, Fast directory operations, Improve availability and reliability	Low scalability, Hotspot, do not have dynamic metadata load balancing, Costly maintenance of parent-child relationship
Cassandra (R22)	Provide high availability, Guard system against a single point of failure, Small lookup time, Less memory overhead, Highly Scalable and Reliable, High data write throughput, Metadata Replication	High metadata migration cost, Resource intensive Recovery Operation, Logical structure maintenance, Non-deterministic lookup, do not exploit locality.
Dcache (R23)	Parent directory path ID used for metadata distribution, Chord improvement, Scalable, double layer cooperative caching for client intimacy algorithm and to improve repeated metadata queries	Slow directory operation, Large migration cost, Require additional resources for recovery operation, Logical structure maintenance cost, Non-deterministic lookup, Very limited metadata locality
FPFS (R24)	Improved OSD using hashing, Improve lookup efficiency and reliability	High metadata migration, complex recovery operation, Slow directory operations
HVFS (R25)	2-level Directory partitioning, exploit extendable hashing, Improve lookup efficiency and reliability	Slow directory operations, Large metadata migration, Costly recovery operation
DLBH (R26)	Directory Level hashing, Improve lookup efficiency, Avoid root-node bottleneck, Low memory overhead, Improve reliability	High metadata migration, Slow directory operations, Resource consuming recovery, Low scalability

Name (Code)	Strengths	Shortcomings
DDG (R27)	High scalability, Improve metadata throughput, Even metadata load distribution, Two-phase commit for metadata consistency, Fast directory operations	Memory overhead, Metadata migration cost, High recovery cost, do not exploit locality, Costly maintenance of the parent-child relationship, Additional resources for recovery and consistency
RFS (R28)	Achieves high scalability, Fault tolerance and improve throughput using DHT Ring, High load balancing, Low memory overhead	High migration cost, additional logical structure maintenance cost, Non-deterministic lookup, High metadata migration, high resources required for recovery, Slow directory operations, Limited metadata locality
GHBA (R29)	Distributes metadata in a Groups wise hierarchal Bloom filter for caching, Reduce broadcasting to multicasting for lookup queries and Improve lookup performance, High Scalability and Load balancing	False Positive in lookup queries, Multicasting of lookup queries still impact system performance, Additional caching to complete metadata function other than membership function, Update and delete operation of metadata requires more space and computation, Limited metadata locality, Small metadata migration required that effects recovery
AbFS (R30)	Two-level Hash, Improve I/O performance, Exploit client cache, Improve lookup and high reliability	High metadata migration, Slow directory operations, Resource consuming recovery, Low scalability
DROP (R31)	Maintains metadata locality using locality-preserving hashing, Dynamic distribute metadata for load balancing, Low memory overhead, Highly scalable	High metadata migration, additional resources required for recovery and load balancing, Additional caching required for logical structure maintenance, Non-deterministic lookup, Controls metadata locality and In-efficient directory operations.
MRFS (R32)	Geo-Replicated metadata, Low metadata migration, Achieve high recovery, Fast directory operations,	No dynamic metadata load balancing, Low scalability, Issue of the hotspot in MDS, uneven metadata load balancing, Costly maintenance of parent-child relationship

Name (Code)	Strengths	Shortcomings
H2FS (R33)	Co-operate storage, Sharing oriented model, High scalability, Optimize I/O performance, Provide load balancing, Fast directory Operations	Additional memory overhead, do not exploit locality, Memory overhead, High metadata migration, Slow recovery
PPMS (R34)	Two-layer structure to achieve high scalability and low latency, High performance, Small Lookup time, Improve system throughput, Low memory overhead	Sub-directory operations are slow in the second layer and Require metadata migration, Slow recovery
Granary (R35)	A reliable data storage, Data sharing oriented model, Use DHT to scatter large data, Adaptive Recovery, High Scalability, Availability and Consistency, Improve throughput, Low memory overhead, Fast lookup operation	Rename and move operations require metadata migration, Slow directory operations, Resource intensive recovery
MaaS (R36)	Cloud Bloom filter, Bloom Filter used for efficient retrieval of metadata, Metadata dispersed geographically, Metadata as a Service, Probabilistic lookup, Reduces lookup latency, High Scalability and Load balancing	False Positive in lookup queries, Broadcasting/Multicasting in lookup queries, Non-deterministic lookup, Additional caching for other metadata operations, Update and delete operations of metadata require more space and computation, Small metadata migration, Recovery impact performance
FAST (R37)	Correlation-aware near-real-time retrieval, Cost-effective semantic queries using correlation-aware hashing, Use bloom filter for lookup, High Scalability and Load balancing	Small loss of metadata search accuracy, Multicasting and Broadcasting of lookup queries due to false positive, Update and deletion operations of metadata require more space and computation, limited metadata locality, additional caching for other metadata operations, Small metadata migration, Recovery degrades performance

Table 2.2: Strengths and shortcomings of metadata management techniques

2.4 Taxonomy of Distributed Structures and parameters

Metadata management techniques rely heavily on their distribution structures, and their effectiveness is assessed based on several critical parameters. These parameters encompass various factors essential for efficient metadata management, such as optimizing metadata lookup time, implementing effective load balancing techniques, considering scalability aspects to accommodate increasing data volumes, reducing memory overhead to ensure efficient resource utilization, minimizing the cost associated with metadata migration, devising robust recovery strategies to handle system failures, addressing locality concerns to improve data access efficiency, and employing replication techniques for data redundancy and fault tolerance.

The categorization of distribution structures used in metadata techniques is diverse, offering a range of approaches to handle metadata distribution efficiently. The key types of distribution structures include Static Subtree-based, Table-based, Hash-based, Dynamic Subtree-based, Distributed Hash Table-based, and Bloom Filter-based methodologies.

To provide a comprehensive overview, Table 2.3 summarizes the different distribution structures and their corresponding parameters, considering their strengths and weaknesses. The variables 'n' and 'd' in the table denote the total number of files and the total number of partitioned sub-directories within each file, respectively. This classification of distribution structures facilitates a deeper understanding of the various metadata management techniques and aids researchers and practitioners in selecting appropriate strategies based on their specific requirements and priorities.

Metadata Distribution Structures							
		1	2	3	4	5	6
Parameters	Code	Static Subtree based	Table based	Hash-based	Dynamic Subtree based	Distributed Hash Table based	Bloom Filter based
Scalability	P1	Very Low	Low	Low	High	High	Very High

Lookup Time	P2	$O(\log d)$	$O(\log n)$	$O(1)$	$O(\log d)$	$O(1)$	$O(1)$
Load Balancing	P3	No	Low	Medium	High	Very High	High
Memory Overhead	P4	Low	High	Low	Medium	Low	High
Recovery	P5	High	High	Low	Low	Low	Medium
Locality	P7	No	No	No	No	Limited	Limited
Migration	P8	Low	Low	High	High	High	Small
Directory Operations	P9	Fast	Slow	Slow	Fast	Slow	Medium

Table 2.3: Taxonomy of Distributed on Structures and their parameters

2.5 Parameters of Metadata Management Techniques

The key parameters of metadata management techniques encompass efficient metadata lookup time, load balancing, scalability, memory overhead, metadata migration, recovery, locality, and replication. These features have been categorized as P1 to P9 in Table 2.3. A detailed analysis of the strengths and weaknesses of existing metadata management techniques can be found in Table 2.4. This work presents a scale (–, +, ++, +++) to represent the parameters of each metadata management technique. In this scale, ‘–’ indicates the lowest value, while ‘+++’ indicates the highest, depending on the relative performance of the feature. The scale values are assigned carefully, considering the distribution structure’s strengths, weaknesses, potential improvements, and strategies to address any limitations. Table 2.4 provides the relative values of each parameter for individual techniques.

Features of Metadata Management Techniques										
Technique Name	Code	P1	P2	P3	P4	P5	P6	P7	P8	P9
FAST	R37	+++	+++	++	+++	++	++	+	+	++
MaaS	R36	+++	+++	++	+++	++	++	+	+	++

PPMS	R34	++	+++	+++	--	+	+	+	+++	+
Granary	R35	++	+++	+++	--	+	+	+	+++	+
H2FS	R33	++	+	++	+	+	+	--	+++	+++
MRFS	R32	--	+	--	--	+++	+++	--	--	+++
DROP	R31	++	+++	+++	--	+	+	+	+++	+
AbFS	R30	+	+++	++	--	+	+	--	+++	+
GHBA	R29	+++	+++	++	+++	++	++	+	+	++
RFS	R28	++	+++	+++	--	+	+	+	+++	+
DDG	R27	++	+	++	+	+	+	--	+++	+++
FPFS	R24	+	+++	++	--	+	+	--	+++	+
HVFS	R25	+	+++	++	--	+	+	--	+++	+
DLBH	R26	+	+++	++	--	+	+	--	+++	+
Cassandra	R22	++	+++	+++	--	+	+	+	+++	+
DCache	R23	++	+++	+++	--	+	+	+	+++	+
Gfarm	R21	--	+	--	--	+++	+++	--	--	+++
HBA	R20	+++	+++	++	+++	++	++	+	+	++
DIMS	R19	+	+++	++	--	+	+	--	+++	+
GIGA+	R18	++	+	++	+	+	+	--	+++	+++
Kosha	R17	++	+++	+++	--	+	+	+	+++	+
CEPH	R16	++	+	++	+	+	+	--	+++	+++
RBMMs	R15	--	+	--	--	+++	+++	--	--	+++
SRB	R14	--	+	--	--	+++	+++	--	--	+++
GDSS	R13	--	+	--	--	+++	+++	--	--	+++
OBFS	R12	++	+	++	+	+	+	--	+++	+++
Lustre	R11	+	+++	++	--	+	+	--	+++	+
zFS	R10	+	--	+	+++	+++	+++	--	--	+
Farsite	R9	--	+	--	--	+++	+++	--	--	+++
HeRMES	R8	+	--	+	+++	+++	+++	--	--	+
RAMA	R7	+	+++	++	--	+	+	--	+++	+

xFS	R6	+	--	+	+++	+++	+++	--	--	+
Vesta	R4	++	+	++	+	+	+	--	+++	+++
NFSv3	R5	--	+	--	--	+++	+++	--	--	+++
CODA	R3	--	+	--	--	+++	+++	--	--	+++
Sprite	R2	--	+	--	--	+++	+++	--	--	+++
AFS	R1	--	+	--	--	+++	+++	--	--	+++

Table 2.4: Relative Parameters of Metadata Management Techniques

2.5.1 Taxonomy of Techniques based on Scalability

Scalability is a crucial aspect of metadata management techniques, as it determines how effectively a system can handle the growing demand for metadata management without compromising performance or computational requirements. Achieving scalability while efficiently processing increasing user requests for metadata operations is essential for the success of a storage system’s metadata management.

Figure 2.2. presents a relative classification of distributed metadata management techniques based on their scalability, drawing insights from the discussions in sections 2.2, 2.3, and 2.4. The scalability of each technique is categorized as low, average, high, or very high, depending on the underlying metadata distribution techniques.

Static subtree-based distribution techniques, including MRFS, Gfarm, RBMMS, SRB, GDSS, Farsite, NFSv3, Coda, Sprite, and AFS, demonstrate low scalability due to their static nature. These techniques subdivide the file system tree and map metadata of these subtrees onto multiple MDSs. However, as some subtrees grow faster than others, uneven load distribution of metadata hampers scalability. Common file requests concentrate on a single server, creating hotspots while leaving other MDSs underutilized.

On the other hand, table-based and hash-based techniques show average or medium-level scalability. Techniques like AbFS, FPFS, HVFS, DLBH, DIMS, Lustre, zFS, HeRMES, RAMA, Vesta, and xFS are classified under this category. Centralized tables in these techniques limit scalability in very large distributed systems due to the challenges of

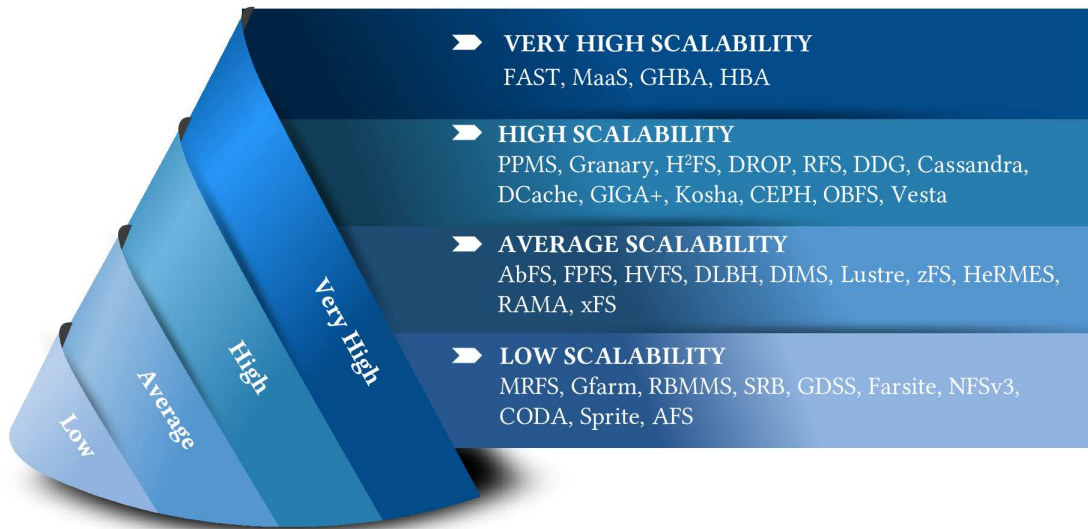


Figure 2.2: Taxonomy of Techniques based on Scalability.

sharing and maintaining such centralized structures, leading to potential hotspots and scalability limitations.

Achieving high and very high scalability requires employing dynamic and decentralized distribution techniques. Techniques such as DHT-based distribution, Dynamic Subtree-based, Bloom Filter-based, and Semantic-based approaches demonstrate higher scalability potential. These techniques dynamically distribute metadata across servers, effectively balancing the load and facilitating better scalability as the system grows.

To address the limitations of low and average scalability in metadata management techniques, the use of Distributed Hash Table (DHT) and dynamic subtree-based distribution techniques has been adopted. These approaches offer high scalability by enabling dynamic load balancing and adapting to changing user requirements. Techniques such as CEPH, PPMS, Granary, H²FS, DROP, RFS, DDG, Cassandra, DCache, GIGA+, Kosha, and OBFS fall under the high scalability category.

Another approach to achieve very high scalability is through the use of bloom filters for membership lookup queries in metadata lookup. Techniques like FAST, MaaS, GHBA, and HBA have successfully utilized this feature, demonstrating very high scalability.

By leveraging these advanced distribution techniques, metadata management techniques can efficiently handle increasing demands for metadata operations, effectively distribute

the load across servers, and adapt to changing user behavior. High and very high scalability is crucial for large-scale distributed systems, as it ensures optimal performance and responsiveness, even as the system grows in size and complexity. These scalable approaches pave the way for robust and efficient metadata management, meeting the challenges of modern data-intensive applications and distributed storage systems.

2.5.2 Taxonomy of Techniques based on Lookup

The efficiency of metadata lookup operations plays a crucial role in the performance of an ultra-large-scalable storage system. Improving the lookup operation directly enhances the overall efficiency of the storage system. Various metadata management techniques exhibit different complexities in their lookup operations based on their distribution structures.

Techniques that utilize bloom filters, distributed hash tables, and hash table-based approaches such as FAST, MaaS, GHBA, HBA, PPMS, Granary, DROP, RFS, Cassandra, DCache, Kosha, AbFS, FPFS, HVFS, DLBH, DIMS, Lustre, and RAMA demonstrate very small lookup complexity due to their well-designed distribution structures. These techniques offer highly efficient metadata lookup operations.

On the other hand, techniques like H2FS, DDG, GIGA+, CEPH, OBFS, Vesta, MRFS, Gfarm, RBMMS, SRB, GDSS, Farsite, NFSv3, CODA, Sprite, and AFS are based on static subtree and dynamic subtree metadata distribution structures. These techniques show slightly higher lookup complexities due to restrictions imposed by parent-child relationships in their distribution structures. They fall into the category of average lookup time.

Furthermore, table-based techniques like HeRMES, zFS, and xFS exhibit large metadata lookup time due to the presence of central mapping table structures that impact their overall performance. These techniques are classified under the category of large lookup time.

Figure 2.3 presents a classification of metadata management techniques based on their lookup time complexities. Techniques are relatively classified into small, average, and

large lookup time categories. Efficient lookup operations are crucial for high-performance storage systems, making the techniques with small lookup time more suitable for ultra-large-scalable storage applications.

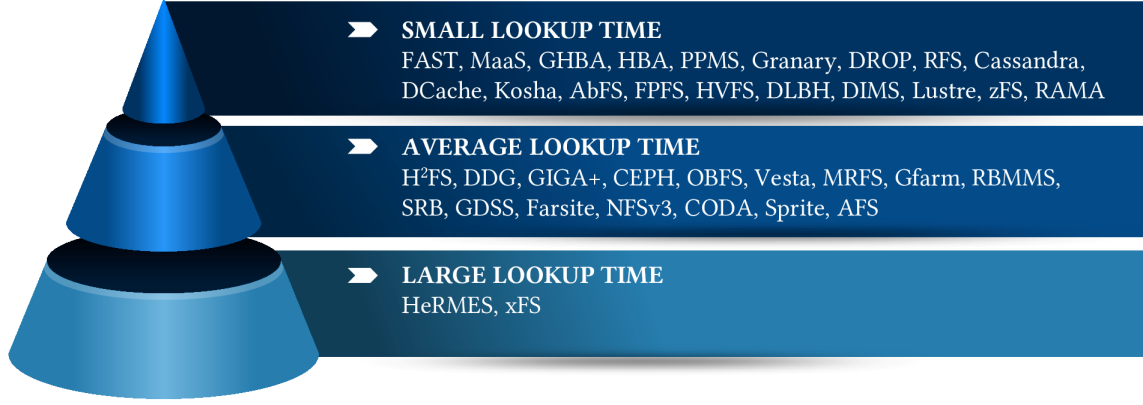


Figure 2.3: Taxonomy of Techniques based on Lookup time

These techniques are categorized relatively in average lookup time. Table-based techniques like HeRMES, zFS, and xFS have large metadata lookup time due to central mapping table structures and their performance. Based on the discussion in sections 2.2, 2.3 and 2.4, the techniques are relatively classified as small, average and large lookup time and are shown in Figure 2.3.

2.5.3 Taxonomy of Techniques based on Load Balancing

Load balancing is a crucial aspect in ensuring the efficient performance of an ultra-large-scalable storage system. Uneven distribution of metadata load can lead to MDS hotspots, which significantly degrade the overall performance of the distributed storage system. Techniques based on distributed hash tables (DHT) inherently possess load-balancing features, making them well-suited for achieving even distribution of metadata load.

Techniques utilizing DHT, such as PPMS, Granary, DROP, RFS, Cassandra, DCache, and Kosha, demonstrate high load-balancing capabilities and are classified under the category of very high load balancing. These techniques dynamically adapt to changing workloads, ensuring efficient load distribution.

Similarly, techniques based on bloom filters and dynamic subtree distributions, such as

FAST, MaaS, GHBA, HBA, AbFS, FPFS, HVFS, DLBH, DIMS, Lustre, RAMA, H2FS, DDG, GIGA+, CEPH, OBFS, and Vesta, are grouped in the high load balancing category. These techniques also exhibit load-adaptive features, which contribute to their efficient load balancing.

In contrast, table-based techniques like HeRMES, zFS, and xFS have limited load-balancing capabilities due to their central and less adaptive structures. These techniques are relatively less effective in achieving even distribution of metadata load.

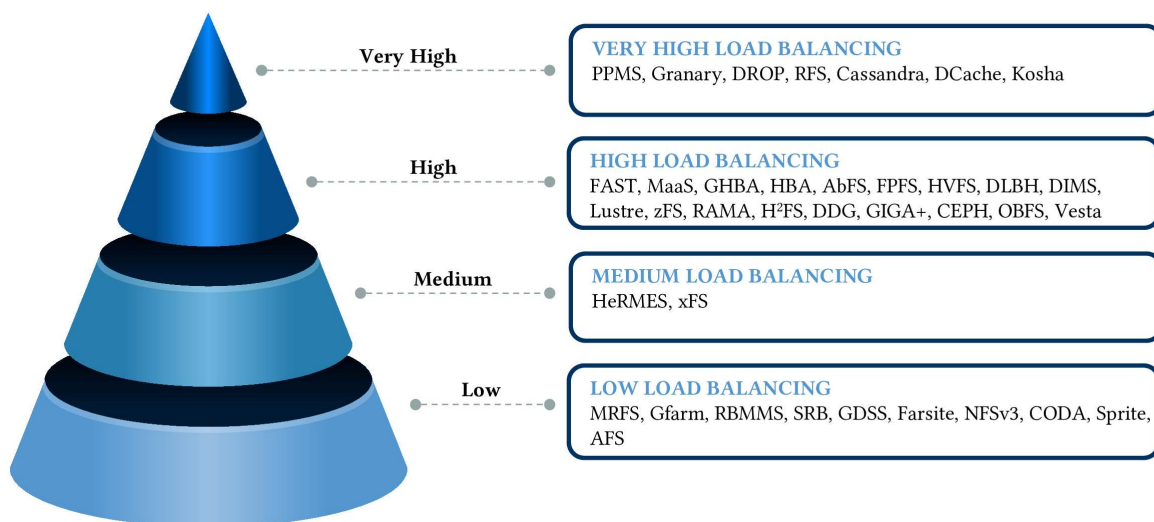


Figure 2.4: Taxonomy of Techniques based on Load Balancing.

Furthermore, static subtree-based metadata distribution techniques, including MRFS, Gfarm, RBMMS, SRB, GDSS, Farsite, NFSv3, CODA, Sprite, and AFS, either lack or exhibit very low load-balancing features due to the limitations of their static subtree structures.

Figure 2.4 presents a classification of distributed metadata management techniques based on their load-balancing capabilities. Techniques are relatively classified into very high, high, low, and very low load balancing categories. Efficient load balancing is crucial for maintaining the scalability and performance of an ultra-large-scalable storage system.

2.5.4 Taxonomy of Techniques based on Memory Overhead

Efficient memory usage is a critical parameter for ensuring optimal metadata operations in an ultra-large-scalable storage system. Different metadata management techniques exhibit varying levels of memory overhead based on their design and features.

Static subtree, hash, and distributed hash table (DHT) based techniques, such as PPMS, Granary, DROP, RFS, Cassandra, DCache, Kosha, AbFS, FPFS, HVFS, DLBH, DIMS, Lustre, RAMA, MRFS, Gfarm, RBMMS, SRB, GDSS, Farsite, NFSv3, CODA, Sprite, and AFS, demonstrate small memory overhead due to their simplicity and self-organizing characteristics. These techniques efficiently utilize memory resources and require relatively low memory overhead.

On the other hand, techniques like H2FS, DDG, GIGA+, CEPH, OBFS, and Vesta, which are based on dynamic subtree distributions, require additional memory to support their dynamic nature.

Table-based techniques like HeRMES, xFS, and zFS necessitate higher memory overhead for caching and sharing purposes, which can affect their memory efficiency.

Techniques utilizing bloom filters, such as FAST, MaaS, GHBA, and HBA, incur additional memory overhead for update and delete operations of metadata, which impacts their memory utilization.

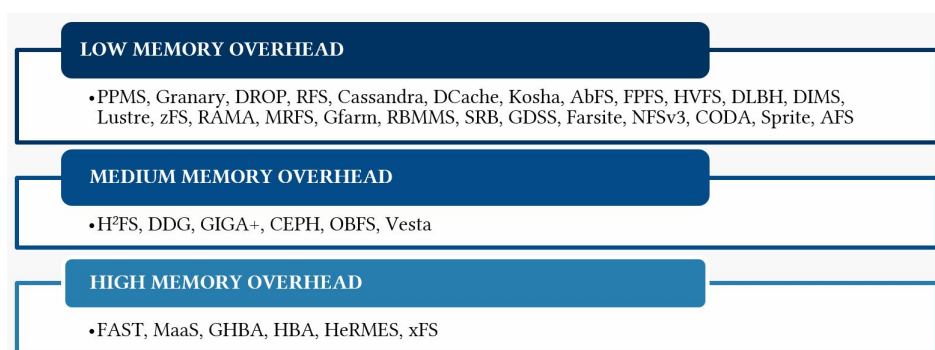


Figure 2.5: Taxonomy of Techniques based on Memory Overhead.

Figure 2.5 presents a taxonomy of distributed metadata management techniques based on their memory overhead. Techniques are relatively classified into small, average, and high memory overhead categories. Efficient memory utilization is crucial for achieving

optimal metadata operations and overall system performance in an ultra-large-scalable storage system.

2.5.5 Taxonomy of Techniques based on Recovery

The recovery feature of a metadata management technique is crucial in ensuring the system's resilience against failures and maintaining its state. The presence of dynamic load balancing in a technique leads to metadata migration, which, in turn, requires additional computation, memory, and bandwidth to complete the recovery operation. As a result, techniques with high metadata migration tend to have lower recovery capabilities and incur higher recovery costs.

Simple and static techniques generally achieve high recovery as they involve less dynamic metadata distribution and migration. Techniques like MRFS, Gfarm, RBMMS, SRB, GDSS, zFS, Farsite, HeRMES, xFS, NFSv3, CODA, Sprite, and AFS are based on static subtree and table-based metadata structures and exhibit high recovery.

In contrast, bloom filter-based techniques like FAST, MaaS, GHBA, and HBA demonstrate average metadata migration and subsequent average recovery capabilities.



Figure 2.6: Taxonomy of Techniques based on Recovery.

Furthermore, techniques like PPMS, Granary, H2FS, DROP, AbFS, RFS, DDG, FPFS, HVFS, DLBH, Cassandra, DCache, DIMS, GIGA+, Kosha, CEPH, OBFS, Lustre, RAMA, and Vesta are based on dynamic metadata distribution structures, leading to high meta-

data migration and subsequent lower recovery efficiency.

Figure 2.6 illustrates the classification of metadata management techniques based on their recovery features. It is essential to consider the recovery aspect when selecting a metadata management technique to ensure data integrity and system robustness in the face of failures.

2.5.6 Taxonomy of Techniques based on Replication

The replication of metadata is crucial for achieving high availability and ensuring system uptime in distributed metadata management. Geographic replication is employed to prevent system outages, and incremental replication and consistency checks are essential to keep track of system changes. The process of metadata migration in distributed systems directly affects replication and its consistency.

All metadata management techniques provide replication to achieve high availability and maintain the replication factor. The level of replication in these techniques depends on the amount of metadata migration and dynamic load balancing they employ.

Techniques based on static subtree and table-based approaches, such as PPMS, Granary, H2FS, DROP, AbFS, RFS, DDG, FPFS, HVFS, DLBH, Cassandra, DCache, DIMS, GIGA+, Kosha, CEPH, OBFS, Lustre, RAMA, and Vesta, can achieve high replication due to their limited metadata migration. These techniques exhibit a more stable distribution of metadata and require less frequent updates.

On the other hand, techniques like FAST, MaaS, GHBA, and HBA, which have small metadata migration, achieve medium replication. These techniques strike a balance between dynamic load balancing and replication efficiency.

Meanwhile, techniques like MRFS, Gfarm, RBMMS, SRB, GDSS, zFS, Farsite, HeRMES, xFS, NFSv3, CODA, Sprite, and AFS achieve low replication. These techniques necessitate higher levels of metadata migration to maintain dynamism and load balancing, which can affect their replication efficiency.

Figure 2.7 illustrates the classification of metadata management techniques based on the discussion in sections 2.2, 2.3 and 2.4 and replication capabilities. Appropriate consider-

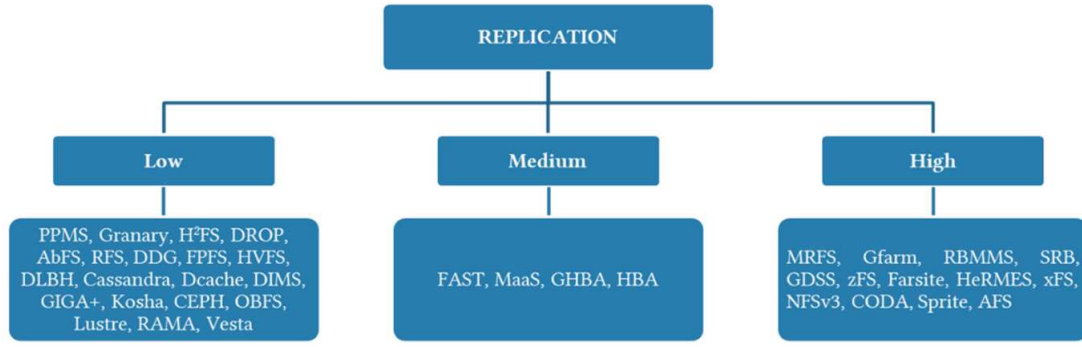


Figure 2.7: Taxonomy of Techniques based on Replication.

ation of replication factors is vital in distributed systems to ensure high availability and fault tolerance.

2.5.7 Taxonomy of Techniques based on Locality

The locality of metadata is a crucial parameter in distributed metadata management techniques as it directly impacts the performance of lookup operations and reduces network latency. While existing techniques primarily focus on parameters like scalability, lookup time, load balancing, and memory overhead, they often have limited control over maintaining metadata locality.

Bloom filter-based and DHT-based techniques, such as FAST, MaaS, GHBA, HBA, PPMS, Granary, DROP, RFS, Cassandra, DCache, and Kosha, achieve some level of control over locality. However, other techniques, including MRFS, Gfarm, RBMMS, SRB, GDSS, zFS, Farsite, HeRMES, xFS, NFSv3, CODA, Sprite, AFS, H2FS, AbFS, DDG, FPFS, HVFS, DLBH, DIMS, GIGA+, CEPH, OBFS, Lustre, RAMA, and Vesta, have no control over the locality of metadata placement.



Figure 2.8: Taxonomy of Techniques based on Locality.

Figure 2.8 presents the taxonomy of metadata management techniques based on their level of control over locality. Techniques with better control over locality can achieve improved lookup performance and reduced network latency, leading to enhanced overall system efficiency. Proper consideration of metadata locality is essential in designing efficient and high-performance distributed storage systems.

2.5.8 Taxonomy of Techniques based on Migration

Metadata migration is a crucial parameter that significantly impacts various aspects of a metadata management technique, including recovery, replication, directory operations, and load balancing. Techniques with high metadata migration tend to increase the complexity of metadata management and may require additional bandwidth to handle the migration process efficiently.

Based on the detailed discussions in sections 2.1, 2.2, 2.3, and 2.4 the metadata management techniques are classified based on their metadata migration parameters, as depicted in Figure 2.9. Techniques with fixed and static structures, such as MRFS, Gfarm, RBMMS, SRB, GDSS, zFS, Farsite, HeRMES, xFS, NFSv3, CODA, Sprite, and AFS, which are based on static subtree and table-based approaches, demonstrate very low metadata migration. This is mainly due to the limited dynamism and static nature of their distribution structures.

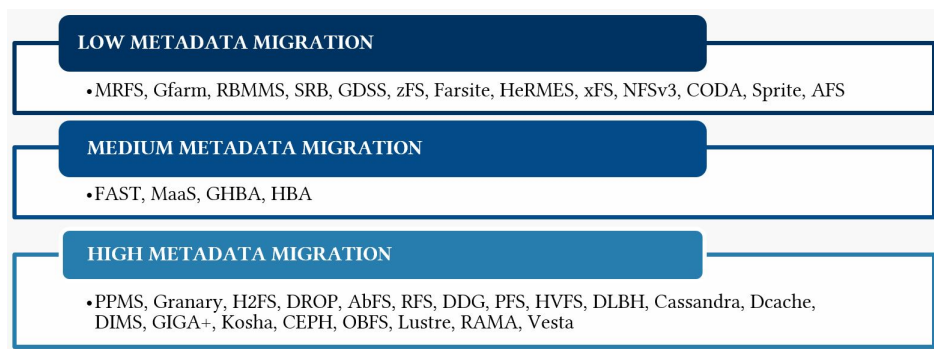


Figure 2.9: Taxonomy of Techniques based on Migration.

On the other hand, bloom filter-based techniques like FAST, MaaS, GHBA, and HBA exhibit a requirement for small metadata migration. These techniques leverage bloom filters' efficiency in membership queries to optimize the migration process. Conversely,

dynamic structure-based techniques like PPMS, Granary, H2FS, DROP, AbFS, RFS, DDG, FPFs, HVFS, DLBH, Cassandra, DCache, DIMS, GIGA+, Kosha, CEPH, OBFS, Lustre, RAMA, and Vesta necessitate high metadata migration. These techniques employ dynamic distribution structures to achieve better load balancing and adaptability, but this results in increased metadata migration.

2.5.9 Taxonomy of Techniques based on Directory Operations

Directory operations play another crucial role in the performance of distributed storage systems. The efficiency of directory operations directly impacts the overall storage system performance. In a distributed environment, directory operations may require metadata migration, which in turn affects storage performance and recovery. Maintaining the parent-child relationships while distributing metadata is essential for successful directory operations.

Static subtree and dynamic subtree-based techniques, such as MRFS, Gfarm, RBMMS, SRB, GDSS, Farsite, NFSv3, CODA, Sprite, AFS, H2FS, DDG, GIGA+, CEPH, and OBFS, demonstrate fast directory operations. These techniques efficiently manage parent-child relationships and offer quick directory lookup and traversal. However, it should be noted that maintaining these relationships may lead to uneven metadata distribution and can limit the scalability of the storage system.

On the other hand, table-based, hash-based, and DHT-based techniques like zFS, HeRMES, xFS, PPMS, Granary, DROP, AbFS, RFS, FPFs, HVFS, DLBH, Cassandra, DCache, DIMS, Kosha, Vesta, Lustre, and RAMA exhibit slower directory operations. These techniques may face challenges in efficiently managing parent-child relationships due to their distributed nature and centralized data structures.

Bloom filter-based techniques like FAST, MaaS, GHBA, and HBA show medium directory operation performance. These techniques leverage the efficient membership lookup of bloom filters to achieve moderate directory operation speeds.

To achieve optimal directory operation performance in a distributed storage system, it is essential to strike a balance between maintaining parent-child relationships and ensuring

efficient metadata distribution. Based on the discussions in sections 2.2, 2.3, and 2.4, the relative classification of distributed metadata management techniques on directory operation performance is shown in Figure 2.10.



Figure 2.10: Taxonomy of Techniques based on Directory Operations.

2.5.10 Decoupling of Metadata

The decoupling of metadata and data operations for load distribution of metadata is key to large metadata management techniques. The peer-to-peer approaches have maintained balancing due to hashing techniques. Around the year 2003, the decoupling of metadata was realized as an important feature of the performance of distributed scalable storage systems. During this time, social networking platforms were also maturing. Hierarchical-based decentralized architecture maintains parent-child relationships for fast directory operations and is listed separately. Based on the discussion in sections 2.2, 2.3 and 2.4, Table 2.5 lists the techniques, year and type of architecture.

Decoupling metadata and data operations and achieving uniform load distribution of metadata are key aspects of distributed large metadata management techniques. Peer-to-peer approaches have effectively maintained load balancing through hashing techniques. In the early 2000s, around the year 2003, the importance of decoupling metadata became evident for the optimal performance of distributed scalable storage systems. This period also saw the maturation of social networking platforms.

Hierarchical-based decentralized architectures were developed to maintain parent-child relationships for fast directory operations and are considered separately due to their unique characteristics.

Based on the discussions in sections 2.2, 2.3, and 2.4, Table 2.5 provides a comprehensive list of techniques, along with their respective years of development and their categorization based on their characteristics.

Year	Metadata Architecture	TECHNIQUE NAME
2003-2007	Decentralized	zFS (2003), Kosha (2006)
	Hierarchical Based Decentralized	OBFS (2004), SRB (2005), GDSS (2005), CEPH (2006), RBMMS (2006), GIGA+ (2007)
2008-2012	Decentralized	HBA (2008), DIMS (2008), Cassandra (2010), Dcache (2010), GHBA (2011), RFS (2011), FPFS (2011), HVFS (2011), DLBH (2011).
	Hierarchical Based Decentralized	Gfarm (2010), DDG (2011)
2013-2017	Decentralized	DROP (2013), AbFS (2013), PPMS (2014), Granary (2014), MaaS (2015), FAST (2016)

Table 2.5: Taxonomy Based on Decoupling of Metadata

2.6 Conclusion

In this chapter, a systematic review of distributed metadata management techniques in very large storage systems was presented. The analysis provided a comprehensive overview of these techniques, focusing on metadata distribution structures, their strengths, weaknesses, and a detailed comparison based on key performance parameters.

The upcoming chapter will introduce the architecture of the proposed locality-aware metadata management (LaMeta) techniques. It will address the problem of formulat-

ing locality for metadata servers in metadata distribution techniques, aiming to improve metadata operation query routing and enhance the overall performance of the distributed storage system. The proposed approach will tackle the concerns identified during problem formulation and strive to achieve the objectives set forth in this research. By incorporating locality-aware strategies, the LaMeta techniques aim to optimize metadata management in distributed storage systems.

Chapter 3

Proposed LaMeta: Locality-Aware Metadata Management Technique for an Ultra-Large Distributed Storage System

The previous chapter presented a comprehensive review of the existing literature and examined the current state of distributed metadata management techniques in a very large scalable storage system. It also provided the existing research categorised these based on the distribution structures and key performance parameters.

This chapter aims to address the issue of the locality of metadata on MDSs in the metadata distribution techniques that improve metadata query routing and enhance the performance of the distributed storage system. This chapter also discusses the architecture of the proposed locality-aware metadata management (LaMeta).

This chapter describes the MDS path measurement for locality and the need for metadata server locality for performance enhancement. The chapter further discusses the collaborative filtering-based MF techniques Singular value Decomposition (SVD), SVD++, Regularized Non-Negative Matrix Factorization, and Stacked AutoEncoder for predicting the path ratings. The locality of metadata is a critical parameter of data storage and retrieval performance for an ultra-large distributed storage system. The data of an ultra-large distributed storage system is accessed through its distributed metadata servers (MDSs). The distributed MDSs store and cache partial metadata for scalability and performance. The MDSs are connected through varied latencies multipath network connections using various metadata distribution techniques. The locality of MDS in the metadata distribution techniques improves the metadata query routing and enhances the performance of the distributed storage system. Motivated by this insight, this chapter highlights the proposed

intelligent locality-aware metadata management technique based on a recommendation system that exploits MDS path ratings in a globally distributed ultra-large storage system.

The chapter is organized as follows: Section 3.1 presents a brief introduction of the locality of metadata on MDSs, followed by Locality in metadata distribution techniques in Section 3.2. The architecture of the proposed locality-aware metadata management (LaMeta) is presented in Section 3.3. The constituents of the LaMeta such as MDS path measurement for the locality, MDS path ratings and MDS path recommendation system are discussed in Sections 3.4.1, 3.4.2 and 3.4.3, respectively. The chapter discusses the collaborative filtering-based MF techniques such as Singular value Decomposition (SVD), SVD++, Regularized Non-Negative Matrix Factorization, and Stacked AutoEncoder techniques to predict the MDS server locality in section 3.4.4, 3.4.5, 3.4.6, and 3.4.7. The One-Hot encoding and MDS K NextHop cache and metadata operation query routing are discussed in section 3.5 and 3.6 respectively. Section 3.7 concludes the chapter.

3.1 Locality of metadata on MDSs

Metadata locality plays a vital role in shaping the efficiency of data storage and retrieval within an ultra-large distributed storage system. In these sophisticated systems, data retrieval occurs through distributed metadata servers (MDSs) that diligently store and cache fragmented metadata, thereby facilitating scalability and optimal operational efficiency. Interlinked by multi-path network connections characterized by variable latencies, these MDSs strategically employ a range of metadata distribution techniques. This orchestrated approach significantly contributes to the overall performance enhancement of the system. This section introduces and discusses the locality in distributed metadata management techniques for storage systems. The incorporation of MDS locality within metadata distribution techniques yields improvements in metadata query routing and consequently elevates the overall performance of distributed storage systems. Motivated by this insight, this work introduces an intelligent locality-aware metadata management technique based on recommendation system that exploits MDS path ratings in a globally distributed ultra-large storage system and proposes a novel locality-aware metadata query routing algorithm.

The metadata gets divided among multiple MDSs to improve caching, scalability, and accessibility. The metadata of the file system is divided into static subtree, allocated to MDS 1, 2, and 3, as illustrated in Figure 3.1. The tasks of handling metadata operations like creating, looking up, and retrieving are carried out by the metadata servers. It executes or routes these operations to other MDSs, as each MDS holds only a portion of the complete metadata of the entire storage system.

Currently, there is a lack of systematic approaches to effectively utilize and manage the locality of MDS [117, 118]. The selection of the NextHop MDS identifier for MDS caching is either done randomly or with predetermined patterns during the creation of the distribution structure. The effectiveness of an MDS path, characterized by the lowest latency, significantly impacts the success of MDS query routing and its overall storage and retrieval performance.

The primary goal is to enhance the efficiency of metadata query routing and enhance the

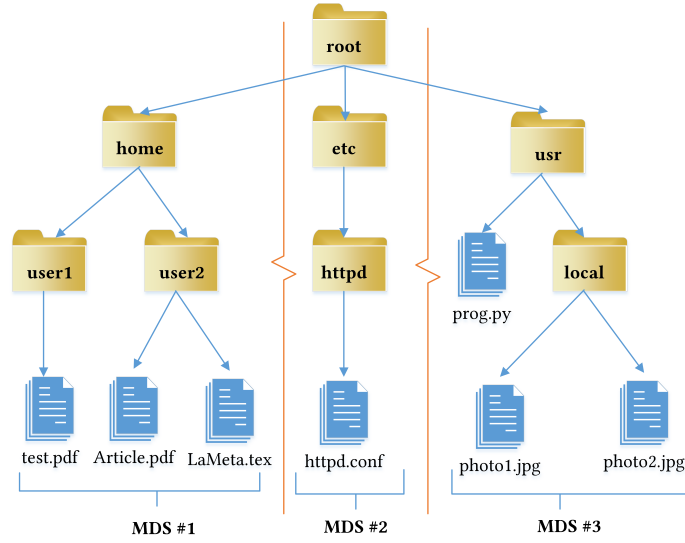


Figure 3.1: Static subtree metadata partition on MDSs

overall storage system performance. This is achieved by giving special attention to the locality of MDSs in metadata distribution techniques. Building upon this insight, this chapter proposes intelligent locality-aware metadata management technique, which leverages a recommendation system approach. This intelligent approach takes into account the ratings assigned to paths between MDSs, allowing the system to make well-informed choices regarding the most effective paths for metadata queries within a extensively distributed storage system. In this context, the concept of locality pertains to effectively handling metadata across distributed MDSs that are connected through paths with minimal latency. The aim is to optimize the routing of metadata queries and enhance the overall storage performance. The degree of MDS locality within distributed metadata performance is contingent upon the logical structure that interconnects the MDSs. Consequently, the efficiency of storing and retrieving metadata queries heavily relies on the latencies of these intermediate paths within the MDS network.

3.2 Locality in Metadata Distribution Techniques

Over time, numerous techniques for managing distributed metadata have been proposed for ultra-large distributed cloud storage systems [2, 118]. The primary emphasis has centered on the distribution of metadata, while the aspect of metadata server locality has remained unexplored within the realm of distributed metadata techniques designed

for cloud storage systems [117, 118].

The existing methods for distributing metadata are categorized into various types, including table-based, static subtree-based, hash-based, dynamic subtree-based, and distributed hash table-based approaches [117, 118]. The metadata distribution techniques follow a logical structure for metadata partitioning and query routing. The metadata query routing depends on next-hop decisions of the logical structure of the distribution technique. For the process of next-hop routing, the distribution techniques employ various routes such as parent-child, predecessor-successor, and long-short routes. The table-based technique, exemplified by systems like xFS and zFS, allocates metadata using a predetermined range of metadata keys. Notably, the concept of MDS locality is not integrated into the mapping of metadata ranges to MDSs in table-based metadata distribution strategies.

Static subtree distribution techniques involve partitioning metadata into non-overlapping subtrees, which are then linked to MDS. However, due to this fixed allocation, certain MDS may become disproportionately burdened with operations, while others remain underutilized. This load imbalance inherent in static subtree partitioning curtails both performance and scalability. Noteworthy implementations of this technique include NFS [79] and Sprite [83]. On the other hand, dynamic subtree distribution techniques slice metadata by employing hashing on directories near the root directory [2, 5, 117, 118]. This dynamic slicing enables the dynamic mapping of metadata to MDS. To maintain load equilibrium, metadata is continuously migrated to MDS. This approach finds application in systems such as CEPH [44], DDG [5], Kosha [35], and GIGA+ [119]. The MilkyWay-2 supercomputer’s file system also relies on dynamic subtree-based metadata distribution [89]. The MDS locality has not been explored in these techniques and still an unexplored area of research.

Hashing techniques are ingeniously harnessed to achieve effective load balancing of metadata lookups [117]. Notable file systems, such as ABFS [103] and Vesta [71], leverage hash-based mapping for metadata allocation and subsequent lookup operations. A unique approach, Lazy Hybrid (LH), employs full pathname hashing to manage hierarchical directories [39]. Worth noting is that this technique doesn’t consider the proximity to MDS

during metadata distribution. A distinct structure known as Dynamic Ring Online Partitioning (DROP) embodies a distributed hash overlay structure. Several DHT-based metadata distribution techniques have been proposed, including IChord [120], HDLB [120], and PPMS [97].

The metadata management distribution techniques and their performance parameters are summarized in Table 3.1 Here n represents total files, and d represent total partitioned sub-directories in a file system.

Techniques/ Parameters	Scalability	Lookup Time	Load Bal- ancing	Memory Over- head	Recovery	Replication	Locality	Migration
Static Sub- tree Based	Very Low	$O(\log d)$	No	Low	High	High	No	Low
Table Based	Low	$O(\log n)$	Low	High	High	High	No	Low
Hash-Based	Low	$O(1)$	Medium	Low	Low	Low	No	High
Dynamic Subtree Based	High	$O(\log d)$	High	Medium	Low	Low	No	High
Distributed Hash Table Based	High	$O(1)$	Very High	Low	Low	Low	Limited	High
Bloom Filter- Based	Very High	$O(1)$	High	High	Medium	Medium	Limited	Small

Table 3.1: Metadata Distribution Techniques and their performance parameters

The contemporary techniques lack a systematic approach to utilize and manage MDS locality [117, 118]. When constructing the distribution structure, the MDS cache NextHop MDS identifier is either randomly selected or pre-determined. The efficiency of an MDS path with minimal latency significantly influences the performance of MDS query routing.

This research endeavors to tackle the aforementioned challenges inherent in distributed metadata management of a very large storage system. Our forthcoming chapters will discuss the proposed technique design elements to address these challenges and discuss the impact on performance on distributed storage system.

3.3 Proposed Metadata Locality-aware Technique

In this section, we delve into the comprehensive architecture of the proposed LaMeta technique for a very large distributed storage system, along with its constituent elements. The

novel LaMeta is a metadata server locality aware recommendation system that exploits ordinal MDS path ratings and its machine learning-based prediction to explore locally available MDS for an ultra-large scalable globally distributed storage system. It capitalizes on ordinal MDS path ratings and employs machine learning-based predictions to navigate locally accessible MDSs in an ultra-large distributed storage system. The visual representation of the proposed LaMeta is presented in Figure 3.2

This chapter also discusses MDS path measurement, path rating, MDS paths recommendation techniques like matrix factorization techniques and Stacked AutoEncoder structure design in detail. The next chapter presents the probing and ratings of MDS interconnection paths, AutoEncoder structure design employed for the recommender system and also discusses the proposed the equation to express the locality of metadata query routing. The later chapter discusses the experiments and results.

This work proposes a locality-aware metadata management technique to improve the locality and aggregate throughput of metadata operation in distributed metadata management techniques by predicting the nearby MDSs path ratings. This research work proposes a novel metadata locality-aware technique (LaMeta), which exploits the MDS locality in a globally distributed ultra-large storage system. We evaluated the performance of collaborative filtering-based MF techniques and Stacked AutoEncoder. Collaboration filtering (CF) is one of the renowned recommendation techniques that can be used for predicting unavailable Quality-of-Service (QoS) values of Web services [121].

We propose LaMeta based on Stacked AutoEncoder (SAE) structure for efficient and intelligent MDS locality prediction in metadata distribution techniques. The work also proposes a next k MDS algorithm for metadata query routing and a locality function to computer the metadata operation locality. The extensive experiments and performance evaluations are performed using three real metadata datasets. The result shows that the proposed LaMeta technique has improved the locality and performance of baseline metadata distribution in terms of aggregate throughput and locality of metadata operations. We will discuss the constituents of proposed LaMeta techniques in detail.

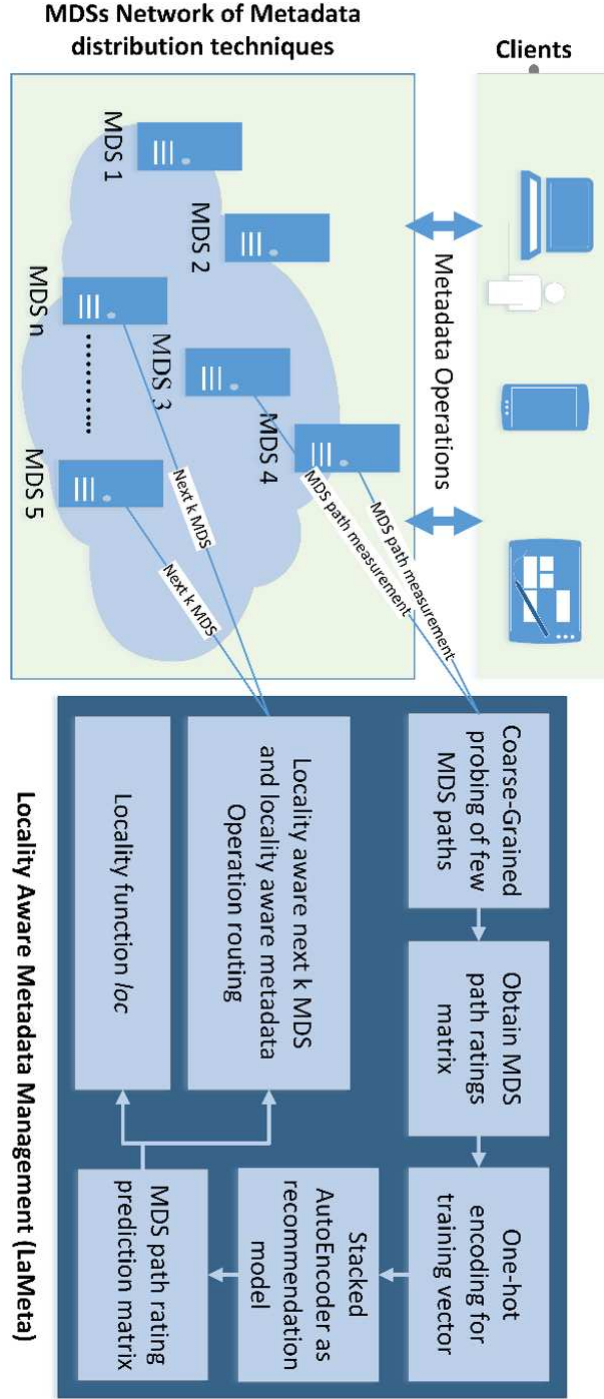


Figure 3.2: Architecture of proposed Locality Aware Metadata Management (LaMeta)

3.4 Constituents of Proposed LaMeta Technique

This section presents the various components of the proposed Locality-Aware Metadata Management (LaMeta) namely MDS path measurement, MDS paths recommendation

techniques in detail. The matrix factorization techniques like singular value decomposition (SVD), SVD++, Regularised Non-Negative Matrix Factorization (RNMF) and deep neural network Stacked AutoEncoder structure design are presented in detail. The other proposed components like one-hot encoding, AutoEncoder structure design employed for the recommender system, K next MDS routing, a proposed the equation to express the locality of metadata query routing and experiments and results are presented in next chapters.

3.4.1 MDS Path measurement for locality

Metadata servers (MDSs) play a pivotal role within distributed storage systems, being intricately linked through an intricate web of network connections. Each MDS has a multitude of redundant network paths that interconnect it with other MDS servers, forming the backbone of the system's communication infrastructure. As depicted in Figure 3.2, the MDS path connection matrix offers a comprehensive visualization of these intricate interconnections. In this representation, the green entries denote established MDS paths, highlighting the known connections. Conversely, the grey entries signify unknown network paths that await measurement. Notably, the matrix's diagonal entries represent self-paths, distinctly marked with a light blue hue, underscoring the unique connections each MDS maintains with itself. This visual representation illuminates the complex network topology that underpins distributed storage systems, inspiring further exploration into optimizing path utilization and enhancing locality for the performance of MDS. Figure 3.3 Figure 3.4 and Figure 3.5. Representation of MDSs path matrix.

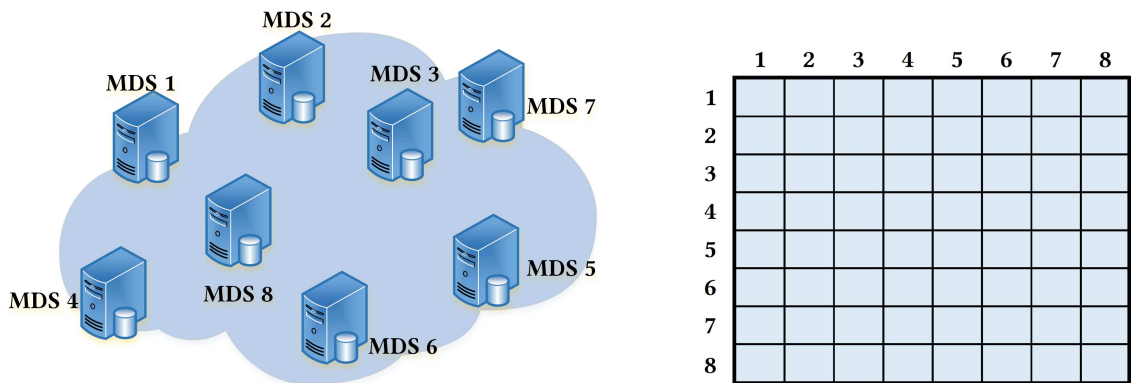


Figure 3.3: Represents 8 numbers of MDSs distributed globally and a matrix of with no connections

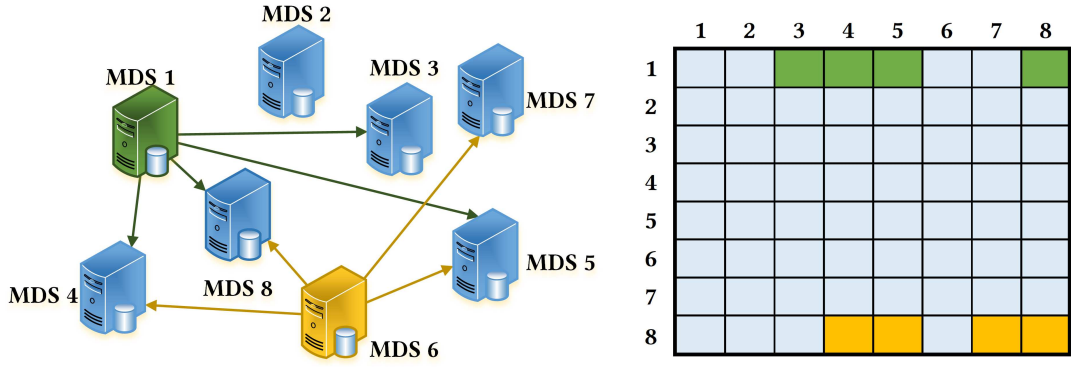


Figure 3.4: MDS1 connecting MDS3, MDS4, MDS5, and MDS8 are represented in green color. MDS6 connecting MDS4, MDS5, MDS7, and MDS8 are represented in yellow color.

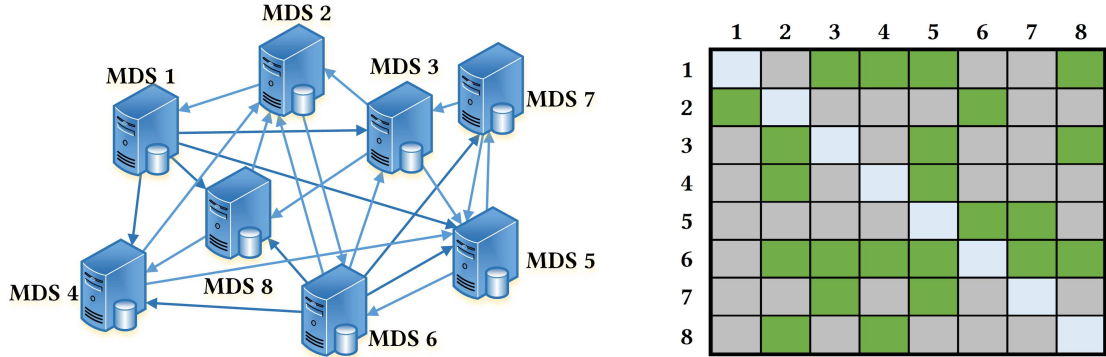


Figure 3.5: The green entries represent known MDS paths, and the grey represents unknown MDS paths. The diagonal MDS path are self-paths and represents with light blue color.

The network traffic prediction is considered a very important and challenging task for latency-sensitive networks such as IoT and storage [122]. The MDS network path’s round-trip time (RTT) can be leveraged to identify low-latency connections between MDSs. The quadratic complexity of measuring all MDS paths poses a significant scalability challenge. Achieving fine-grained and precise feature detection for network paths necessitates substantial resources. In response, we introduce a solution aimed at mitigating this issue. Our approach involves employing coarse-grained probing, targeting a subset of MDS paths for evaluation. This curated subset is then leveraged to derive MDS path ratings, which, in turn, enable the prediction of ratings for unknown MDS paths.

Interestingly, the concept of metadata server locality bears a resemblance to challenges encountered in recommender systems [123]. The MDS path attribute RTT can be a friendship measure between other MDSs. The next MDS hop selection for metadata

query routing can be expressed as a friend recommendation. This reasoning allows us to predict MDS locality via MDS path ratings using recommendation systems. The strategic integration of recommendation systems empowers us to predict these unknown path ratings efficiently. We have chosen matrix factorization-based techniques like SVD, SVD++, RNMF and a stack autoencoder to predict unknown path ratings. Through the translation of a broad spectrum of MDS path measurement values into a narrower range of MDS path ratings, we streamline the complexity of the data while retaining essential insights. we have chosen 1-5 rating scale where 1 is the lowest locality or most elevated RTT while rating 5 corresponds least RTT or best locality. This refined system-generated recommendation value plays a pivotal role in influencing recommendation outcomes, ensuring alignment with the dynamic nature of the system. This innovative approach addresses scalability concerns and enhances our predictions’ precision and adaptability, propelling our endeavour to optimize MDS path ratings within distributed storage systems.

The sparsity of this path-rating matrix is another issue that increases the complexity of its storage and processing. The matrix factorization (MF) techniques can obtain low-rank matrices to address the space complexity of the high-rank original rating matrix [124–126]. The deep neural network-based Stacked AutoEncoder can be exploited for feature reduction and predicting unknown ratings [127–130]

The task ahead involves establishing a mechanism to convert RTT values into ratings. In the subsequent section, we delve into how we plan to measure and represent these values.

3.4.2 MDS path ratings

MDS path rating P_r is represented by ordinal numbers of $P_r = [1 - 5]$. This rating representation ensures compact storage, efficient processing, and minimal transmission costs due to its concise nature. This rating is derived from RTT (Round-Trip Time) values, where a rating of 5 corresponds to the most favorable RTT or exceptionally high locality. Conversely, a rating of 1 signifies an elevated RTT, indicative of significantly lower locality. Our LaMeta technique is strategically designed to optimize MDS connections characterized by high locality, orchestrating metadata operation routing with

precision. By enhancing the concentration of high-locality connections, our approach serves to augment the overall efficiency and performance of the metadata management technique, aligning with our primary objective.

3.4.3 MDS Path Recommendation System

Performing real-time measurements for all MDS paths exacts an additional load on both MDSs and the MDS network. The challenge of sparsity within the path-rating matrix further exacerbates complexities related to storage and processing. To address this, matrix factorization (MF) techniques emerge as a viable solution, allowing the derivation of low-rank matrices capable of mitigating the space complexity inherent in the original high-rank rating matrix [124–126]. Additionally, the utilization of Stacked AutoEncoders a deep neural networks, presents an opportunity for feature reduction and predicting unknown ratings [126–130]. Deep neural network-based predictive models have revolutionised general practices in many healthcare applications [131]. These predictive techniques successfully augmented effective decision-making in the medical field, such as Diabetes, EEG, ECG, Lung diseases, etc [132–135]. The predictive models are also exploited in techniques for software defect predictions [136].

Expressed mathematically, we consider an $n \times n$ matrix of rank r , where r has very small values relative to n . This relationship is articulated through the equation:

$$X = PQ^T \tag{3.1}$$

Here, P and Q stand as low-rank matrices, each sized $n \times k$. The strategic employment of matrix factorization and deep learning architectures holds the potential to streamline both computation and prediction, effectively optimizing the intricacies of metadata operation processing.

The matrix factorization techniques are exploited to solve the data sparsity problem. The rating prediction exploited by collaborative-based recommendation systems has been commonly exploited in many fields [137–142]. Within this work, our approach encompasses the utilization of various recommendation techniques to predict unknown MDS

path ratings. Specifically, we harnessed techniques including Singular value Decomposition (SVD) with Bias, SVD++, Regularised Non-Negative Matrix Factorization (RNMF), and harnessed the power of a deep neural network via the Stacked Autoencoder (SAE). The next section will discuss these MF techniques and their mathematical representations.

3.4.4 Singular value Decomposition (SVD) with Bias

Singular Value Decomposition (SVD) serves as a mechanism to project source destination features onto a latent space of k -dimensions [129]. In this context, $R_{n \times n}$ symbolizes the rating matrix, $P_{n \times k}$ and $Q_{n \times k}$ denote the features of source MDS and destination MDS respectively. When considering a specific source MDS 's' and destination MDS 'd', the predictive rating $\hat{R}_{sd}$ is estimated as indicated by Eq. 3.2. Essential notations relevant to this process are succinctly outlined in Table 3.2. This approach inherently facilitates the exploration of feature relationships and contributes to the predictive capabilities within the context of MDS path ratings.

$$\hat{r}_{sd} = \mu + b_s + b_d + q_d^T p_s \quad (3.2)$$

Notations	Definition
n	Number of MDS
k	Number of latent factors
$R = [r_{sd}]$	MDS Path rating matrix
$\hat{R} = [\hat{r}_{sd}]$	Predicted MDS Rating matrix
μ	Average of all ratings
P, Q	Low ranked $n \times k$ matrix
γ	Learning Rate
λ	Regularisation parameter to avoid overfitting
p_s	Source MDS factors
q_d	Destination MDS factors
b_s	Source MDS biases
b_d	Destination MDS biases
y_j	The implicit destination factors
λ_s	Regularisation Parameter for source MDS
λ_d	Regularisation Parameter for destination MDS

Table 3.2: List of Notations

The prediction error is represented by Eq. 3.4

$$e_{sd} = r_{sd} - \hat{r}_{sd} \quad (3.3)$$

To optimize the parameters b_s, b_d, p_s, q_d with the aim of minimizing the regularized squared error outlined in Eq. 3.4, we employ stochastic gradient descent (SGD). The training process, as detailed in Eq.(3.5 - 3.8)., revolves around the use of SGD to achieve this objective.

$$\min_{b_s, b_d, p_s, q_d} \sum_{r_{sd} \in R} (r_{sd} - \hat{r}_{sd})^2 + \lambda \cdot (b_d^2 + b_s^2 + \|q_d\|^2 + \|p_s\|^2) \quad (3.4)$$

$$b_s \leftarrow b_s + \gamma \cdot (e_{sd} - \lambda \cdot b_s) \quad (3.5)$$

$$b_d \leftarrow b_d + \gamma \cdot (e_{sd} - \lambda \cdot b_d) \quad (3.6)$$

$$p_s \leftarrow p_s + \gamma \cdot (e_{sd} \cdot q_d - \lambda \cdot p_s) \quad (3.7)$$

$$q_d \leftarrow q_d + \gamma \cdot (e_{sd} \cdot p_s - \lambda \cdot q_d) \quad (3.8)$$

Where λ_s and λ_d are regularization parameters for source and destination MDS.

3.4.5 SVD++

SVD++ leverages implicit ratings and builds upon the foundation of SVD [129]. The computation of the prediction rating $\hat{r}_{sd}$ is determined in accordance with Eq. 3.9.

$$\hat{r}_{sd} = \mu + b_s + b_d + q_d^T \left(P_s + |I_s|^{-\frac{1}{2}} \sum_{j \in I_s} y_j \right) \quad (3.9)$$

$$\min_{b_s, b_d, p_s, q_d, y_i} \sum_{r_{sd} \in R} (r_{sd} - \hat{r}_{sd})^2 + \lambda \cdot \left(b_d^2 + b_s^2 + \|q_d\|^2 + \|p_s\|^2 + |I_s|^{-\frac{1}{2}} \sum_{j \in I_s} \|y_j\|^2 \right) \quad (3.10)$$

$$b_s \leftarrow b_s + \gamma \cdot (e_{sd} - \lambda \cdot b_s) \quad (3.11)$$

$$b_d \leftarrow b_d + \gamma \cdot (e_{sd} - \lambda \cdot b_d) \quad (3.12)$$

$$p_s \leftarrow p_s + \gamma \cdot (e_{sd} \cdot q_d - \lambda \cdot p_s) \quad (3.13)$$

$$q_d \leftarrow q_d + \gamma \cdot \left(e_{sd} \cdot \left(p_s + |I_s|^{-\frac{1}{2}} \sum_{j \in I_s} y_j \right) \right) \quad (3.14)$$

$$y_j \leftarrow y_j + \gamma \cdot \left(e_{sd} \cdot |I_s|^{-\frac{1}{2}} \cdot q_d - \lambda \cdot y_j \right) \quad (3.15)$$

The term y_j represents the implicit ratings. Through stochastic gradient descent (SGD), parameters are learned and the objective is to minimize the regularized squared error, outlined in Eq. 3.10. The training process to learn these parameters is mentioned in Eq. (3.11 - 3.14).

3.4.6 Regularised Non-Negative Matrix Factorization (RNMF)

RNMF capitalizes on the non-negative characteristics of input matrix values, ensuring that the training and optimization procedure generates positive factors to uphold the positivity of values. The optimization process employs regularized stochastic gradient descent to attain the minimum regularized squared error as indicated, with the training process detailed in Eq. (3.17-3.18)

$$\hat{r}_{sd} = \mu + b_s + b_d + q_{df}^T p_{sf} \quad (3.16)$$

$$p_{sf} \leftarrow p_{sf} \cdot \frac{\sum_{d \in I_s} q_{df'} r_{sd}}{\sum_{d \in I_s} q_{df'} \cdot \hat{r}_{sd} + \lambda_s |I_s| p_{sf}} \quad (3.17)$$

$$q_{df} \leftarrow q_{df} \cdot \frac{\sum_{s \in U_d} p_{sf'} r_{sd}}{\sum_{s \in U_d} p_{sf'} \hat{r}_{sd} + \lambda_d |U_d| q_{df}} \quad (3.18)$$

where λ_s and λ_d are regularisation parameters. the remaining parameters are optimized in the same manner as outlined in the SVD model.

Matrix factorization-based reduction techniques primarily handle linear computations, leading to limitations in predicting non-linear feature relationships. To address this, state-of-art deep learning neural models can be employed to extract non-linear relationships within data.

3.4.7 Stacked AutoEncoder for MDS path rating

Encoder learns and memorizes the important information about input value to reconstruct the output and bottleneck limits the amount of MDS path rating information, forcing the compression of the input data learning and reducing the path rating matrix dimensions [126–130]. This research exploits the SAE model for reduction and prediction of MDS path ratings. SAE is capable of learning complex non-linear relation input features for efficient feature reductions

The input dataset $\{x_n\}_{n=1}^N$, where $x_n \in R^{m \times 1}$, h_n is a hidden encoder vector and is mentioned in Eq.3.19.

$$h_n = f(w_1 x_n + b_1) \quad (3.19)$$

where f , w_1 and b_1 represent the activation function, weight vector, and bias vector, respectively.

$$x_n = g(w_2 h_n + b_2) \quad (3.20)$$

where g , w_2 and b_2 represent decoding function, weight vector, and bias vector respectively, to reconstruct the path rating matrix. The training comprises backpropagation to

minimize the reconstruction loss $loss(x, \hat{x})$.

$$\emptyset(\Theta) = \arg \min_{\theta, \theta'} \frac{1}{n} \sum_{i=1}^n L(x^i, \hat{x}^i) \quad (3.21)$$

where L represents a loss function $L(x, \hat{x}) = \|x - \hat{x}\|^2$.

3.5 One-Hot Encoding

One-hot encoding is a technique used in data preprocessing to represent categorical variables as binary vectors in deep neural networks. Each unique label is assigned a unique integer index. For each data point, a binary vector is created, where the index corresponding to the category is set to 1 and all other indices are set to 0. For example, consider a categorical variable "Color" with three possible values: Red, Blue, and Green. One-hot encoding is represented as Red: $[1, 0, 0]$, Blue: $[0, 1, 0]$ and Green: $[0, 0, 1]$

One-hot encoding ensures that the model doesn't interpret any ordinal relationship between the categories and treats them as distinct entities. One-hot encoding is used to convert categorical variables into a format that can be fed into machine learning models, as most models require numerical input. It's essential for tasks like classification, where the model needs to learn relationships between categories without imposing any unintended ordinal relationships.

In MDS network, we exploit hot-encoding in data preprocessing to represent source MDS and all designation MDS in a vector form. This data preprocessor vector is feed in Stacked Autoencoder. We will discuss this source designation vector representation with example in next chapter.

3.6 MDS K NextHop Cache and Metadata Operation Query Routing

In distributed metadata management techniques, the MDSs interact with other MDSs to execute metadata operation query routing. The MDSs also cache NextHop MDS identi-

fiers for metadata operation routing either randomly or in a predetermined manner. The efficient optimization of the connecting path with minimal latency is crucial in enhancing the performance of MDS query routing. The MDS operation, such as create, lookup, etc routes from one MDS to another MDS for lookup and execution. Identifying local or nearby MDS servers in metadata query routing poses a complex and unresolved problem in an ultra-large distributed storage system. We will discuss the proposed MDS K NextHop algorithm in next chapter.

3.7 Conclusion

This chapter introduces the locality of metadata on MDSs, followed by the architecture of the proposed locality-aware metadata management (LaMeta) and its constituents are presented. The constituents of the LaMeta, such as MDS path measurement for the locality, MDS path ratings and MDS path recommendation system are discussed. The chapter discusses collaborative filtering-based MF techniques such as Singular value Decomposition (SVD), SVD++, Regularized Non-Negative Matrix Factorization, and Stacked AutoEncoder techniques to predict the MDS server locality. The One-Hot encoding, MDS K NextHop cache and metadata operation query routing are discussed briefly.

The next chapter will introduce input data preprocessing of MDS rating matrix, the design of LaMeta based on Stacked AutoEncoder (SAE) structure for efficient and intelligent MDS path rating locality prediction in metadata distribution techniques. It will also discuss the next k MDS algorithm for efficient locality-aware metadata query routing and a locality function to computer the metadata operation locality in a very large distributed storage system. The chapter will also discuss the implementation platform for experiments.

Chapter 4

Design of the Proposed LaMeta Technique

The preceding chapter provided a concise overview of metadata locality on MDSs, followed by an exploration of locality in existing metadata distribution techniques. The architecture of the proposed locality-aware metadata management (LaMeta) and its components were introduced and analyzed. Additionally, the previous chapter delved into collaborative filtering-based matrix factorization techniques like Singular Vector Decomposition (SVD), SVD++, Regularized Non-Negative Matrix Factorization, and Stacked AutoEncoder methods. These techniques were discussed in the context of reducing space complexity and predicting MDS server locality.

This chapter aims to present the MDS path probing and path ratings for the locality of metadata on distributed MDSs. This chapter includes the detailed design structures of the LaMeta technique. This includes the structure of the proposed Stacked AutoEncoder, locality-aware next K MDS and their algorithms. The chapter presents the translation of MDS path RTT values to MDS path ratings and input data preprocessing using one-hot encoding for Stacked AutoEncoder. The chapter further discusses the proposed locality expression loc of metadata operation routing in value to measure and compare the impact of LaMeta on distribution metadata techniques in an ultra-large distributed storage system.

The chapter unfolds in a structured manner, each section building upon the previous one to comprehensively explore the topic. It commences with Section 4.1, discuss the proposed MDS path probing and path ratings, offering insights into how metadata locality is assessed on MDSs. This is followed by Section 4.2, where we delve into the MDS path ratings matrix and the input feature vector employed in the training dataset. Section 4.3 focuses on datasets and the transformation of MDS path ratings from RTT values.

The chapter further discusses the Stack AutoEncoder employed in LaMeta technique in Section 4.4, with an in-depth exploration of its structure and algorithm. Sections 4.5 and 4.5.1 introduce the proposed locality-aware routing and its associated Locality Aware Next K MDS algorithm, providing a comprehensive understanding of how metadata routing efficiency is enhanced within the system. Section 4.6 delves into the proposed locality function loc, elucidating its role in measuring the cumulative impact of intermediate MDS path ratings and its reflection on metadata query routing efficiency. Finally, Section 4.7 concludes the chapter.

In this chapter, we delve into the intricate design structures of the LaMeta technique, providing a comprehensive overview of the architecture of the proposed Stacked AutoEncoder, the concept of locality-aware Next-K MDS, and the associated algorithms. Additionally, we explore the process of translating MDS path RTT values into MDS path ratings and delve into the preprocessing of input data, where one-hot encoding is specifically utilized for the Stacked AutoEncoder. Furthermore, we discuss the proposed locality expression, which measures the impact of LaMeta on metadata operation performance using different distribution techniques.

4.1 MDS Path Probing and Path Ratings

As elaborated in the preceding chapter, the measurement of all fine-grained MDS paths entails quadratic complexity, thereby imposing constraints on scalability. To overcome this limitation, we introduce a solution involving the coarse-grained probing of a subset of MDS paths, leading to the translation of these values into ratings. Predicting unknown MDS path ratings are accomplished through the application of matrix factorization techniques and Stacked AutoEncoder, which are classified under collaborative filtering within recommendation systems. The merit of translating MDS path ratings lies in its ability to condense a wide array of MDS path measurement values into a narrower spectrum of rating values. These systematically generated rating values regulate the outcomes of recommendations and align more harmoniously with the system’s dynamics.

4.2 MDS Path Ratings Matrix

MDS path ratings, denoted as P_r , are expressed using ordinal numbers within the range of $P_r = [1 - 5]$. Furthermore, MDS path ratings exhibit efficiency in storage and processing, as they can be stored in a compact format. In this context, P_r symbolizes the MDS path rating, with a rating scale derived from RTT values. A rating of 5 signifies the lowest RTT, indicating very high locality, whereas a rating of 1 corresponds to high RTT value of MDS path, signifying very low locality.

The primary objective of our proposed LaMeta technique is to actively utilize MDS

connections characterized by high locality in metadata operation routing. This approach aims to enhance both the locality and overall performance of the metadata management technique.

Metadata operations in a distributed system often follow multiple paths as metadata is distributed across various Metadata Servers (MDSs). These operations involve a journey from the source MDS, where the metadata operation originates or is routed from another MDS, to the destination MDS, where the operation is executed. It is worth noting that there can be scenarios where there are multiple intermediate destinations between the source and the final MDS destination, and the operation may hop through these intermediate MDSs.

		<i>Destination MDS #ID</i>			
		<i>d1</i>	<i>d2</i>	<i>d3</i>	<i>d4</i>
<i>Source MDS #ID</i>	<i>s1</i>	?	2	?	1
	<i>s2</i>	5	?	4	?
	<i>s3</i>	3	5	?	2
	<i>s4</i>	4	?	5	?

Figure 4.1: The MDS path rating matrix of 4 source MDS and 4 destinations MDS

To visualize and represent the various MDS paths between the source and destination MDSs, we use a matrix-based representation, as illustrated in Figure 4.1. The Fig 4.1 presents a sample of 4x4 MDS Network. The MDS path ratings are represented in each cell corresponding to source and destination MDS. This matrix serves as a convenient way to depict the intricate network of path ratings that metadata operations can take as they traverse through the distributed system, providing valuable insights into the metadata operation routing within the system.

Each source MDS and destination MDS in our system possesses a distinctive path rating feature vector matrix, as illustrated in Figure 4.1. To provide a concrete example, let's consider the source MDS *s1*, which can be represented as a feature vector such as (0, 2, 0, 1). In this representation, we assign a value of 0 for all unknown MDS path ratings.

In addition to the path ratings, we incorporate the unique identification numbers of the

source and destination MDSs as additional features. To differentiate feature vectors belonging to different source and destination MDS pairs effectively, we employ a technique called one-hot encoding. This method enables us to encode the MDS identification numbers in a structured way, enhancing the distinction between various source-destination pairs.

As indicated in the scoring matrix displayed in Figure 4.1 we utilize this feature-rich dataset to construct the training dataset required for the Stacked AutoEncoder, as outlined in Table 4.7. This dataset forms the foundation for training and fine-tuning our Stacked AutoEncoder model, facilitating the prediction of MDS path ratings. All source MDS and destination MDS has a path rating feature vector matrix as mentioned in Figure 4.1. for example, MDS source s_1 may be represented as $(0,2,0,1)$ by treating 0 value for all unknown MDS path ratings. The source MDS ID and destination MDS ID are also taken as features to differentiate the same feature vector of different source and destination MDSs. To include the MDS identification number information one-hot encoding is exploited. As per the scoring matrix mentioned in Figure 4.1, the training dataset is created for Stacked AutoEncoder mentioned in Table 4.7.

Path Rating information	Training Data Set
$(s_1, d_2, 2)$	$t_1 = (1, 0, 0, 0, 0, 1, 0, 0, 5, 0, 4, 0, 2, 0, 5, 0, 2)$
$(s_1, d_4, 1)$	$t_2 = (1, 0, 0, 0, 0, 0, 0, 1, 4, 0, 5, 0, 1, 0, 2, 0, 1)$
$(s_2, d_1, 5)$	$t_3 = (0, 1, 0, 0, 1, 0, 0, 0, 0, 2, 0, 1, 0, 5, 3, 4, 5)$
$(s_2, d_3, 4)$	$t_4 = (0, 1, 0, 0, 0, 0, 1, 0, 3, 5, 0, 2, 0, 4, 0, 5, 4)$
$(s_3, d_1, 3)$	$t_5 = (0, 0, 1, 0, 1, 0, 0, 0, 0, 2, 0, 1, 0, 5, 3, 4, 3)$
$(s_3, d_2, 5)$	$t_6 = (0, 0, 1, 0, 0, 1, 0, 0, 5, 0, 4, 0, 2, 0, 5, 0, 5)$
$(s_3, d_4, 2)$	$t_7 = (0, 0, 1, 0, 0, 0, 0, 1, 4, 0, 5, 0, 1, 0, 2, 0, 2)$
$(s_4, d_1, 4)$	$t_8 = (0, 0, 0, 1, 1, 0, 0, 0, 0, 2, 0, 1, 0, 5, 3, 4, 4)$
$(s_4, d_3, 5)$	$t_9 = (0, 0, 0, 1, 0, 0, 1, 0, 3, 5, 0, 2, 0, 4, 0, 5, 5)$

Table 4.1: Training set construction

4.3 Datasets and MDS Path Ratings

For the evaluation of the recommendation system discussed, we have utilized two distinct datasets, the particulars of which are provided in Table 4.2. The RTT values within these datasets have been transformed into ratings, as mentioned in Table 4.3. These ratings are categorized within a scale of 1 to 5, with each interval illustrated in Table 4.3. Notably, the threshold value for ratings differs across the three datasets, as specified in Table 4.3. It is essential to emphasize that MDS paths characterized by very high and high ratings are instrumental in enhancing the routing performance of metadata operations executed on distributed metadata servers within a very large storage system. Conversely, path ratings categorized as low or very-low can potentially impede performance and are therefore advisable to avoid during metadata operations routing and performance optimization.

Source	Code	Hosts	Min RTT (ms)	Max RTT (ms)
Institute Firewall WAN IP	DS-I	1183	110	504
RIPE-Atlas measurement of ping	DS-II	478	6.95	483.5

Table 4.2: Dataset for experiments

Locality	MDS Path Ratings	Dataset-I	Dataset-II
Very High	5	110	87
High	4	179	94
Medium	3	193	117
Low	2	257	226
Very Low	1	504	483

Table 4.3: Rating Thresholds in RTT (ms)

4.4 Stacked AutoEncoder

The Stacked AutoEncoder (SAE) model is a sophisticated neural architecture consisting of multiple layers, resembling a hierarchical AutoEncoder structure. This model falls under the category of unsupervised learning and is graphically represented in Figure 4.2. SAE has ability to process data through a series of hidden layers, with each layer serving as both an encoder and a decoder. The output vectors generated by one layer are fed as

input into the subsequent layers of AutoEncoders (AEs), allowing for a hierarchical and progressive feature extraction process.

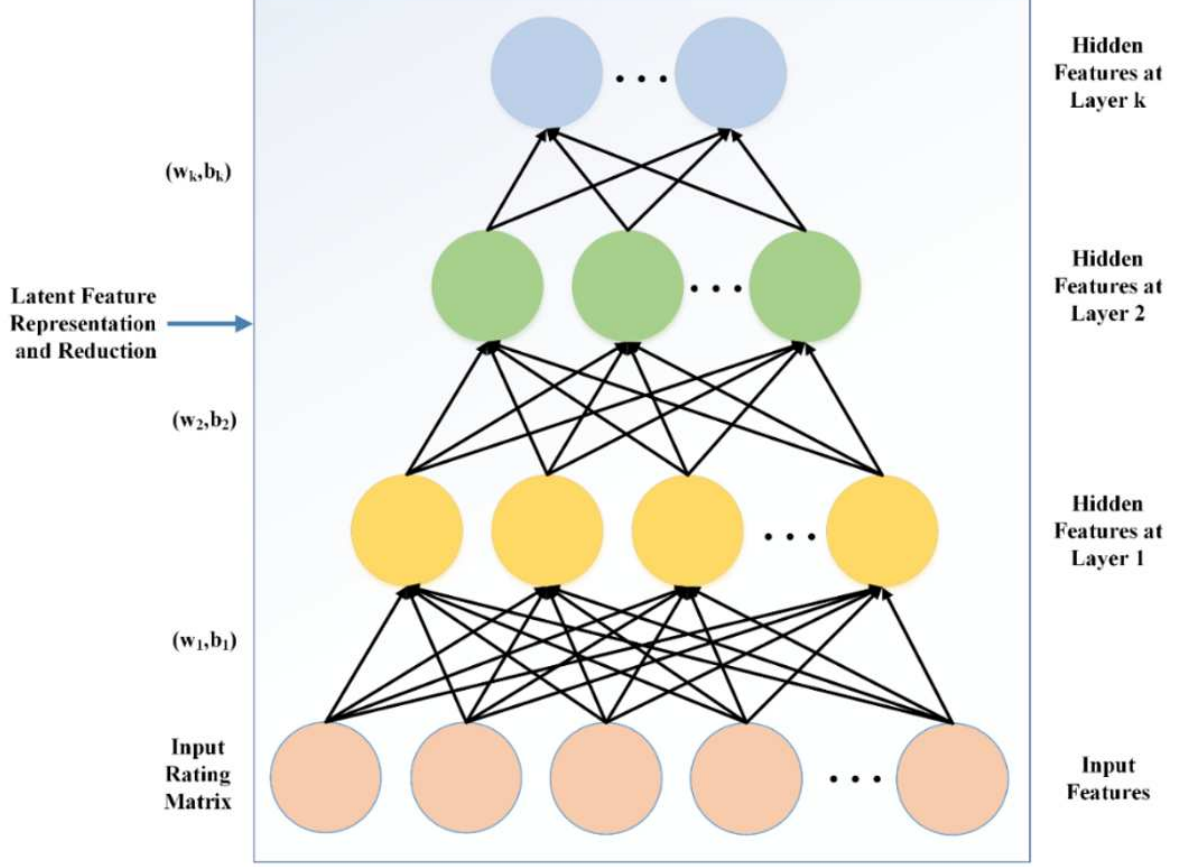


Figure 4.2: Structure of Stacked AutoEncoder (SAE)

The primary goal of the SAE is to perform a stepwise reduction of the high-level features initially present in the path rating matrix dataset. This reduction is achieved by encoding and decoding the data through the multiple layers of the SAE, gradually abstracting and representing the dataset in a more compact and informative manner. This process is outlined in detail in Algorithm 1, which governs the steps of the SAE model.

Here, we express the mathematical equations and notations for a three-layer Stacked AutoEncoder (SAE) algorithm, including the intermediate Z matrix, weight matrix, and bias matrix. The notations and mathematical equations for layers and intermediate Z matrix are mentioned below:

X : Input data matrix (with each column representing a data point)

W_1, b_1 : Weight matrix and bias vector for the first hidden layer

Z_1 : Intermediate representation (output) of the first hidden layer
 W_2, b_2 : Weight matrix and bias vector for the second hidden layer
 Z_2 : Intermediate representation (output) of the second hidden layer
 W_3, b_3 : Weight matrix and bias vector for the output layer
 Z_3 : Final output

The intermediate Z matrices, Weight Matrices and Bias Vectors at each layer will be represented by:

Intermediate Z Matrix (First Hidden Layer): $Z_1 = f(W_1 \times X + b_1)$

Intermediate Z Matrix (Second Hidden Layer): $Z_2 = f(W_2 \times Z_1 + b_2)$

Final Output Z Matrix (Output Layer): $Z_3 = f(W_3 \times Z_2 + b_3)$

Here $f()$ denotes an activation function. These equations represent the forward pass of the three-layer stacked AutoEncoder, transforming the input data into a compressed representation and then reconstructing it back to the output layer.

Once the feature reduction phase is complete, the SAE utilizes a trained softmax model, incorporating the high-level feature vector, along with the weight vector W' and bias vector b' . This combination enables the model to reconstruct the rating prediction matrix. The resultant matrix effectively encapsulates the intricate relationships and patterns present within the dataset, making it a valuable tool for predicting MDS path ratings and optimizing metadata operations routing.

This algorithm outlines the process of training a Stacked AutoEncoder (SAE) to extract and represent essential features from the input MDS path rating training set X . The SAE consists of multiple layers, and the goal is to iteratively learn the network connection weights $\{W_h, 1 \leq h \leq k\}$ and bias vectors $\{b_h, 1 \leq h \leq k\}$ for each layer. The output of the k -th hidden layer, denoted as Z_k , is obtained, and this representation can be used for prediction of MDS path ratings. Here's a breakdown of the algorithm steps:

- Initialization: Begin with the input feature matrix $Z_0 = X$, setting the initial representation to be the raw MDS path ratings.

Algorithm 1: Stacked AutoEncoder (SAE)

Input: MDS path rating training set X , the number of layers k , number of iteration noi

Output: Predicted MDS path rating matrix matrices of network connection weights $\{W_h, 1 \leq h \leq k\}$, bias vectors $\{b_h, 1 \leq h \leq k\}$, the output Z_k of the k -th number of hidden layer.

1. Initialize: $Z_0 = X$; //starts with an input feature matrix
 2. Obtain W_1 weight vector parameters obtained using the MDS path training set based on AutoEncoder, and output Z_1 is obtained on the trained parameters
 3. output Z_1 of the preceding step feed as input to the second AutoEncoder to obtain W_2 and Z_2
 4. Repeat Step 3 for all k hidden layers, and, respectively obtain W_k weight matrices and Z_i as an output feature matrix
 5. Fine-tuning the W_i and b_i based on Back Propagation till noi iterations
 6. Obtain W_i , b_i and Z_i
 7. $[W', b'] = \text{Softmax}(Z_k)$
-

- AutoEncoder Layer 1: Train the first AutoEncoder layer with the input Z_0 to obtain the weight vector parameters W_1 . The output Z_1 represents the features learned from the first layer.
- AutoEncoder Layers 2 to k : Repeat the following steps for each hidden layer, from the second layer (2) to the k -th layer:
 - Take the output Z from the previous layer as input. Train an AutoEncoder to obtain the weight matrix W_i for the current layer, where i represents the layer number.
 - Compute the output Z for the current layer, representing the features learned from this layer.
- Feature Learning: After processing through all k hidden layers, you have W_i weight matrices and Z_i output feature matrices for each layer.
- Fine-Tuning: Refine the weight matrices W_i and bias vectors b_i using Back Propagation. This fine-tuning process is essential to optimize the performance of the SAE.

- Final Weight Matrices and Features: Obtain the optimized weight matrices W_i , bias vectors b_i , and the final feature matrix Z_k .
- Softmax Layer: Apply a Softmax layer to the output of the last hidden layer (Z_k) to obtain the network connection weights W' and bias vector b' . These parameters are used for prediction of MDS path ratings based on the learned features.

In essence, this algorithm guides the training and fine-tuning of a Stacked AutoEncoder to extract meaningful features from MDS path rating data, predict unknown MDS path ratings, ultimately enabling more efficient and informed decision-making in metadata operation routing and other related tasks.

The structure and hyper parameters of Stack AutoEncoder are listed in Table 4.4.

Hyper-Parameters	Values
Activation Function	Tanh() for encoder layers, hidden layers, and softmax for output layer or prediction matrix
Optimizer	Adam with $\gamma = 0.0001$
Loss Function	MSE
Dropout	0.1
Hidden Layers	$k = 3$
Number of nodes	$k1 = 1500, k2 = 500, k3 = 200$

Table 4.4: Hyper-Parameters of the Stacked AutoEncoder

For theoretical proof of our proposed algorithms take an 8x8 metadata servers path rating input matrix mentioned below in table 4.5 having MDS rating from 0-5. The unknown ratings are marked by sign ‘?’.

	MDS1	MDS2	MDS3	MDS4	MDS5	MDS6	MDS7	MDS8
MDS1	0	?	4	?	4	2	4	?
MDS2	?	0	5	?	?	3	2	1
MDS3	4	5	0	4	?	4	?	5
MDS4	?	?	4	0	4	1	?	3
MDS5	4	?	?	4	0	?	1	?

MDS6	2	3	4	1	?	0	3	5
MDS7	4	2	?	?	1	3	0	?
MDS8	?	1	5	3	?	5	?	0

Table 4.5: Path rating matrix with unknown ratings

The unknown path ratings are replaced by 0. The self-path are also represented by zero i.e. $P_{ij} = 0$ where $i = j$ for all path ratings. The input is mentioned in Table 4.6

	MDS1	MDS2	MDS3	MDS4	MDS5	MDS6	MDS7	MDS8
MDS1	0	0	4	0	4	2	4	0
MDS2	0	0	5	0	0	3	2	1
MDS3	4	5	0	4	0	4	0	5
MDS4	0	0	4	0	4	1	0	3
MDS5	4	0	0	4	0	0	1	0
MDS6	2	3	4	1	0	0	3	5
MDS7	4	2	0	0	1	3	0	0
MDS8	0	1	5	3	0	5	0	0

Table 4.6: Input data matrix of 8x8 matrix of MDSs

The training data set is prepared using the Table 4.6. The representation of training data set is mentioned in Table 4.7 created as mentioned below for the above input path rating matrix.

Path Rating information	Training Data Set
(MDS3,MDS1,4)	t1=(0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,4,5,0,4,0,4,0,5,0,0,4,0,4,2,4,0,4)
(MDS5,MDS1,4)	t2=(0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,4,0,0,4,0,0,1,0,0,0,4,0,4,2,4,0,4)
(MDS6,MDS1,2)	t3=(0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,2,3,4,1,0,0,3,5,0,0,4,0,4,2,4,0,2)
(MDS7,MDS1,4)	t4=(0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,4,2,0,0,1,3,0,0,0,0,4,0,4,2,4,0,4)
(MDS3,MDS2,5)	t5=(0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,4,5,0,4,0,4,0,5,0,0,5,0,0,3,2,1,5)

Path Rating information	Training Data Set
(MDS6,MDS2,3)	t6=(0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,2,3,4,1,0,0,3,5,0,0,5,0,0,3,2,1,3)
(MDS7,MDS2,2)	t7=(0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,4,2,0,0,1,3,0,0,0,0,5,0,0,3,2,1,2)
(MDS8,MDS2,1)	t8=(0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,5,3,0,5,0,0,0,0,5,0,0,3,2,1,1)
(MDS1,MDS3,4)	t9=(1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,4,0,4,2,4,0,4,5,0,4,0,4,0,5,4)
(MDS2,MDS3,5)	t10=(0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,5,0,0,3,2,1,4,5,0,4,0,4,0,5,5)
(MDS4,MDS3,4)	t11=(0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,4,0,4,1,0,3,4,5,0,4,0,4,0,5,4)
(MDS6,MDS3,4)	t12=(0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,2,3,4,1,0,0,3,5,4,5,0,4,0,4,0,5,4)
(MDS8,MDS3,5)	t13=(0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,1,5,3,0,5,0,0,4,5,0,4,0,4,0,5,5)
(MDS3,MDS4,4)	t14=(0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,4,5,0,4,0,4,0,5,0,0,4,0,4,1,0,3,4)
(MDS5,MDS4,4)	t15=(0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,4,0,4,0,0,1,0,0,4,0,4,1,0,3,4)
(MDS6,MDS4,1)	t16=(0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,2,3,4,1,0,0,3,5,0,0,4,0,4,1,0,3,1)
(MDS8,MDS4,3)	t17=(0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1,5,3,0,5,0,0,0,0,4,0,4,1,0,3,3)
(MDS1,MDS5,4)	t18=(1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,4,0,4,2,4,0,4,0,0,4,0,0,1,0,4)
(MDS4,MDS5,4)	t19=(0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,4,0,4,1,0,3,4,0,0,4,0,0,1,0,4)
(MDS7,MDS5,1)	t20=(0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,4,2,0,0,1,3,0,0,4,0,0,4,0,0,1,0,1)
(MDS1,MDS6,2)	t21=(1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,4,0,4,2,4,0,2,3,4,1,0,0,3,5,2)
(MDS2,MDS6,3)	t22=(0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,5,0,0,3,2,1,2,3,4,1,0,0,3,5,3)
(MDS3,MDS6,4)	t23=(0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,4,5,0,4,0,4,0,5,2,3,4,1,0,0,3,5,4)
(MDS4,MDS6,1)	t24=(0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,4,0,4,1,0,3,2,3,4,1,0,0,3,5,1)
(MDS7,MDS6,3)	t25=(0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,4,2,0,0,1,3,0,0,2,3,4,1,0,0,3,5,3)
(MDS8,MDS6,5)	t26=(0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,5,3,0,5,0,0,2,3,4,1,0,0,3,5,5)
(MDS1,MDS7,4)	t27=(1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,4,0,4,2,4,0,4,2,0,0,1,3,0,0,4)
(MDS2,MDS7,2)	t28=(0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,5,0,0,3,2,1,4,2,0,0,1,3,0,0,2)
(MDS5,MDS7,1)	t29=(0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,4,0,0,4,0,0,1,0,4,2,0,0,1,3,0,0,1)
(MDS6,MDS7,3)	t30=(0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,2,3,4,1,0,0,3,5,4,2,0,0,1,3,0,0,3)
(MDS2,MDS8,1)	t31=(0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,5,0,0,3,2,1,0,1,5,3,0,5,0,0,1)
(MDS3,MDS8,5)	t32=(0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,4,5,0,4,0,4,0,5,0,1,5,3,0,5,0,0,5)
(MDS4,MDS8,3)	t33=(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,4,0,4,1,0,3,0,1,5,3,0,5,0,0,3)
(MDS6,MDS8,5)	t34=(0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,2,3,4,1,0,0,3,5,0,1,5,3,0,5,0,0,5)

Table 4.7: Training set construction

The unknown ratings are predicted using Stacked AutoEncoder and an output matrix is mentioned below in Table 4.8

	MDS1	MDS2	MDS3	MDS4	MDS5	MDS6	MDS7	MDS8
--	------	------	------	------	------	------	------	------

MDS1	2	3	4	2	4	2	4	3
MDS2	3	3	5	3	4	3	2	1
MDS3	4	5	4	4	3	4	2	5
MDS4	2	3	4	2	4	1	2	3
MDS5	4	4	3	4	3	2	1	3
MDS6	2	3	4	1	2	1	3	5
MDS7	4	2	2	2	1	3	2	5
MDS8	3	1	5	3	3	5	5	4

Table 4.8: Output path rating matrix with predicted path ratings

4.5 Locality Aware Metadata Routing

In this research, a novel approach for enhancing metadata routing, known as LaMeta, is introduced. LaMeta leverages MDS path ratings to make strategic decisions when selecting the next hop in the metadata distribution structure for routing metadata operations. This technique identifies and utilises the top k MDS servers that offer the most optimal path ratings for the next step in the metadata distribution process. The computational steps for determining the locality-aware next k MDSs are outlined in Algorithm 2.

This scientific approach to selecting efficient MDS paths for connecting to other MDS servers can significantly improve metadata storage performance. Additionally, the research provides an expression that quantifies the global metadata query routing locality, denoted as *loc* expression, within the LaMeta framework. This expression allows for a comprehensive assessment of the overall metadata operation routing efficiency, taking into account the strategic choices made based on MDS path ratings.

4.5.1 Locality Aware Next K MDS

The predicted MDS rating matrix is used to create the distribution logical structure of metadata distribution techniques. Each distribution logical structure needs to identify the locally available next k MDS servers, which we can intelligently extract from the predicted MDS rating matrix. The algorithm for Locality Aware Next K MDS for any host is outlined below:

Algorithm 2: Locality Aware Next K MDS

Input: P_{ij} MDS Path Rating matrix, i , and j represent rows and column number or source id and destination id respectively

k number of Top Next MDS hops where $k < j$ and having MDS path rating value equal or greater than threshold path rating r

n number of MDS in matrix

Output: $NextH_{in}$ MDS Next Hop matrix, row i represents the source MDS id and each columns store next MDS id for i^{th} MDS.

1. Initialize matrix $P = SAE()$ with prediction matrix as per Algorithm 1
 2. Set $r_{ij} = 0$ for $i = j$ and $i, j \in [0, n - 1]$. The diagonal paths rating is set to zero to avoid self-NextHop for all
 3. Obtain MDS source list for source MDS s_i , traverse rows of matrix P_{ij} for MDS id having MDS path ratings greater than or equal to threshold path rating r and store the MDS id in $NextH_{im} = j$ where $j \in [0, n - 1]$
 4. Repeat step 3 for $k - 1$ times to get k MDS ids and store these in $NextH_{im}$ vector
 5. Repeat step 3 for all s_i where $i \in [1, n - 1]$ store these MDS ids in $NextH_{im} = j$ where $j \in [0, n - 1]$
 6. Return $NextH_{in}$ Matrix
-

In Algorithm 2 Locality Aware Next K MDS, the output matrix of Stacked AutoEncoder mentioned in table 4.8 is passed to it as input. In this output predicted matrix, the self-path are also again updated with zero rating i.e. $(P_{ij}) = 0$ where $i = j$ for all path ratings. The output matrix thus obtained is mentioned in table 4.9 with predicted rating values. These are presented in Step1 and Step2 of the explanation mentioned below.

Here, we express the mathematical equations and notations for Locality Aware Next K MDS. The notations and mathematical equations for this algorithm are mentioned below:

1. Input:

- P_{ij} : MDS Path Rating matrix, where i and j represent rows and column numbers or source id and destination id, respectively. MDS Next Hop matrix, where row i represents the source MDS id, and each column stores the next MDS id for the i^{th} MDS.
- k : Number of Top next MDS hops where $k < j$ and having MDS path rating value equal to or greater than threshold path rating r .
- n : Number of MDS in the matrix.

Here P_i represent the table 4.8 obtained from the SAE for $n = 8$ as we have taken 8×8 MDSs.

2. Output:

- $NextH_{in}$: MDS vector having top k MDS id having rating greater than r .

3. Procedure:

- **Step1:** By applying Stacked AutoEncoder (SAE) as per Algorithm 1, the prediction matrix P is obtained. P matrix is the output matrix having predicted path ratings.

$$P = SAE()$$

The table 4.8 is obtained from the Algorithm 1: Stacked AutoEncoder (SAE)

- **Step2:** The diagonal paths' ratings to set to zero to avoid self-NextHop for all MDS servers. This step ensures the self-connection do not exist in metadata routing Set $r_{ij} = 0$ for $i = j$ and $i, j \in [0, n - 1]$. The predicted path is presented in Table 4.9

	MDS1	MDS2	MDS3	MDS4	MDS5	MDS6	MDS7	MDS8
MDS1	0	3	4	2	4	2	4	3
MDS2	3	0	5	3	4	3	2	1
MDS3	4	5	0	4	3	4	2	5
MDS4	2	3	4	0	4	1	2	3
MDS5	4	4	3	4	0	2	1	3

MDS6	2	3	4	1	2	0	3	5
MDS7	4	2	2	2	1	3	0	5
MDS8	3	1	5	3	3	5	5	0

Table 4.9: Predicted path rating matrix with 0 diagonal rating value to avoid self-routing.

- **Step3:** For each MDS source s_i , traverse the rows of the matrix P_{ij} . Identify the MDS ids having MDS path ratings greater than or equal to the threshold path rating (r), and store these MDS ids in $NextH_{im} = j$, where $j \in [0, n - 1]$. It will get the MDS IDs with the highest path rating from the threshold.
- **Step4:** Repeat step 3 for $k - 1$ times to collect k MDS ids in the $NextH_{im}$ vector.
- **Step5:** Repeat step 3 for all source MDS servers s_i where $i \in [1, n - 1]$, and store the resulting MDS ids in $NextH_{im} = j$, where $j \in [0, n - 1]$.
- **Step6:** Return the $NextH_{in}$ Matrix, which represents the next MDS hops for each source MDS having path rating greater than threshold r .

We will explain this with the following examples using the Table 4.9 and various values of number of NextHop and threshold rating

For $k = 3$ and $r = 3$: We need to extract maximum 3 NextHop where path rating is greater than or equal to 3 for each MDS. The $NextH_{in}$ obtained and is represented in table 4.10.

MDS ID	Locality Aware Next-Hop IDs for $k = 3$ and $r \geq 3$
MDS1	(MDS3,MDS5,MDS7)
MDS2	(MDS3,MDS5,MDS1)
MDS3	(MDS2,MDS8,MDS1)
MDS4	(MDS3,MDS5,MDS2)
MDS5	(MDS1,MDS2,MDS4)
MDS6	(MDS8,MDS3,MDS3)
MDS7	(MDS8,MDS1,MDS6)

MDS8	(MDS3,MD6,MDS7)
------	-----------------

Table 4.10: Locality Aware Next-Hop IDs for $k = 3$ and $r \geq 3$

For $k = 2$ and $r = 4$: We need to extract maximum 2 NextHop where path rating is greater than or equal to 4 for each MDS. The *NextH_in* obtained and is represented in table 4.11.

MDS ID	Locality Aware Next-Hop IDs for $k = 2$ and $r \geq 4$
MDS1	(MDS3,MDS5)
MDS2	(MDS3,MDS5)
MDS3	(MDS2,MDS8)
MDS4	(MDS3,MDS5)
MDS5	(MDS1,MDS2)
MDS6	(MDS8,MDS3)
MDS7	(MDS8,MDS1)
MDS8	(MDS3,MD6)

Table 4.11: Locality Aware Next-Hop IDs for $k = 2$ and $r \geq 4$

Prior to LaMeta technique, the MDS connects to random NextHop and do not consider path latencies in MDS distribution network and also do not have capability to avoid or distinguish the use slow and high latency paths. To address this issue, our proposed locality aware next K MDS scientifically gets best MDS paths for MDS distribution network routing. The metadata distribution techniques follow various distribution techniques. The result of these judiciously locality aware NextHops in very large distributed storage will have performance impact on metadata query routing for metadata storage retrieval. The following chapter provides a detailed discussion of the outcomes obtained from employing these distribution techniques across different number of MDSs. The LaMeta-based metadata management within a storage system enhances operational efficiency and proves crucial, particularly in applications sensitive to storage latency.

In summary, the algorithm employs the MDS path rating matrix and the SAE model to identify the next k MDS hops with favourable path ratings for metadata operation routing. It accounts for both self-paths and paths to other MDS servers, ultimately improving the efficiency of metadata operation routing in a distributed storage system.

4.6 Locality Expression

In a system where metadata is distributed across N metadata servers, each metadata server is uniquely identified as $\{n_i \mid 0 \leq i \leq N - 1\}$. The "hops," denoted by the variable h , which signifies the number of MDS paths that must be traversed to process a metadata operation. It is essential to understand that there are two primary ways to handle metadata operations routing:

1. Direct Processing ($h = 0$): In this case, the metadata operation is executed directly on the MDS server where the metadata is originally stored, requiring no additional hops.
2. Routing to Other MDS Servers ($h > 0$): Here, the metadata operation is routed to other MDS servers, and it may involve multiple hops to reach its destination. Such hops represented by variable h it has value in the range of ($h = \{0 < h_i \leq N - 1\}$)

The locality function loc expresses the aggregate effect intermediate of MDS path ratings. By combining these ratings effectively, we can create a representation that reflects the locality efficiency of metadata query routing within the system. This proposed loc expression allows us to measure the locality and compare the performance enhancement of metadata operations in a distributed storage system.

Definition 1. Defines locality loc of metadata operation routing in value of ultra-large distributed storage system, which can be expressed as

$$loc = \begin{cases} \sum_{i=1}^h \frac{i \times (r_{isd} - r_{\max} - 1)}{(r_{\max} \times r_{isd})}, & \forall s \neq d, \\ 0, & \forall s = d, \end{cases} \quad (4.1)$$

In our Eq. 4.1 the output value of $loc \{loc < 0 \mid \forall s \neq d \text{ and } loc = 0 \mid \forall s = d \text{ and } r_{isd} = 0\}$. $r_{isd} = 0$ for all $s = d$ represent self-loop.

The function represented as loc , calculates a value that represents the locality of metadata query routing. loc is defined as a mathematical expression that depends on several variables and conditions. The variable i is a counter that iterates through a range of values from 1 to h . h is a parameter that represents the number of hops or MDS paths traversed in the metadata query routing. r_{isd} represents the MDS path rating for a specific combination of source s and destination d for the i^{th} hop. In other words, it is the rating associated with a particular step in the routing process.

r_{max} is a constant that represents the maximum possible MDS path rating. It serves as a reference point for the ratings. Here $r_{max} = 5$ is the maximum rating value. Now, there are two cases within the function:

1. Case for $s \neq d$: In this case, the function calculates a sum over the range of hops i from 1 to h . For each hop, it computes a term using the loc function and then added together. This case essentially accumulates impact of intermediate path rating values for different hops when the source s is not equal to the destination d .
2. Case for $s = d$: In this case, the function simply returns 0. This means that when the source and destination are the same i.e., the metadata operation is processed on the same MDS without routing. So the locality value is 0 which is the highest.

The overall purpose of this function is to assess the locality of metadata query routing based on the MDS path ratings. It calculates a value that reflects how efficiently metadata queries are routed, taking into account the ratings associated with each step in the routing process. This value can help in optimizing metadata query routing strategies in a distributed storage system, aiming for higher locality and improved performance.

The h value plays a significant role in determining locality. For instance, when considering a scenario with h equal to 4 and intermediate path ratings r_{isd} of 3, 4, 5, and 2, the resulting locality loc is calculated as -2.12 . This calculation involves subtracting various fractions ($-0.20, -0.20, -0.12, \text{ and } -1.6$) from each other.

As the number of hops h increases, the locality loc will tend to decrease. The specific value of loc is relative and depends on the ratings assigned to the intermediate MDS path values. A higher value of locality loc indicates that metadata operations are being accessed from MDS servers that are closer in proximity. In other words, it signifies that the metadata queries are executed more efficient in terms of routing, which can be advantageous for improving performance in a distributed storage system.

4.7 Conclusion

In this chapter, we delve into the design considerations of the LaMeta technique tailored for ultra-large-scale storage systems. Our exploration encompasses the architectural framework of the proposed Stacked AutoEncoder, the concept of locality-aware next K MDS, and the algorithms associated with them. Additionally, we shed light on the process of translating MDS path RTT values into MDS path ratings, along with the preprocessing of input data using one-hot encoding, a crucial aspect specific to the Stacked AutoEncoder.

Furthermore, our chapter extensively discusses the novel concept of "locality expression", denoted as loc , in the context of metadata operation routing within the expansive landscape of ultra-large distributed storage systems. This comprehensive exploration seeks to provide valuable insights into the design and functionality of the LaMeta technique, tailored to enhance the performance and efficiency of large-scale storage systems.

The upcoming chapter will investigate the performance evaluation of LaMeta-based metadata distribution techniques. To begin, the chapter will focus on evaluating the performance of collaborative filtering (CF)-based recommender systems and the Stacked AutoEncoder. Additionally, it will examine the performance evaluation of LaMeta-based metadata distribution techniques. Furthermore, this chapter will present the outcomes of performance assessments regarding the impact of LaMeta techniques on four contemporary metadata distribution techniques: Static Subtree, Dynamic Subtree, Hash-Based, and DROP. The performance of locality-aware metadata routing is evaluated using average locality loc gathered at particular numbers of hops. The results of av-

erage locality achieved using different metadata distribution techniques with LaMeta and its corresponding baseline technique. Through these evaluations, we aim to provide comprehensive results of the effectiveness of LaMeta in enhancing metadata distribution techniques.

Chapter 5

Validation of the Proposed LaMeta-based Techniques

The previous chapter explores the locality and presented the design of the proposed LaMeta technique and its constituents tailored for ultra-large-scale storage systems. Our discussion encompasses the proposed Stacked AutoEncoder structure, the locality-aware next K MDS, and their associated algorithms. We also shed light on translating MDS path RTT values into MDS path ratings, along with the preprocessing of input data using one-hot encoding, a crucial aspect specific to the Stacked AutoEncoder. Furthermore, the previous chapter also discusses the "locality expression", denoted as loc , in the context of metadata operation routing within very large distributed storage systems.

This chapter will validate the proposed LaMeta technique user four static subtree, dynamic subtree, hash-based and DROP-based metadata distribution techniques by performing comparative analysis on various scales of MDSs. First, we will evaluate the performance of the recommendation system based on comparing the prediction accuracy of inferred ratings. As a result, the confusion matrix is presented and discussed. We will evaluate the performance of proposed LaMeta-based techniques using three real datasets and four contemporary scalable metadata distribution techniques. We will evaluate the performance in terms of aggregate throughput of metadata distribution techniques. The experimental results show the average percentage increase in aggregate throughput of LaMeta-based metadata techniques using all three datasets. The performance of locality-aware metadata routing will be evaluated using average locality loc gathered at particular numbers of hops.

The chapter delves into experiments conducted to validate the LaMeta Technique in Section 5.1. Initially, the performance of CF-based recommendation techniques is assessed

using two datasets, and the results are detailed in Section 5.2. Sections 5.3 and 5.4 of this chapter present the performance evaluation of LaMeta-based distribution techniques, focusing on two key metrics: the aggregate throughput of metadata operations per second and locality. Four baseline contemporary metadata distribution techniques—Static Subtree, Dynamic Subtree, Hash-Based, and DROP—are employed in these experiments. Lastly, Section 5.5 provides the results of the performance of locality-aware metadata routing, as assessed by measuring average locality (*loc*) at specific numbers of hops. The section 6 concludes the chapter.

5.1 Validation of LaMeta-Based Technique

The validation of the LaMeta technique involves a series of experiments that compare it with baseline metadata distribution techniques. Initially, we assess the performance of MF-based recommendation techniques by measuring their prediction accuracy on test datasets. This evaluation involves the construction and analysis of a confusion matrix to gain insights into the technique’s performance.

Subsequently, we conduct experiments to evaluate the proposed LaMeta-based techniques in terms of aggregate throughput. These experiments are carried out using three real datasets and four contemporary scalable metadata distribution techniques. The goal is to assess the performance of LaMeta-based approaches, including static subtree, dynamic subtree, hash-based, and DROP-based techniques, by comparing them with their respective baseline distribution techniques across various scales of MDSs. To measure throughput, we quantify the number of operations per second in thousands on different scales of MDSs.

The experimental results also include the presentation of the average percentage increase in aggregate throughput achieved by LaMeta-based metadata techniques across all three datasets. This analysis provides insights into the performance improvements facilitated by LaMeta.

Finally, the evaluation extends to the performance of locality-aware metadata routing. This assessment involves calculating the average locality denoted as "loc" at specific hop counts, allowing us to gauge the effectiveness of locality-aware metadata routing in distributed metadata management techniques.

The experiments were carried out on an Ubuntu 20.04 server with specific hardware specifications. The server was equipped with a 12-core Intel Xeon Silver 4116 processor running at 2.1GHz, an 8 GB GDDR5 GPU, 128 GB of DDR4 RAM, and eight 1.2TB 10K RPM SAS hard drives with a 12Gbps interface. For the implementation and evaluation of the experiments, we utilized Python as the primary programming language. In addition to Python, we employed various allied libraries and leveraged TensorFlow, a popular

machine-learning framework, to facilitate the experiments.

5.2 Performance of MF based recommendation techniques

The performance assessment of the MF recommendation system relies on the comparison of predicted ratings on the test data. We employed two separate datasets, DS-I and DS-II, the details of which are already furnished in Table 4.2 within section 4.3. The RTT values contained in these datasets have been converted into ratings, as outlined in Table 4.3 under chapter 4. The learning rate and regularization parameters for MF techniques, including SVD, SVD++, and RNMF, are presented in Table 5.1.

MF Technique	Learning Rate γ	Regularisation term λ
SVD	0.005	0.02
SVD++	0.005	0.02
RNMF	0.005	$\lambda_s = 0.006, \lambda_d = 0.006$

Table 5.1: Hyper-parameter of SGD used in SVD, SVD++, RNMF

The hyper-parameters employed for configuring the Stacked AutoEncoder have been previously detailed in Table 4.4, which can be referenced in Chapter 4. In the context of evaluating the recommendation system's performance, the focus is on predicting unknown MDS path ratings, and the results are analyzed using confusion matrices.

Confusion matrices are utilized to present and analyze the performance of the recommendation system. These matrices have rows and columns, where the rows correspond to actual MDS path ratings, and the columns represent the predicted MDS path ratings. The diagonal elements within each cell of the matrix indicate the percentage of accurate predictions, while the non-diagonal cells indicate percentages of confusion.

The Tables 5.2, 5.3, 5.4, 5.5, 5.8, 5.9, 5.10 and Table 5.11, Present the confusion matrices generated by the various recommendation techniques, including SVD, SVD++, RNMF, and SAE, using both datasets DS-I and DS-II. These matrices offer a comprehensive view of the performance of the recommendation system in terms of its ability to predict MDS path ratings accurately and identify instances of confusion.

SVD DS-I						
		Predicted MDS Path Ratings				
		1	2	3	4	5
Actual MDS Path Ratings	1	64.2	35.8	0	0	0
	2	0	89.2	10.8	0	0
	3	0	0.23	90.6	9.17	0
	4	0	0	9.77	90.1	0.13
	5	0	0	0.21	43.6	56.19

Table 5.2: Confusion Matrix of SVD on DS-I

SVD++ DS-I						
		Predicted MDS Path Ratings				
		1	2	3	4	5
Actual MDS Path Ratings	1	53.5	37.5	9	0	0
	2	0	73.5	26.5	0	0
	3	0	0	93.4	6.6	0
	4	0	0	10.8	89	0.2
	5	0	0	973	4.87	85.4

Table 5.3: Confusion Matrix of SVD++ on DS-I

RNMF DS-I						
		Predicted MDS Path Ratings				
		1	2	3	4	5
Actual MDS Path Ratings	1	93.7	4.28	2.02	0	0
	2	0	91.1	5.36	3.54	0
	3	0	0	88.2	11.8	0
	4	0	0	9.49	85.4	5.11
	5	0	0.21	0.49	10.1	89.2

Table 5.4: Confusion Matrix of RNMF on DS-I

SAE DS-I						
		Predicted MDS Path Ratings				
		1	2	3	4	5
Actual MDS Path Ratings	1	96.62	3.18	0.20	0	0
	2	1.83	93.5	4.67		0
	3	0	2.71	94.8	2.49	0
	4	0	0	1.52	96.2	2.28
	5	0	0	0	4.31	95.69

Table 5.5: Confusion Matrix of SAE on DS-I

The confusion matrices of SVD, SVD++, RNMF and SAE on data set DS-I have multiple classes, so, we have converted the multiclass confusion matrices to binary class confusion matrices and then calculated the True Positive (TP), True Negative (TN), False Positive

Technique	Class	TP	FP	FN	TN
SVD DS-I	1	64.2	0	35.8	546.1
	2	89.2	10.8	0	546.1
	3	90.6	9.4	9.4	546.1
	4	90.1	9.9	9.9	546.1
	5	56.2	43.8	0.2	546.1
SVD++ DS-I	1	53.5	0	37.5	646.1
	2	73.5	26.5	0	646.1
	3	93.4	6.6	0	646.1
	4	89	10.8	1	646.1
	5	85.4	4.6	0.2	646.1
RNMF DS-I	1	93.7	0	6.3	391.459
	2	91.1	5.439	8.9	386.02
	3	88.2	7.87	11.8	383.589
	4	85.4	25.44	6.059	374.56
	5	89.2	5.11	10.8	386.349
SAE DS-I	1	96.62	1.83	3.38	397.17
	2	93.5	6.4	6.5	392.6
	3	94.8	6.28	5.2	392.72
	4	96.2	5.4	2.79	394.61
	5	95.69	2.28	4.32	396.71

Table 5.6: True positive, false positive, false negative and true negative values for each binary class of confusion matrices on DS-I.

(FP) and False Negative (FN) and mentioned in Table 5.6. Based on this data, we computed metrics of each binary class of confusion matrices, such as accuracy, precision, recall, and F1 Score, for each class rating and mentioned in Table 5.7. The definitions of these metrics are defined below:

- **Binary Classes of Confusion Matrix:** For each class i , A binary classification where class i is the positive class and all other classes are the negative classes.
- **Definitions:**
 - True Positive (TP): The number of correctly predicted positive instances.
 - True Negative (TN): The number of correctly predicted negative instances.
 - False Positive (FP): The number of incorrectly predicted positive instances.
 - False Negative (FN): The number of incorrectly predicted negative instances.
- Calculate Metrics:

– **Accuracy:**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

– **Precision:**

$$\text{Precision} = \frac{TP}{TP + FP}$$

– **Recall:**

$$\text{Recall} = \frac{TP}{TP + FN}$$

– **F1 Score:**

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Technique	Class	Accuracy	Precision	Recall	F1 Score
SVD DS-I	1	0.945	1	0.642	0.782
	2	0.973	0.892	1	0.942
	3	0.928	0.777	0.906	0.837
	4	0.946	0.901	0.901	0.901
	5	0.906	0.562	0.996	0.719
SVD++ DS-I	1	0.946	1	0.535	0.697
	2	0.946	0.735	1	0.847
	3	0.984	0.934	1	0.966
	4	0.982	0.891	0.989	0.937
	5	0.974	0.997	0.854	0.92
RNMF DS-I	1	0.987	1	0.937	0.967
	2	0.971	0.944	0.911	0.927
	3	0.96	0.918	0.882	0.9
	4	0.936	0.77	0.934	0.844
	5	0.968	0.946	0.892	0.918
SAE DS-I	1	0.99	0.981	0.966	0.974
	2	0.974	0.936	0.935	0.935
	3	0.977	0.938	0.948	0.943
	4	0.984	0.947	0.972	0.959
	5	0.987	0.977	0.957	0.967

Table 5.7: Accuracy, precision, recall and F1 score for each binary class of confusion matrices on DS-I.

The confusion matrices of SVD, SVD++, RNMF and SAE on data set DS-II have multiple classes, so, we have converted the multiclass confusion matrices to binary class confusion matrices and then calculated the True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) and mentioned in Table 5.12. Based on this data, we

SVD DS-II						
		Predicted MDS Path Ratings				
		1	2	3	4	5
Actual MDS Path Ratings	1	86.1	13.9	0	0	0
	2	3.34	96.5	0.16	0	0
	3	0	0	42.3	57.7	0
	4	0	0	0.52	97.4	2.08
	5	0	0	0	8.8	91.2

Table 5.8: Confusion Matrix of SVD on DS-II

SVD++ DS-II						
		Predicted MDS Path Ratings				
		1	2	3	4	5
Actual MDS Path Ratings	1	57.8	28.9	13.3	0	0
	2	8.61	82.8	8.59	0	0
	3	0	0	41.2	58.8	0
	4	0	0	0	94.3	5.7
	5	0	0	0	30.7	69.3

Table 5.9: Confusion Matrix of SVD++ on DS-II

computed metrics of each binary class of confusion matrices, such as accuracy, precision, recall, and F1 Score, for each class rating and mentioned in Table 5.13.

The Stacked AutoEncoder, which has better accuracy and scalability in previous experiments, was integrated into our proposed LaMeta technique to predict unknown path ratings of Metadata Servers (MDSs). In our experiments involving LaMeta, we exclusively employed a dataset referred to as DS-II. Our objective was to thoroughly assess the effectiveness of LaMeta within the context of distributed metadata management techniques. To accomplish this, we proceeded with the implementation of these techniques and conducted a rigorous evaluation of their performance across varying numbers of

RNMF DS-II						
		Predicted MDS Path Ratings				
		1	2	3	4	5
Actual MDS Path Ratings	1	91.1	8.34	0.56	0	0
	2	3.24	89.5	7.26	0	0
	3	0	3.93	88.8	7.27	0
	4	0	0	6.58	89.6	3.82
	5	0	0	2.94	9.83	87.23

Table 5.10: Confusion Matrix of RNMF on DS-II

SAE DS-II						
		Predicted MDS Path Ratings				
		1	2	3	4	5
Actual MDS Path Ratings	1	93.3	6.49	0.21	0	0
	2	2.06	95.7	2.24	0	0
	3	0	2.06	94.8	3.14	0
	4	0	0	1.08	96.6	2.32
	5	0	0	1.94	3.83	94.23

Table 5.11: Confusion Matrix of SAE on DS-II

Technique	Class	TP	FP	FN	TN
SVD DS-II	1	86.1	3.34	13.9	389.25
	2	96.5	14.32	3.5	378.27
	3	42.3	0.68	58.12	391.49
	4	97.4	58.67	2.6	333.92
	5	91.2	2.08	0.97	398.34
SVD++ DS-II	1	57.8	8.61	42.2	391.6
	2	82.8	29.31	17.2	370.9
	3	41.2	22.97	59.31	376.73
	4	94.3	89.6	5.4	310.91
	5	69.3	4.32	30.7	395.89
RNMF DS-II	1	91.1	3.24	8.9	381.86
	2	89.5	12.22	10.5	372.88
	3	88.8	17.34	5.15	373.81
	4	89.6	2.25	10.4	382.85
	5	87.23	3.82	3.92	390.13
SAE DS-II	1	93.3	2.06	6.62	397.86
	2	95.7	8.73	4.3	391.11
	3	94.6	5.39	5.32	394.53
	4	96.6	6.91	3.4	392.93
	5	94.23	2.32	5.77	397.52

Table 5.12: True positive, false positive, false negative and true negative values for each binary class of confusion matrices on DS-II.

Technique	Class	Accuracy	Precision	Recall	F1 Score
SVD DS-II	1	0.966	0.86	0.966	0.91
	2	0.966	0.97	0.873	0.919
	3	0.882	0.42	0.977	0.587
	4	0.86	0.97	0.591	0.735
	5	0.978	0.91	0.978	0.943
SVD++ DS-II	1	0.898	0.58	0.866	0.695
	2	0.906	0.822	0.741	0.779
	3	0.82	0.41	0.569	0.477
	4	0.792	0.86	0.489	0.623
	5	0.928	0.69	0.932	0.793
RNMF DS-II	1	0.976	0.91	0.968	0.938
	2	0.956	0.9	0.882	0.891
	3	0.942	0.89	0.832	0.86
	4	0.944	0.891	0.841	0.865
	5	0.966	0.87	0.956	0.911
SAE DS-II	1	0.984	0.939	0.979	0.959
	2	0.976	0.96	0.923	0.941
	3	0.98	0.95	0.95	0.95
	4	0.98	0.97	0.933	0.951
	5	0.984	0.94	0.979	0.959

Table 5.13: Accuracy, precision, recall and F1 score for each binary class of confusion matrices on DS-II.

MDSs. These experiments were carried out on institute servers and clients, all running on the Ubuntu 20.04 LTS operating system. The experimental setup encompassed a wide range of scales, ranging from 5 to 50 metadata servers, allowing us to comprehensively evaluate and analyze the performance of the LaMeta-based metadata distribution techniques under varying scale.

5.3 Performance Evaluation of LaMeta Based Metadata Distribution Techniques

To comprehensively assess the performance of LaMeta-based metadata distribution techniques, we conducted a performance evaluation focusing on two key aspects: aggregate throughput of metadata operations per second and locality. In order to conduct these performance evaluations, we employed three real datasets of metadata, as detailed in Table 5.14, and four contemporary metadata distribution techniques, namely Static Subtree,

Dynamic Subtree, Hash-Based, and DROP, as part of our experiments.

To gauge the effectiveness of LaMeta-based techniques, we compared their results with those of the baseline techniques mentioned above. In this evaluation, we established a setup involving a single server responsible for continuously generating and dispatching batches of query workloads to the Metadata Servers (MDSs), while concurrently collecting performance evaluation data for all the techniques under consideration.

Trace Source	Data Size	Metadata Counts (millions)
LiveMap Dataset	15.1 GB	44.75
DevTools Dataset	5.90 GB	17.95
MSN Storage dataset	5.77 GB	26.65

Table 5.14: Data Sources used in Experiments

Each batch of workload consisted of two thousand queries designed to be executed on the distributed metadata system. We executed multiple batches of operations in parallel to ensure robust data collection for performance assessment. This approach allowed us to thoroughly analyze and compare the performance of LaMeta-based techniques against their baseline counterparts.

5.4 MDS Scale and Performance Comparison

To assess the performance and scalability of our LaMeta-based metadata distribution technique, we primarily focused on two key metrics: locality improvement and aggregate throughput. We conducted these evaluations in conjunction with contemporary metadata distribution techniques by distributing metadata operations across Metadata Servers (MDSs) and measuring the operations executed per second across varying scales of MDSs. In all experiments, the scale of MDS varies from 5 to 50 in numbers.

To evaluate the impact of LaMeta metadata management techniques on performance, we applied the LaMeta techniques on top of the baseline metadata distribution techniques. Our performance evaluation was based on measuring aggregate throughput, expressed in thousands of metadata operations, across a range of MDS scales. The results of these evaluations are graphically presented in Fig. 5.1-5.12, utilizing data from three real datasets. These figures effectively illustrate the performance impact of LaMeta-based

techniques in comparison to the baseline techniques. Furthermore, Fig. 5.13. presents the percentage increase in average aggregate throughput across all three datasets, providing a clear picture of the performance enhancements achieved through the implementation of LaMeta-based metadata distribution techniques.

5.4.1 LaMeta Based Static-Subtree Partitioning

The Fig. 5.1, Fig. 5.5, and Fig. 5.9 illustrate the throughput performance of Static Subtree Partition and LaMeta-based Static Subtree Partition techniques using LiveMap, DevTools, and MSN datasets. This discussion focuses on the percentage increase in throughput at different scales (number of Metadata Servers, or MDSs) for each dataset. While analyzing a smaller Metadata Server (MDS) network consisting of 5 to 20 MDSs, it is evident that relatively fewer metadata operations require multiple hops for routing. Within this MDS scale range (5 to 20), the average improvement in aggregate throughput achieved by our proposed LaMeta-based static subtree partitioning technique ranges from nearly 0.9 thousand to 2.8 thousand operations per second on all three datasets compared to baseline techniques which is 2% to 5% at 5 MDs scale and 9% to 12% at 20 MDS scale. This indicates that, at smaller scales, the performance gains are somewhat moderate.

However, as we transition to a larger MDS network with 25 to 50 MDSs, the LaMeta-based static subtree metadata distribution technique demonstrates notably improved performance compared to the baseline static subtree counterpart. This improvement is attributed to the increasing number of metadata operations that necessitate metadata query routing from one MDS to another within a larger network. In this MDS scale range (25 to 50), the average performance gains in terms of aggregate throughput achieved by our proposed LaMeta-based static subtree partitioning technique range from 2.2 to 4.0 thousand operations per second on all three datasets compared to baseline techniques, which is a percentage increase of nearly 12% to 14%. This substantial improvement underscores the positive impact of our proposed locality-aware routing on overall system performance. These results highlight the effectiveness of LaMeta in optimizing metadata distribution and improving the overall system throughput of Static-Subtree Partitioning techniques,

particularly in mid-range and larger configurations. The consistent performance gains across different datasets demonstrate the versatility and robustness of LaMeta on static subtree metadata distribution techniques in various operational environments.

5.4.2 LaMeta Based Dynamic Subtree Partitioning

As illustrated in Fig. 5.2, Fig. 5.6, and Fig. 5.10, when examining a smaller scale of Metadata Servers (MDS), specifically ranging from 5 to 20 MDSs, it becomes apparent that the LaMeta-based dynamic subtree metadata distribution technique exhibits a modest improvement over the baseline dynamic subtree metadata distribution technique across all three datasets. This improvement is less pronounced when fewer operations necessitate MDS routing due to the small scale of MDS. As the MDS scale increases, the percentage gains have also shown good gains.

The performance gain achieved by the proposed LaMeta-based dynamic subtree metadata distribution technique varies within the range of 4% to 9% at 5 MDS scale across all three datasets during this smaller scale. However, as we transition to larger MDS scales, ranging from 25 to 50, the LaMeta-based dynamic subtree partitioning technique demonstrates a notable performance boost, ranging from 10% to 20% in terms of aggregate metadata operations in mid-scale of 10 to 30 MDSs and higher than 20% at higher scale of MDSs ranging from 40 to 50. This gain in performance is especially significant when considering the scalability demands of a distributed storage system. It is important to note that the aggregate throughput of metadata operations in dynamic subtree partitioning exhibits some unevenness and does not scale as effectively due to the inherent dynamics of metadata migration during its operations.

5.4.3 LaMeta Based Hash Partitioning

Hashing techniques have evolved to effectively address load balancing in metadata distribution and improve the efficiency of metadata operations. Fig. 5.3, Fig. 5.7, and Fig. 5.11 present the throughput performance of standard Hash Base Partition and LaMeta-based Hash Base Partition techniques using datasets LiveMap, DevTools, and MSN. At a smaller scale involving 5 to 20 Metadata Servers (MDSs), the LaMeta-based hash-

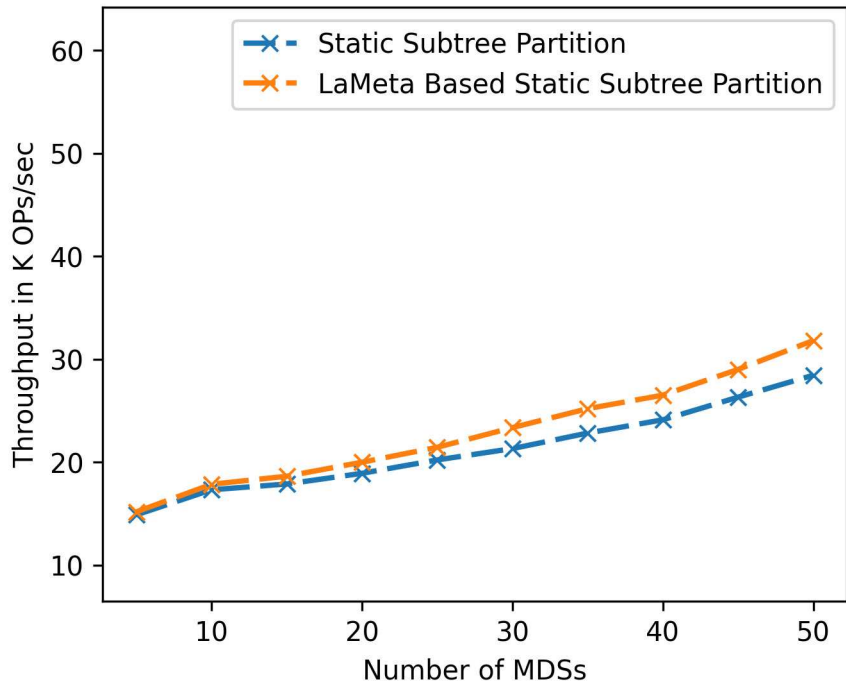


Figure 5.1: Throughput of LaMeta based static subtree metadata distribution techniques using LiveMap dataset.

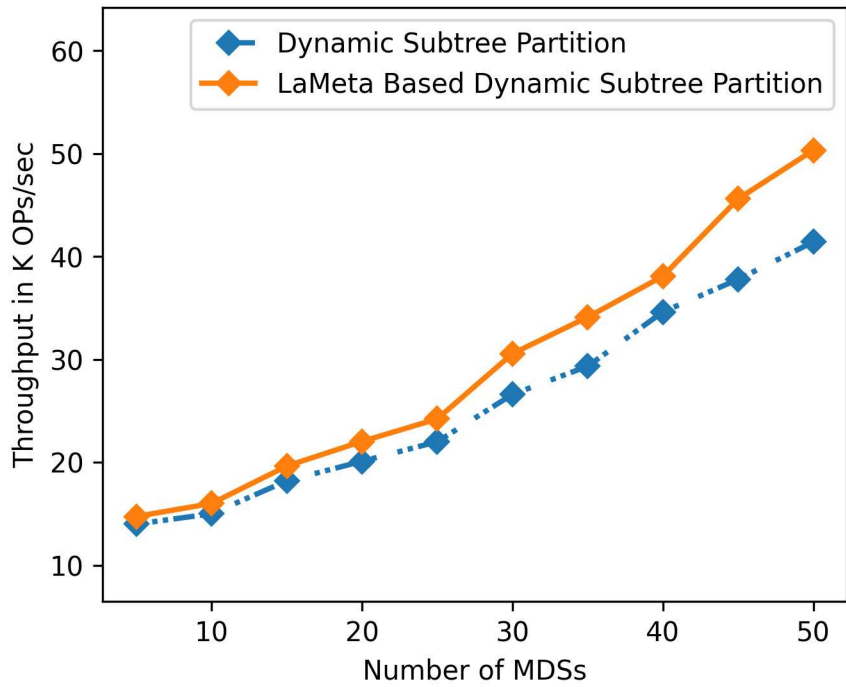


Figure 5.2: Throughput of LaMeta based Dynamic subtree metadata distribution techniques using LiveMap dataset.

based metadata partitioning technique exhibits a modest gain of throughput ranging from nearly 0.15 thousand to 2.9 thousand operations per second in all three datasets when

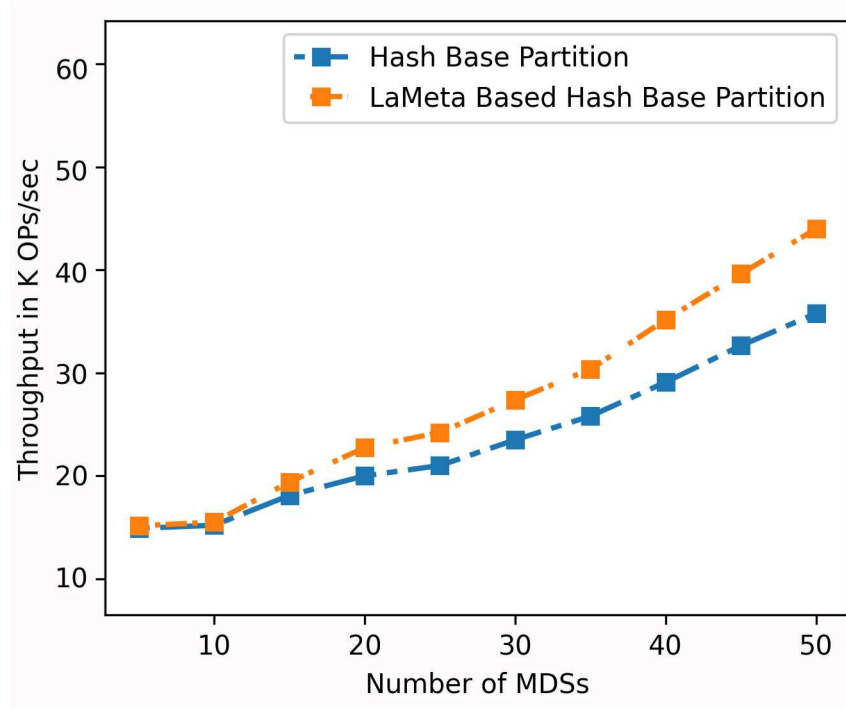


Figure 5.3: Throughput of LaMeta Hash based metadata distribution techniques using LiveMap dataset.

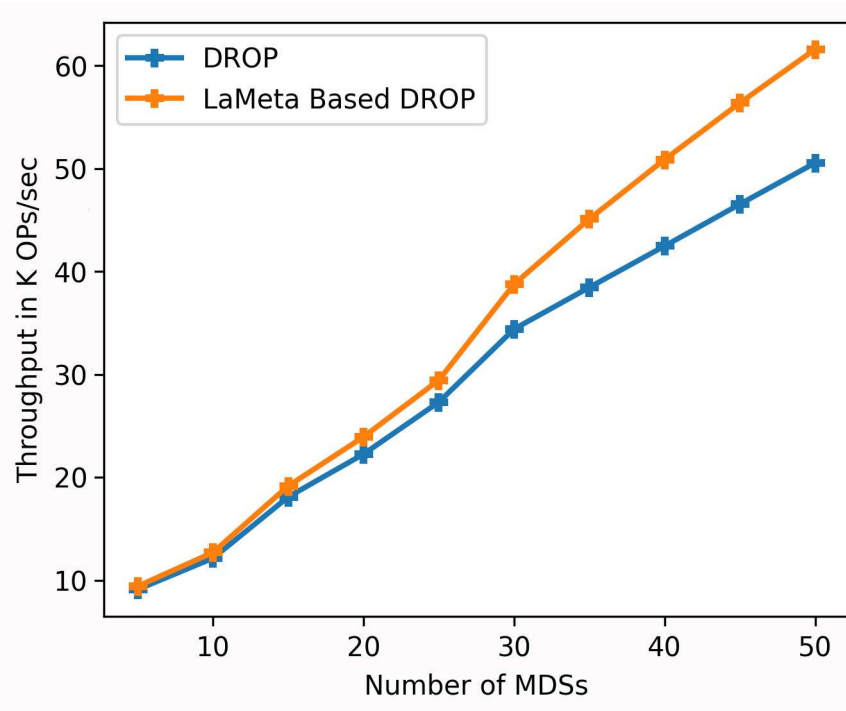


Figure 5.4: Throughput of LaMeta based DROP metadata distribution techniques using LiveMap dataset.

compared to the baseline hash-based metadata distribution technique. The LaMeta-based hash-based metadata partitioning technique exhibits a modest gain of nearly 1% to 9%

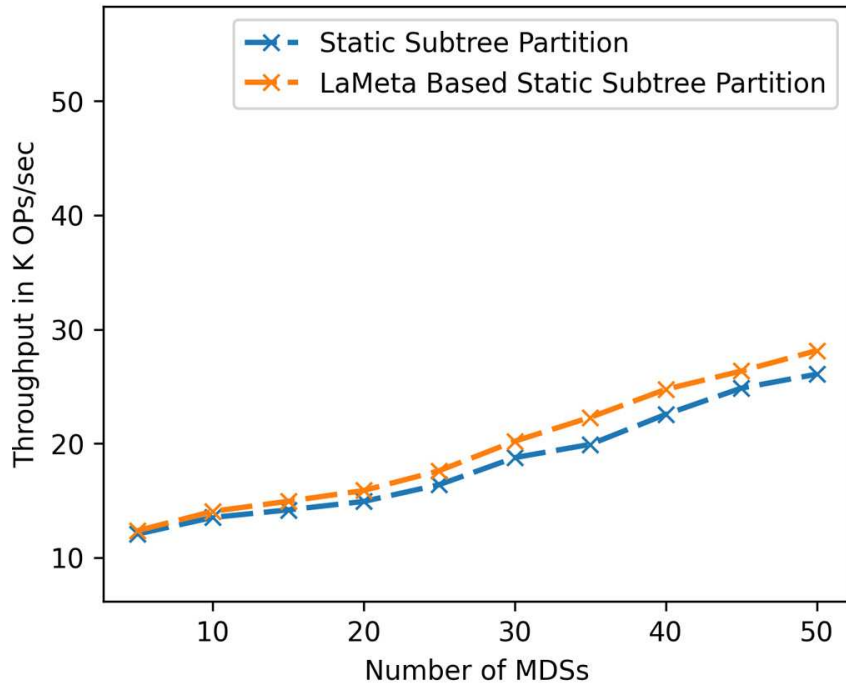


Figure 5.5: Throughput of LaMeta based Static Subtree metadata distribution techniques using DevTools dataset

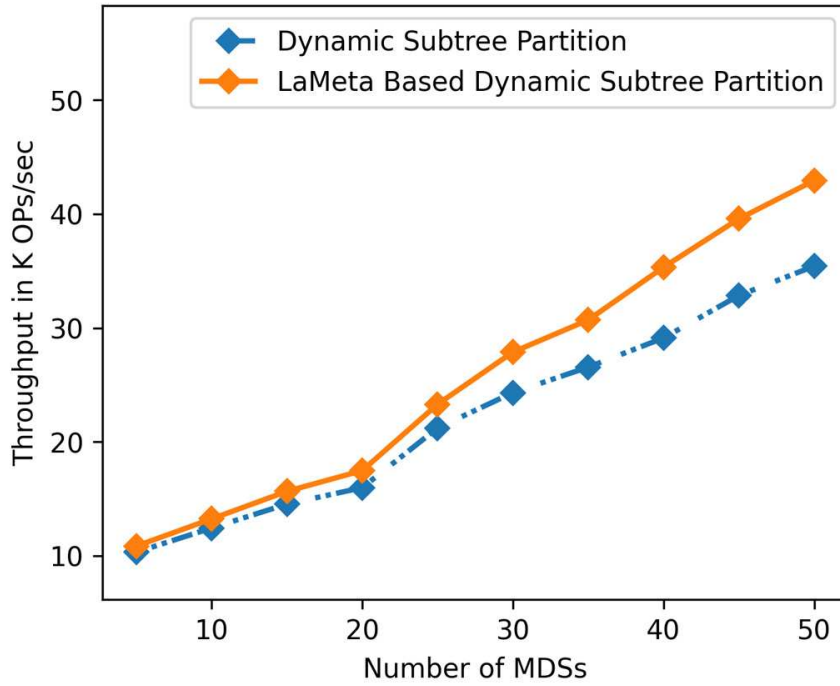


Figure 5.6: Throughput of LaMeta based Dynamic Subtree metadata distribution techniques using DevTools dataset

at MDSs scale of 5 and 20. This indicates that, at smaller scales, the improvement in metadata throughput is somewhat moderate.

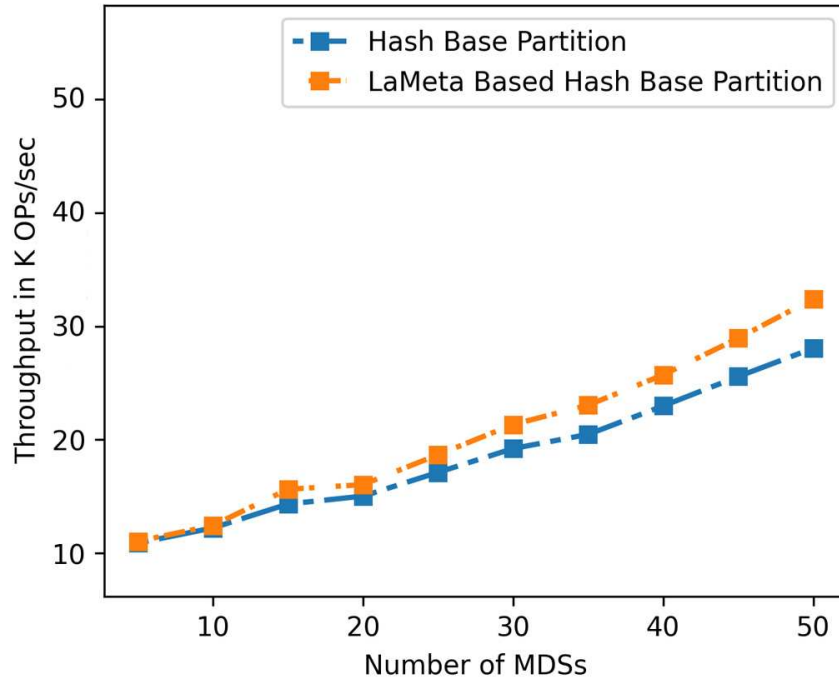


Figure 5.7: Throughput of LaMeta Hash based metadata distribution techniques using DevTools dataset.

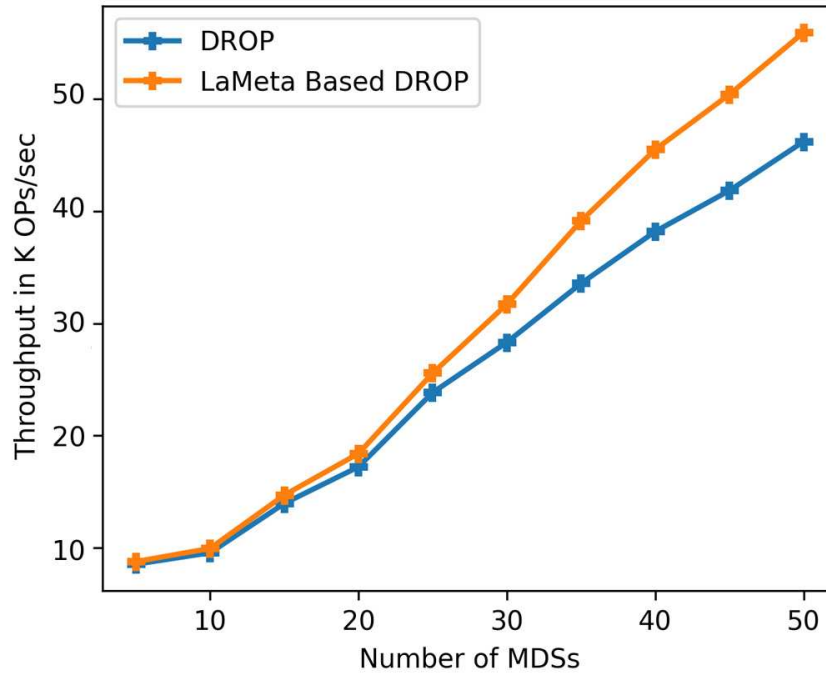


Figure 5.8: Throughput of LaMeta based DROP metadata distribution techniques using DevTools dataset.

However, as we move to a larger MDS network scale of 25 to 40, the LaMeta Hash-based Partitioning technique demonstrates a more substantial performance gain, ranging from

9% to 19% in terms of metadata operations. Hashing techniques offer greater independence in managing parent-child relationships, and this independence becomes particularly effective in scenarios involving a larger number of MDSs. The LaMeta optimizes metadata routing and distribution across the network with 24% gain in throughput of operation per second at 50 MDSs scale.

These results highlight the effectiveness of LaMeta in optimizing metadata distribution and improving the overall system throughput of the LaMeta Hash-based Partitioning technique, particularly in mid-range and larger MDS networks. The consistent performance gains across different datasets demonstrate the versatility and robustness of LaMeta on Hash-based Partitioning technique in various operational environments.

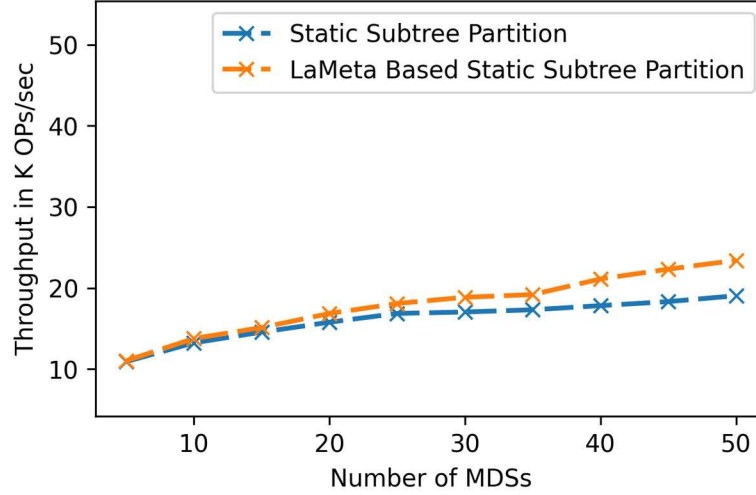


Figure 5.9: Throughput of LaMeta based Static subtree metadata distribution techniques using MSN dataset.

5.4.4 LaMeta Based Dynamic Ring Online Partitioning (DROP)

In Fig. 5.4, 5.8 and 5.12 illustrate the throughput performance of standard DROP and LaMeta-based DROP metadata distribution techniques using LiveMap, DevTools, and MSN datasets. On examining a smaller MDS network scale ranging from 5 to 20 Metadata Servers (MDSs), the LaMeta-based DROP technique shows only a slight gain in throughput, ranging from 2.5% to 5% at 5 MDSs Servers across all three datasets, when compared to the baseline DROP technique. The difference is not very pronounced at this

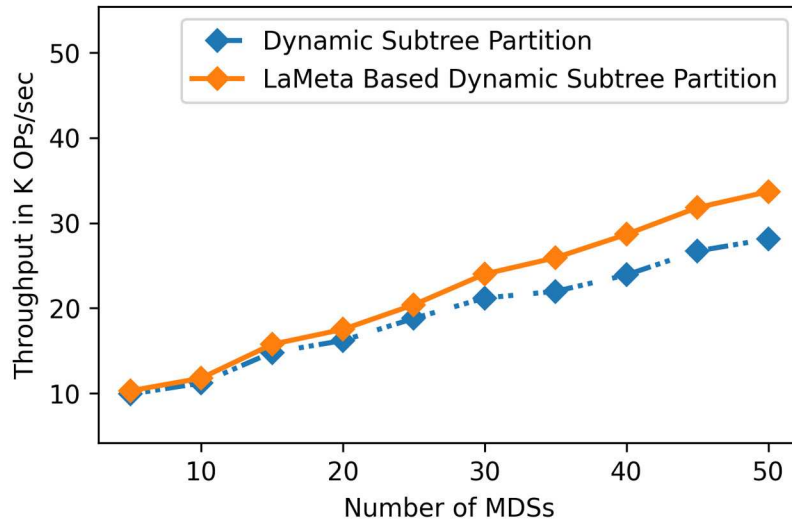


Figure 5.10: Throughput of LaMeta based Dynamic subtree metadata distribution techniques using MSN dataset.

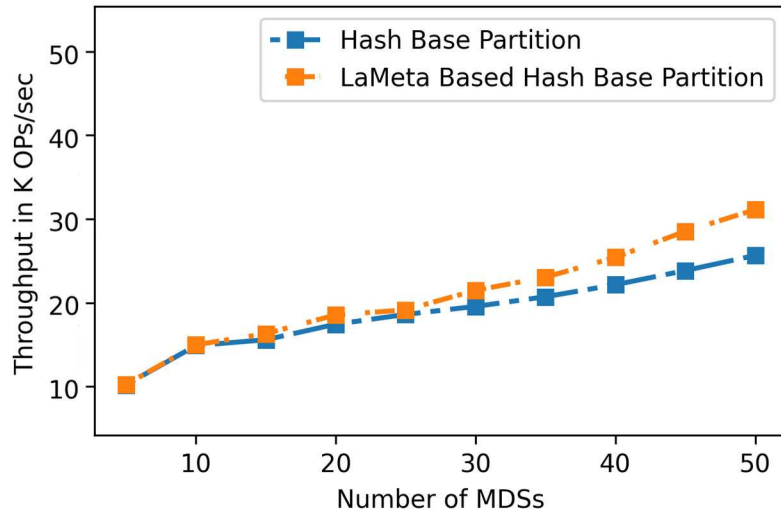


Figure 5.11: Throughput of LaMeta Hash based metadata distribution techniques using MSN dataset.

smaller scale, and the improvement is relatively modest. As it moved to the mid-range MDSs, the throughput gained nearly 14% compared to the baseline technique.

However, as we shift our focus to a larger MDS scale, specifically ranging from 25 to 50 MDSs, the impact of the LaMeta technique becomes more apparent in the results. In this scenario, LaMeta-based DROP exhibits a more significant and consistent performance gain of 21% and more in terms of metadata operations. At one instance, it shows a 26% gain in throughput of DROP the metadata operations. This notable improvement is attributed to LaMeta's effective utilization of implicit locality-preserving hashing in

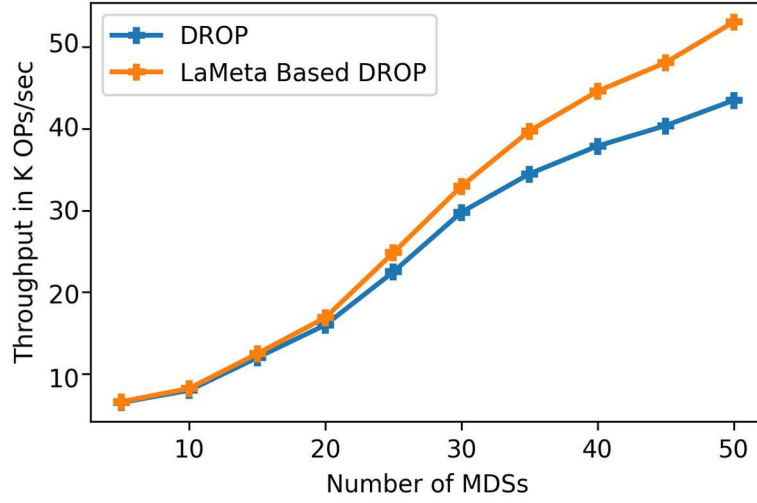


Figure 5.12: Throughput of LaMeta based DROP metadata distribution techniques using MSN dataset.

the DROP technique. The LaMeta-based DROP consistently outperforms the standard DROP across all scales and datasets, with percentage increases in throughput ranging from approximately 15% to 26% in mid-range and larger MDSs networks. This consistent improvement underscores LaMeta’s capability to optimize the performance and efficiency of scalable ultra-large storage systems

Furthermore, Fig. 5.13. presents the percentage increase in average aggregate throughput across all three datasets combined, providing a clear picture of the performance enhancements achieved by LaMeta-based metadata distribution techniques. The LaMeta-based enhancements consistently improve the throughput of various metadata distribution techniques across all scales. In this, LaMeta-based Dynamic Subtree Partition and LaMeta-based DROP show the highest percentage increases, particularly at larger scales, indicating superior scalability and performance optimization capabilities. The LaMeta Hash Base Partition also demonstrates significant improvements, while the LaMeta-based Static Subtree Partition shows steady but more modest gains in throughput.

These results underscore the positive correlation between the scale of the Metadata Servers and the routing of metadata operations, with our proposed LaMeta-based techniques consistently enhancing aggregate throughput in terms of metadata operations per second.

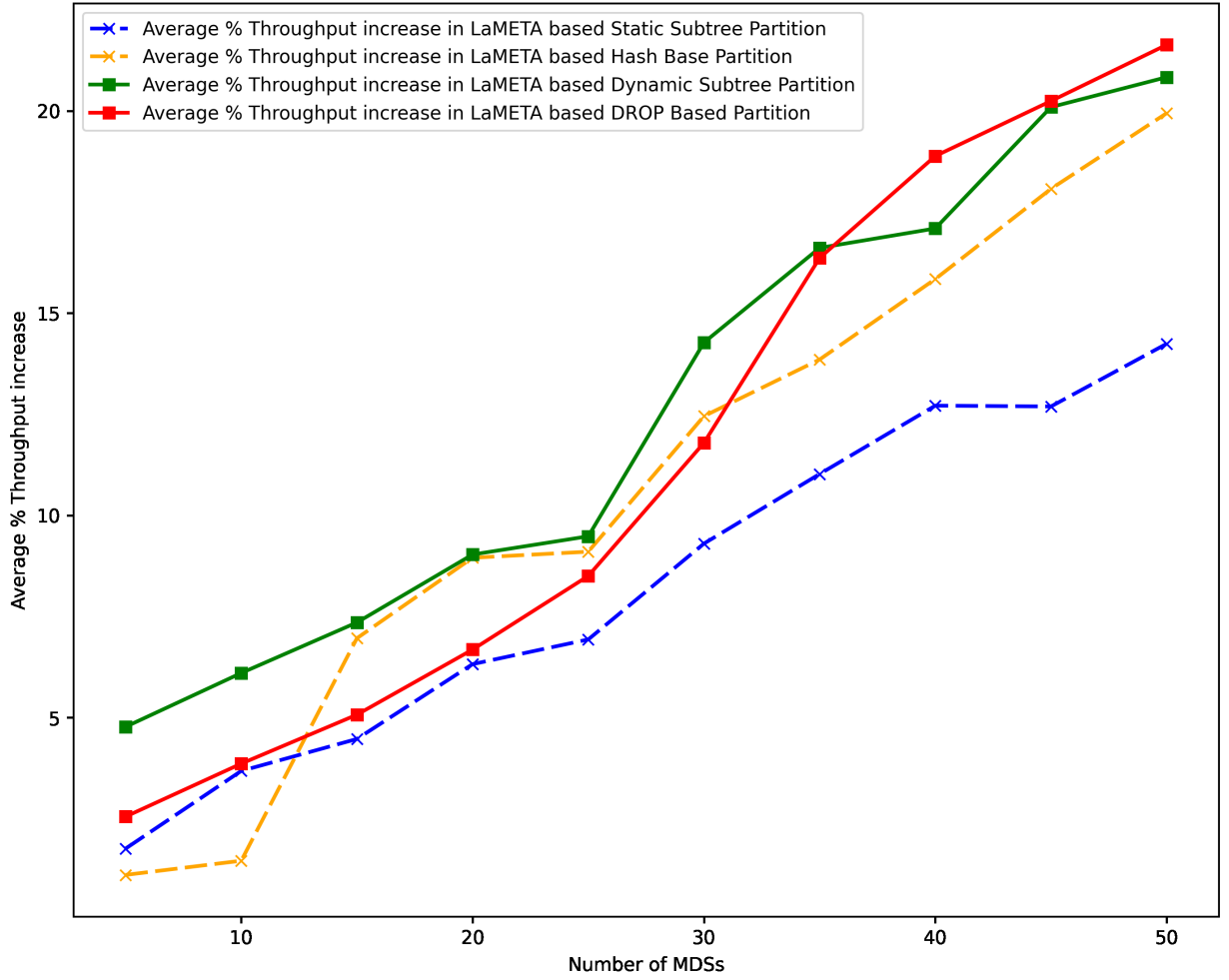


Figure 5.13: Average percentage increase in aggregate throughput of metadata distribution techniques using all three datasets.

5.5 Locality Comparison

The proposed *loc* function, which was introduced in section 4.6, serves as a measure of locality within the metadata management system. It quantifies locality using negative values and takes into account two key input variables: the number of hops and the intermediate MDS path ratings.

In metadata operations, each operation follows a specific path encompassing multiple hops, and it encounters various path ratings during its execution. The efficiency of locality-aware metadata routing is assessed by computing the average locality, denoted as *loc*, which is gathered at specific hop intervals. This allows us to evaluate how well the system maintains locality as metadata operations progress through different routes.

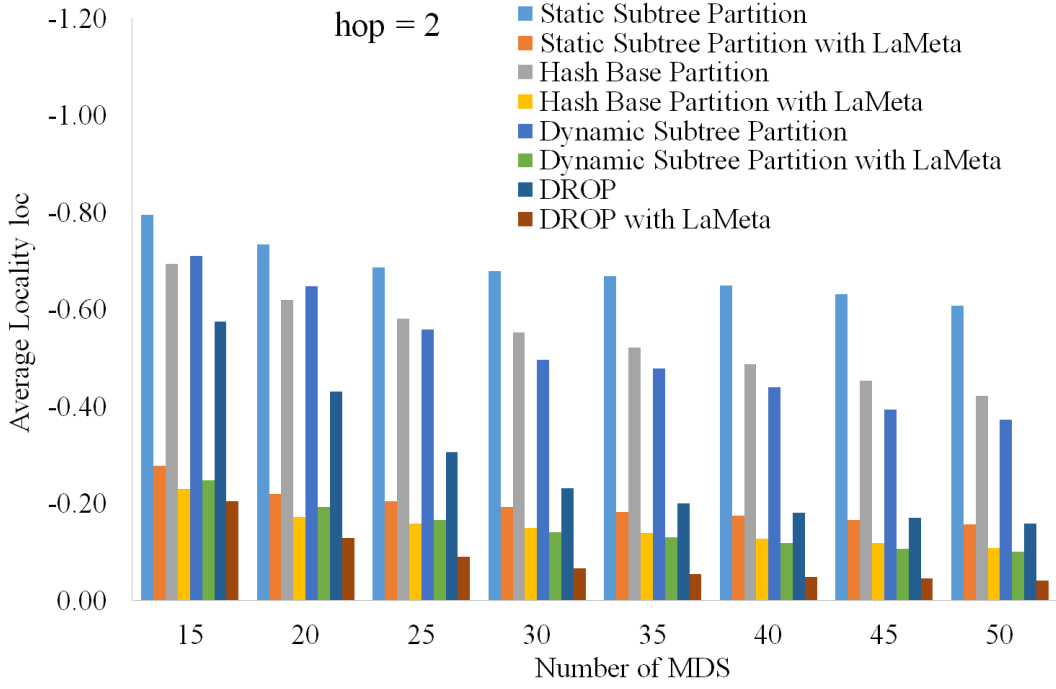


Figure 5.14: The average locality of metadata operations using the MSN dataset at hop=2.

Hop = 2								
Number of MDS	Static Sub-tree Partition	Static Sub-tree Partition with LaMeta	Hash Base Partition	Hash Base Partition with LaMeta	Dynamic Sub-tree Partition	Dynamic Sub-tree Partition with LaMeta	DROP	DROP with LaMeta
15	-0.8	-0.28	-0.69	-0.23	-0.71	-0.25	-0.57	-0.21
20	-0.73	-0.22	-0.62	-0.17	-0.65	-0.19	-0.43	-0.13
25	-0.69	-0.2	-0.58	-0.16	-0.56	-0.17	-0.31	-0.09
30	-0.68	-0.19	-0.55	-0.15	-0.5	-0.14	-0.23	-0.07
35	-0.67	-0.18	-0.52	-0.14	-0.48	-0.13	-0.2	-0.05
40	-0.65	-0.18	-0.49	-0.13	-0.44	-0.12	-0.18	-0.05
45	-0.63	-0.17	-0.45	-0.12	-0.39	-0.11	-0.17	-0.05
50	-0.61	-0.16	-0.42	-0.11	-0.37	-0.1	-0.16	-0.04

Table 5.15: Comparison of LaMeta metrics *loc* for different methods at Hop = 2

To provide a comprehensive view of the results, we measure and compare the average locality achieved using various metadata distribution techniques when combined with LaMeta and its corresponding baseline technique. The evaluations are conducted at different hop intervals, specifically at $hop = 2$, $hop = 4$, $hop = 6$ and $hop = 8$, and these

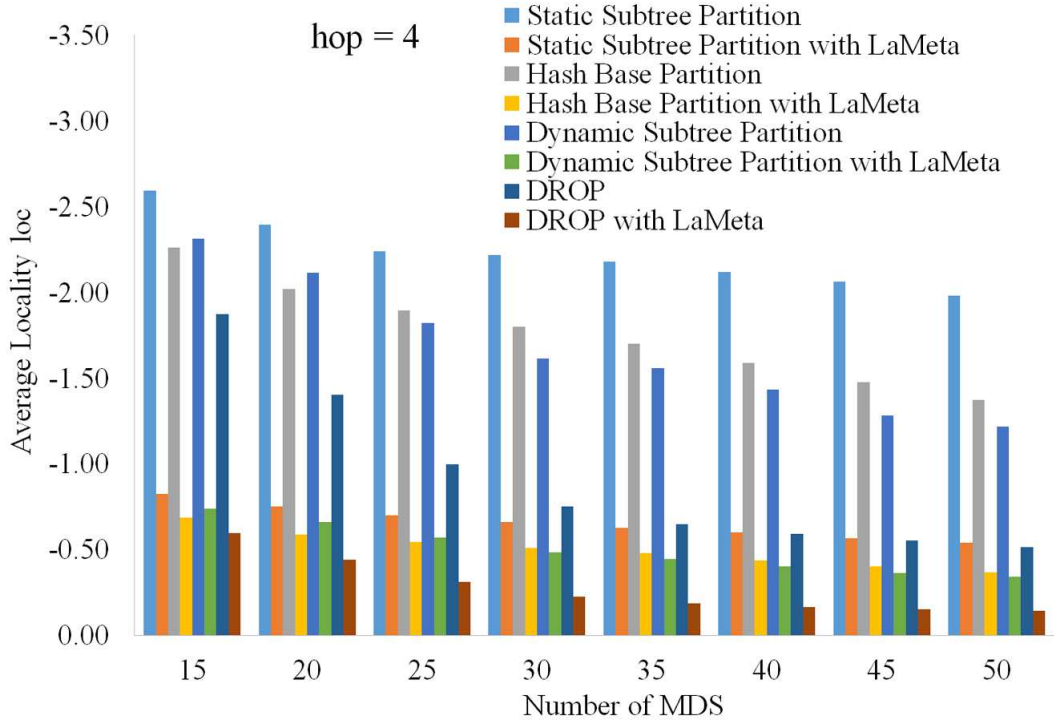


Figure 5.15: The average locality of metadata operations using the MSN dataset at hop=4.

Hop = 4								
Number of MDS	Static Sub-tree Partition	Static Sub-tree Partition with LaMeta	Hash Base Partition	Hash Base Partition with LaMeta	Dynamic Sub-tree Partition	Dynamic Sub-tree Partition with LaMeta	DROP	DROP with LaMeta
15	-2.59	-0.83	-2.26	-0.69	-2.32	-0.74	-1.87	-0.6
20	-2.39	-0.75	-2.02	-0.59	-2.12	-0.66	-1.4	-0.44
25	-2.24	-0.7	-1.89	-0.55	-1.82	-0.57	-1	-0.31
30	-2.22	-0.66	-1.8	-0.51	-1.62	-0.48	-0.75	-0.23
35	-2.18	-0.63	-1.7	-0.48	-1.56	-0.45	-0.65	-0.19
40	-2.12	-0.6	-1.59	-0.44	-1.43	-0.41	-0.59	-0.17
45	-2.06	-0.57	-1.48	-0.4	-1.28	-0.37	-0.56	-0.15
50	-1.98	-0.54	-1.37	-0.37	-1.22	-0.35	-0.52	-0.14

Table 5.16: Comparison of LaMeta metrics *loc* for different methods at Hop = 4

are performed within the MDS scales ranging from 15 to 50. The findings and insights regarding the average locality are presented graphically in Fig. 5.14, 5.15, 5.16 and 5.17. The data values of the same are provided in the Tables 5.15, 5.16, 5.17 and 5.18.

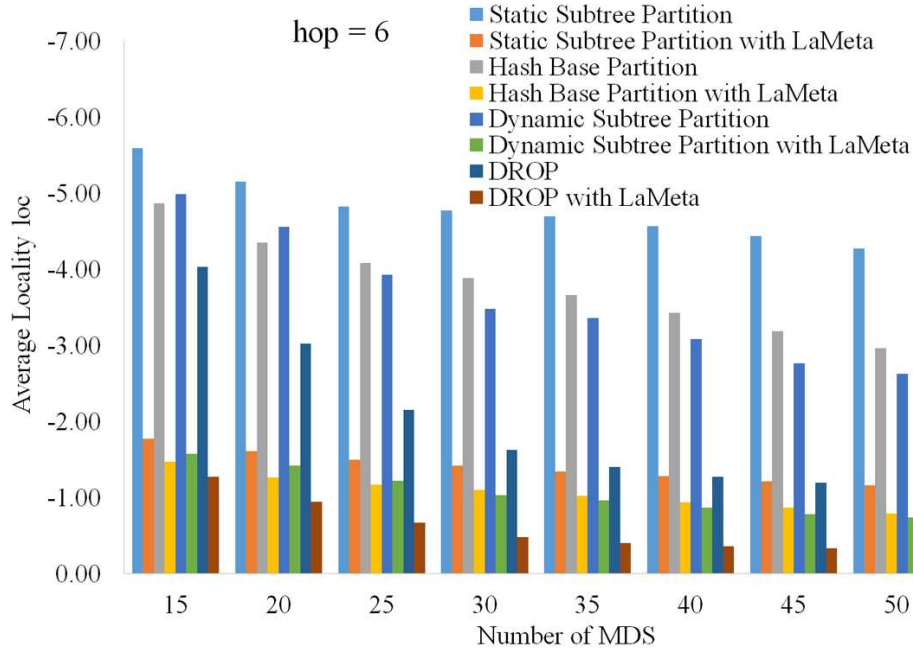


Figure 5.16: The average locality of metadata operations using the MSN dataset at hop=6.

Hop = 6								
Number of MDS	Static Sub-tree Partition	Static Sub-tree Partition with LaMeta	Hash Base Partition	Hash Base Partition with LaMeta	Dynamic Sub-tree Partition	Dynamic Sub-tree Partition with LaMeta	DROP	DROP with LaMeta
15	-5.59	-1.77	-4.87	-1.47	-4.99	-1.58	-4.04	-1.28
20	-5.16	-1.61	-4.36	-1.26	-4.56	-1.42	-3.03	-0.94
25	-4.83	-1.5	-4.08	-1.17	-3.93	-1.22	-2.15	-0.67
30	-4.78	-1.42	-3.88	-1.1	-3.48	-1.04	-1.63	-0.48
35	-4.7	-1.34	-3.67	-1.03	-3.36	-0.96	-1.4	-0.4
40	-4.57	-1.28	-3.43	-0.94	-3.09	-0.87	-1.28	-0.36
45	-4.44	-1.22	-3.19	-0.87	-2.76	-0.78	-1.2	-0.33
50	-4.27	-1.16	-2.96	-0.79	-2.63	-0.74	-1.11	-0.31

Table 5.17: Comparison of LaMeta metrics *loc* for different methods at Hop = 6

The Fig. 5.14, 5.15, 5.16 and 5.17 illustrate the average locality (*loc*) measurements in metadata operation routing for various metadata distribution techniques, both with and without LaMeta, at different hop intervals (2, 4, 6, and 8) and across different scales of Metadata Servers. This discussion analyses the locality improvements achieved through

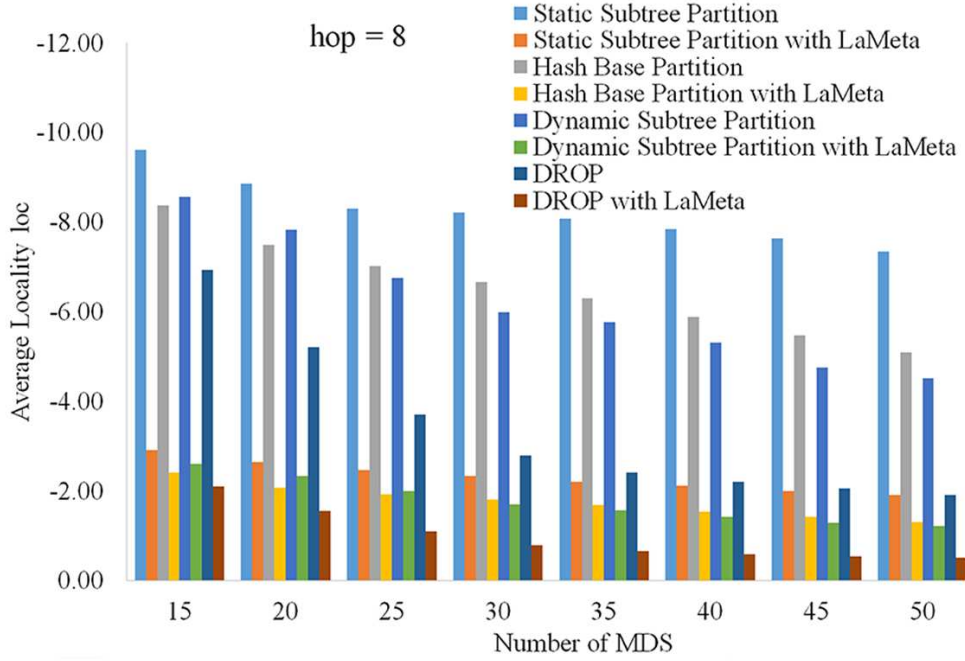


Figure 5.17: The average locality of metadata operations using the MSN dataset at hop=8.

Hop = 8								
Number of MDS	Static Sub-tree Partition	Static Sub-tree Partition with LaMeta	Hash Base Partition	Hash Base Partition with LaMeta	Dynamic Sub-tree Partition	Dynamic Sub-tree Partition with LaMeta	DROP	DROP with LaMeta
15	-9.61	-2.91	-8.37	-2.41	-8.57	-2.59	-6.93	-2.1
20	-8.86	-2.64	-7.49	-2.08	-7.83	-2.33	-5.2	-1.55
25	-8.29	-2.46	-7.02	-1.92	-6.75	-2	-3.7	-1.1
30	-8.21	-2.33	-6.67	-1.8	-5.98	-1.7	-2.79	-0.79
35	-8.08	-2.21	-6.3	-1.69	-5.77	-1.58	-2.41	-0.66
40	-7.85	-2.11	-5.89	-1.54	-5.31	-1.43	-2.2	-0.59
45	-7.63	-2	-5.48	-1.42	-4.75	-1.28	-2.06	-0.54
50	-7.34	-1.9	-5.09	-1.3	-4.51	-1.21	-1.91	-0.5

Table 5.18: Comparison of LaMeta metrics *loc* for different methods at Hop = 8

the LaMeta-based techniques compared to their respective baseline techniques.

The average locality of metadata operations tends to increase as the scale of Metadata Servers (MDS) grows. This is primarily because metadata is distributed on a larger number of MDS, and many paths become available, offering more options for efficient routing.

Interestingly, our LaMeta-based techniques demonstrate more substantial improvements in locality as the scale of MDS and the number of hops increase.

In Fig. 5.14, 5.15, 5.16 and 5.17, we observe that the *loc* value decreases as the number of hops for metadata operations increases. In Fig. 5.14, LaMeta-based techniques slightly improve locality compared to their baselines, reflecting the initial benefit of optimized routing with fewer hops $hop = 2$. The locality values show more variation in Fig. 5.15, indicating increased routing complexity at $hop = 4$. The LaMeta-based techniques consistently outperform their baseline counterparts, with noticeable improvements in locality. In Fig. 5.16, the locality values decrease, reflecting the added complexity of longer paths with $hop = 6$. LaMeta-based techniques continue to exhibit improved locality, with the gap between baseline and LaMeta-based methods widening. In Fig. 5.17, the locality values are the lowest, indicating the highest complexity in routing paths at $hop = 8$, among others mentioned. LaMeta-based techniques show the most pronounced improvements in locality at this interval, highlighting their robustness in managing complex routing. The LaMeta-based Dynamic Subtree Partition and LaMeta-based DROP demonstrate the highest locality improvements, emphasizing the effectiveness of LaMeta in maintaining efficient routing over longer hops.

This decline in locality is expected as longer hops generally traverse a more complex path. However, our proposed LaMeta-based technique showcases a notable enhancement in the average locality, and this improvement becomes more pronounced with an increasing number of hops. These findings underline the effectiveness of our LaMeta approach, particularly within large and distributed MDS networks found in storage systems.

In summary, the aggregate throughput and average locality achieved by our proposed LaMeta-based techniques consistently outperform their respective baseline metadata distribution techniques. This underscores the locality and scalability of the LaMeta technique for deployment in ultra-large, globally distributed metadata management systems.

5.6 Conclusion

We have introduced an innovative metadata management technique called LaMeta, designed to address the critical issue of locality within scalable distributed metadata management systems. The novel LaMeta exploits the MDS path attribute in the metadata distribution structure and proposed routing of metadata operations. To overcome the challenges posed by sparsity in the path rating matrix of MDSs and to make predictions, we have harnessed matrix factorization models based on collaborative filtering techniques and implemented a Stacked AutoEncoder (SAE). Incorporating the Stacked AutoEncoder into LaMeta enables us to predict previously unknown path measurements. These predictions are instrumental in optimizing the routing of metadata operations, as we can choose paths with the least latency. Furthermore, LaMeta computes the optimal k Next MDS based on the prediction matrix to enhance the metadata distribution structure.

Our novel LaMeta technique has been extensively tested alongside contemporary metadata distribution techniques, emphasizing locality exploitation. We evaluated the performance of LaMeta-based metadata distribution techniques against contemporary methods, measuring operations per second across varying scales of Metadata Servers (MDSs) from 5 to 50. Performance gains were assessed based on the aggregate throughput of metadata operations. The LaMeta-based static subtree metadata distribution techniques show gains from 2% to 5% at the 5 MDs scale, 9% to 12% at the 10-20 MDS scale and 12% to 14% at 25 to 50 than baseline static subtree partitioning. In LaMeta-based dynamic subtree partitioning, the average throughput gains range from 4% to 9% at 5 MDS, 10% to 20% at 10-30 MDSs scale and more than 20% at 40-50 MDS scale. While in LaMeta-based hash partitioning, the gains are 1% to 9% at 5-20 MDS scale, 9% to 19% at 25-40 MDS scale and more than 24% at 50 MDS scale.

The experimental results underscore the significant advantages of LaMeta-based metadata distribution techniques as the scale of Metadata Servers (MDS) increases. Specifically, LaMeta consistently outperforms baseline contemporary metadata distribution techniques in terms of aggregate throughput and average locality, demonstrating its effectiveness and scalability in real-world scenarios.

Storage is a core component of computing applications. Latency-sensitive data storage is essential in various computer applications where minimizing data access delays is critical for optimal performance and user experience. Some computer applications that rely on low-latency data storage include Finance and High-Frequency Trading (HFT), Online Gaming, Content Delivery Networks (CDNs), Cloud Computing, IoT (Internet of Things), Autonomous Vehicles, Video streaming, Healthcare, Aerospace and Defence, cyber security and many more. The proposed locality-aware metadata management (LaMeta) for large distributed storage systems is effective for all such latency-sensitive data storage and retrieval applications.

Chapter 6

CONCLUSION

This chapter presents the research’s conclusions, thesis Contributions and future directions, thereby concluding the thesis.

Section 6.1 discusses the Thesis Contributions and 6.2 overall conclusions drawn from this work. In addition, the chapter suggests potential future research directions and extensions of this work, providing valuable insights for further exploration in Section 6.3.

6.1 Contributions

1. A comprehensive survey of various metadata management techniques was conducted, exploring the need for distributed metadata management and analyzing its advantages and disadvantages. The study covered different distributed metadata management schemes, providing detailed insights into their significance.
2. Contemporary metadata management techniques were compared and classified based on taxonomies, including static subtree-based, table-based, hash-based, dynamic subtree-based, distributed hash table-based, and bloom filter-based techniques. Key performance parameters like metadata lookup time, load balancing, scalability, memory overhead, metadata migration, recovery, locality, and replication were analyzed and discussed.
3. The research gaps and future directions in the field of distributed metadata management were identified and presented based on an exhaustive survey of existing techniques.
4. A novel metadata locality-aware technique (LaMeta) was proposed, leveraging MDS locality in a globally distributed ultra-large cloud storage system to enhance meta-

data management performance. LaMeta-based static subtree, dynamic subtree, hash-based, and DROP-based metadata distribution techniques for ultra-large distributed storage systems were introduced.

5. The performance of collaborative filtering-based MF techniques (SVD, SVD++, RNMF) and Stacked AutoEncoder was evaluated the predicting path for metadata query routing in distributed storage systems. LaMeta based on Stacked AutoEncoder (SAE) structure was proposed for efficient and intelligent MDS locality prediction in metadata distribution techniques.
6. A locality-aware next k MDS algorithm for metadata query routing was proposed, defining the metadata operation's locality to compute optimal metadata operation routing. The MDSs utilize recommended top k MDS with optimum path ratings for the next hop in the metadata distribution structure.
7. We evaluated the performance of LaMeta-based metadata distribution techniques against contemporary methods, measuring operations per second across varying scales of Metadata Servers (MDSs) from 5 to 50. Performance gains were assessed based on the aggregate throughput of metadata operations. The LaMeta-based static subtree metadata distribution techniques show gains from 2% to 5% at the 5 MDs scale, 9% to 12% at the 10-20 MDS scale and 12% to 14% at 25 to 50 than baseline static subtree partitioning. In LaMeta-based dynamic subtree partitioning, the average throughput gains range from 4% to 9% at 5 MDS, 10% to 20% at 10-30 MDSs scale and more than 20% at 40-50 MDS scale. While in LaMeta-based hash partitioning, the gains are 1% to 9% at 5-20 MDS scale, 9% to 19% at 25-40 MDS scale and more than 24% at 50 MDS scale.
8. Extensive experiments and performance evaluations of LaMeta-based static subtree, dynamic subtree, hash-based, and DROP-based metadata distribution techniques were conducted using real metadata datasets. The results demonstrated that the proposed LaMeta-based techniques significantly improved the locality and performance of baseline metadata distribution in terms of aggregate throughput and metadata operation locality.

6.2 Conclusions

This research presents locality-aware distributed metadata management or storage and retrieval techniques for very large storage systems. The technique exploits the locality for efficient metadata operation routing. The following section presents the main outcomes of this work:

Chapter 1 The thesis commences with an introduction of metadata management or storage and retrieval techniques for large-scale storage systems in Chapter 1. It offers a comprehensive overview of the evolution of distributed metadata management, and the merits and challenges of distributed metadata management. The chapter briefly outlines the research motivation behind distributed metadata management and presents the primary contributions of this work. Finally, the chapter discusses the organization of the rest of the thesis.

Chapter 2 presents a systematic review of the literature on various distributed metadata management techniques for a very large storage system. The analysis provides a comprehensive overview of these techniques, presents a taxonomy of metadata distribution techniques based on the distribution structure, thoroughly discusses the strengths and shortcomings of metadata distribution techniques and presents a detailed comparison based on key performance parameters of distributed metadata management techniques. Furthermore, based on existing research challenges, the objectives of the thesis have been outlined.

Chapter 3 proposed a Locality aware metadata management (LaMeta) technique, aims to address the issue of the locality of metadata on MDSs in the metadata distribution techniques that improve metadata query routing and enhance the performance of the distributed storage system. This chapter also discusses the architecture of the proposed locality-aware metadata management (LaMeta). This chapter describes the constituents of LaMeta like MDS path measurement for locality and the need for metadata server locality for performance enhancement. The chapter further discusses the collaborative filtering-based MF techniques Singular Value Decomposition (SVD), SVD++, Regularized Non-Negative Matrix Factorization, and Stacked AutoEncoder for predicting the

path ratings.

Chapter 4 presents the MDS path probing and Path Ratings for the locality of metadata on distributed MDSs. This chapter includes the detailed design structures of the LaMeta technique. This includes the structure of the proposed Stacked AutoEncoder, locality-aware next K MDS and their algorithms. The chapter presents the translation of MDS path RTT values to MDS path ratings and input data preprocessing using one-hot encoding for Stacked AutoEncoder. The chapter further discusses the proposed locality expression loc of metadata operation routing in value to measure and compare the impact of LaMeta on distribution metadata techniques in an ultra-large distributed storage system.

Chapter 5 validates the proposed LaMeta-based static subtree, dynamic subtree, hash-based and DROP-based techniques. First, it evaluates the performance of the recommendation system based on comparing prediction accuracy of inferred ratings on test data sets. It evaluates the performance of proposed LaMeta-based techniques using three real datasets and four contemporary scalable metadata distribution techniques in terms of aggregate throughput of metadata distribution techniques. The chapter presents the experimental results of the average percentage increase in aggregate throughput of LaMeta-based metadata techniques using all three datasets. Finally, it evaluates the performance of locality-aware metadata routing using average locality loc gathered at particular numbers of hops. The extensive experimental results show that with the increasing scale of MDS, the LaMeta-based metadata distribution techniques significantly outperform in terms of aggregate throughput and improve average locality compared to base contemporary metadata distribution techniques.

6.3 Future Directions

This effort investigates the gaps and unexplored areas in distributed metadata management, particularly focusing on a very large distributed storage system. Consequently, this section presents some suggestions for potential further research to extend this work in the future:

1. In the future, researchers can focus on exploring and predicting other features or combinations of features of metadata distribution techniques for efficiency for metadata operations at a finer granularity.
2. Despite significant improvements in storage system scalability, existing metadata distribution techniques still have overheads like memory usage, migration cost, lookup time, and load balancing. Future research should focus on advanced techniques that optimize metadata distribution without additional overhead, efficiently using an optimal number of MDSs.
3. Hierarchical metadata distribution schemes face challenges in maintaining parent-child relationships, especially when performing operations like renaming the root directory. Imbalanced distribution may occur due to runtime creation of child objects. Research should address these constraints in static and dynamic subtree partitioning-based techniques.
4. Achieving metadata locality while maintaining load balancing is essential for globally distributed storage systems. Researchers should explore techniques that balance metadata distribution, migration cost, and locality preservation.
5. Bloom Filter-based metadata storage and retrieval can suffer from false-positive membership queries, leading to multicasting of lookup queries. Future research should focus on minimizing multicasting and optimizing metadata migration to improve performance.
6. Distributed storage systems require high availability achieved through data and metadata replication, which puts overhead on memory and computation. Research should address the tradeoff between availability and performance.
7. Geographically distributed metadata and data must adhere to various regulatory compliance rules. Future research should focus on secure, privacy-conscious, and compliant storage systems.
8. Future storage systems need scalable high data stream throughput. Research should develop efficient metadata schemes to support high data streams, balancing work-

load and adapting to changing requirements.

9. QoS policies for metadata operations in cloud-based environments need exploration. Such policies can ensure different quality levels for various applications and users, potentially leading to costing models for cloud storage providers.
10. Client Application Intimacy: Improving client application intimacy with distributed metadata can enhance query performance. Caching metadata differently based on repeated query patterns and implementing client-side caches are areas for future research.
11. As the demand for storage grows, research can focus on Metadata Cloud Services and metadata delivery networks that leverage metadata distribution schemes to provide quality experiences to end-users and applications.
12. Two limitations can be addressed in the proposed LaMeta. First, accessing the RTT value probing consumes resources and these values are highly sensitive to the amount of traffic on that path. Second, LaMeta prioritizes the best possible paths, and these paths' characteristics may change, requiring resource-intensive changes in metadata distribution techniques.
13. The k and r values in the proposed path rating algorithm of LaMeta can be learned. Their impact on the LaMeta-based metadata distribution technique's performance can be explored in future research.

Appendix A

Appendixes

A.1 Summary of Metadata Management Techniques

Technique Name	Description	Citation	Article Code	Year	Organization
FAST	Semantic Aware Search, Dynamic and Adaptive Namespace	[102]	R37	2016	Huazhong University of Science and Technology, Wuhan, China
MaaS	Cloud Based Metadata, Bloom Filter	[100]	R36	2015	Sri Venkateswara College of Engineering, TK, India
Granary	DHT-based	[91]	R35	2014	Tsinghua University, Beijing, China
PPMS	Peer to Peer, Metadata Index Service.	[97]	R34	2014	Sun Yat-sen University, China
H2FS	Hybrid Hierarchy Partitioning, Reduce I/O	[89]	R33	2014	State Key Laboratory of High Performance Computing, Changsha, China
MRFS	Geo-Replicated Metadata, Hierarchical, Replication	[85]	R32	2014	Sun Yat-sen University, China

Continued on next page

Table A.1 – (Continued)

Technique Name	Description	Citation	Article Code	Year	Organization
DROP	HDLB	[104]	R31	2013	Data Storage Institute, Singapore
AbFS	Two-level Hash/ Table, Hashing, Reduce I/O, Client Cache	[103]	R30	2013	University of Granada, Granada, Spain
GHBA	Group Wise HBA	[2, 30]	R29	2011	Huazhong University of Science and Technology, Wuhan, China
RFS	DHT Ring, Scalability, Fault Tolerance	[94]	R28	2011	University of Illinois at Urbana-Champaign, Urbana, USA
DDG	Distribution, Consistency	[5]	R27	2011	Chinese Academy of Sciences, Beijing, China
DLBH	Directory Level Based Hash	[78]	R26	2011	Huazhong University of Science and Technology, Wuhan, China
HVFS	2-Level Directory Partitioning, Extendable Hashing.	[76]	R25	2011	Chinese Academy of Sciences, Beijing, China
FPFS	OSD+, Reduce I/O, Client Cache, Lustre File System Improvement	[73, 74]	R24	2011	University of Murcia, Spain

Continued on next page

Table A.1 – (Continued)

Technique Name	Description	Citation	Article Code	Year	Organization
Dcache	Caching in Client and Server side	[93]	R23	2010	Wuhan University, China
Cassandra	Decentralized Storage system	[92]	R22	2010	Facebook, USA
Gfarm	Global Distributed File System	[86]	R21	2010	University of Tsukuba, Ibaraki, Japan
HBA	Bloom Filter Array	[32]	R20	2008	University of Maine, Orono, ME
DIMS	Directory Hashing, Subtree Partitioning	[77]	R19	2008	Huazhong University of Science and Technology, Wuhan, China
GIGA+	Partitioning, B-link tree or extendible hashing	[26]	R18	2007	Carnegie Mellon University, USA
Kosha	Peer to Peer	[35]	R17	2006	Purdue University, West Lafayette, USA
CEPH	Decoupling, Dynamic Load Balancing	[44]	R16	2006	University of California, Santa Cruz, USA
RBMMs	Replica, Graph, Subtree Partitioning	[87]	R15	2006	Huazhong University of Science and Technology, Wuhan, China
GDSS	Distributed Match table, Subtree Partitioning	[88]	R14	2005	Huazhong University of Science and Technology, Wuhan, China

Continued on next page

Table A.1 – (Continued)

Technique Name	Description	Citation	Article Code	Year	Organization
SRB	Geographical Distributed File System, Static Partitioning of Metadata	[84]	R13	2005	Duke University, Durham, USA
OBFS	OSD	[12]	R12	2004	University of California, Santa Cruz, USA
Lustre	Hash, Interleaved Hash, MDT	[68]	R11	2004	Carnegie-Mellon University and Cluster File System Inc., Intel Corp., USA
zFS	Object Storage Devices	[41]	R10	2003	IBM Labs, Haifa University, Israel
Farsite	Replication	[90]	R9	2002	Microsoft, USA
HeRMES	B+ Trees , Hashing	[65]	R8	2001	University of California, Santa Cruz, USA
RAMA	Hashing	[69]	R7	1997	University of Maryland Baltimore County, USA
xFS	B+ Trees, Manager-Map Table for distribution and sharing of resources	[37]	R6	1995	University of California, Berkeley, USA
NFSv3	Distributed file system	[79]	R5	1994	Sun Microsystem, USA

Continued on next page

Table A.1 – (Continued)

Technique Name	Description	Citation	Article Code	Year	Organization
Vesta	Parallel view, Hashing	[70, 71]	R4	1994	IBM T.J. Watson Research Center, New York, USA
CODA	Distributed file system	[81, 82]	R3	1990	Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA
Sprite	Distributed file system	[83]	R2	1988	University of California at Berkeley, Berkeley, USA
AFS	Distributed file system	[80]	R1	1986	Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA

A.2 Abbreviations List

Terms	Description	Terms	Description
MDS	Metadata Server	Gfarm	Gfarm Grid File System
P2P	Peer to Peer	RBMMS	Replica-Based Distributed Meta- data Management
DN	Data Node	GDSS	Global Distributed Storage Sys- tem
xFS	X File System (SUN, LINUX)	Farsite	Federated Available and Reliable Storage Incompletely Trusted En- vironment
HeRMES	High performance, Reliable, MRAM-Enabled Storage	DCFS	Dawning Cluster File System
zFS	Z File System(SUN, LINUX)	OBFS	Object-Based File System
Vesta	Vesta Parallel File System	P2P	Peer to Peer
Lustre	Lustre Distributed File Sys- tem	DDG	Dynamic Dir-Grain
RAMA	Rapid Access to Massive Archive	H ² FS	Hybrid Hierarchy File System
SPOF	Single Point of Failure	PPMS	Pear to Pear Metadata Manage- ment Service
AbFS	Andrew File System	Kosha	Pear to Pear Enhancement of NFS
FPFS	Fusion Parallel File System	Granary	DHT Metadata Storage
HVFS	Hyper Virtual File System	Cassandra	Decentralized Structure Storage for Facebook
IDC	International Data Corpo- ration	Dcache	Double Cache

Continued on next page

Table A.2 – (Continued)

Terms	Description	Terms	Description
DLBH	Directory-Level-Based Hash	DROP	Dynamic Ring Online Partitioning
NFS	Network File System	RFS	Ring File System
AFS	Andrew File System	HBA	Hieratical Bloom Filter Array
CODA	Constant Data availability	GHBA	Group-HBA
MRFS	Metadata Replication File System	MaaS	Metadata as a Service
SRB	Storage Resource Broker	DHT	Distributed Hash Table

List of Publications

Journal Publications (SCI/SCIE):

1. **Harcharan Jit Singh**, and Seema Bawa. "Scalable metadata management techniques for ultra-large distributed storage systems—A systematic review." ACM Computing Surveys (CSUR) 51, no. 4 (2018): 1-37, [**SCI/SCIE Indexed, Impact Factor - 23.80**]
2. **Harcharan Jit Singh**, and Seema Bawa. "LaMeta: An efficient locality aware metadata management technique for an ultra-large distributed storage system." Journal of King Saud University-Computer and Information Sciences, 34 (10), 8323-8335 (2022), [**SCIE Indexed, Impact Factor - 5.2**]

References

- [1] D. R.-J. G.-J. Rydning, J. Reinsel, and J. Gantz, “The digitization of the world from edge to core,” *Framingham: International Data Corporation*, vol. 16, pp. 1–28, 2018.
- [2] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, “Supporting scalable and adaptive metadata management in ultralarge-scale file systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 4, pp. 580–593, 2010.
- [3] C. Karamanolis, L. Liu, M. Mahalingam, D. Muntz, and Z. Zhang, “An architecture for scalable and manageable file services,” *Hewlett-Packard Labs, Palo Alto, Technical Report, HPL-2001-173*, 2001.
- [4] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg, “Ibm storage tank—a heterogeneous scalable san file system,” *IBM Systems Journal*, vol. 42, no. 2, pp. 250–267, 2003.
- [5] J. Xiong, Y. Hu, G. Li, R. Tang, and Z. Fan, “Metadata distribution and consistency techniques for large-scale cluster file systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 803–816, 2010.
- [6] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller, “Dynamic metadata management for petabyte-scale file systems,” in *SC’04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, pp. 4–4, IEEE, 2004.
- [7] S. Sinnamohideen, R. R. Sambasivan, J. Hendricks, L. Liu, and G. R. Ganger, “A {Transparently-Scalable} metadata service for the ursa minor storage system,” in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*.
- [8] J. Piernas, T. Cortes, and J. M. Garcia, “The design of new journaling file systems: The dualfs case,” *IEEE Transactions on Computers*, vol. 56, no. 2, pp. 267–281, 2007.

- [9] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, “A survey of large scale data management approaches in cloud environments,” *IEEE communications surveys & tutorials*, vol. 13, no. 3, pp. 311–336, 2011.
- [10] D. Roselli, J. R. Lorch, and T. E. Anderson, “A comparison of file system workloads,” in *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, 2000.
- [11] D. Long, S. Brandt, E. Miller, F. Wang, Y. Lin, L. Xue, and Q. Xin, “Design and implementation of large scale object-based storage system,” tech. rep., Technical Report ucsc-crl-02-35, University of California, Santa Cruz, 2002.
- [12] F. Wang, S. A. Brandt, E. L. Miller, and D. D. Long, “Obfs: A file system for object-based storage devices,” in *MSST*, vol. 4, pp. 283–300, 2004.
- [13] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
- [14] A. V. Srinivas, M. V. Reddy, and D. Janakiram, “Distributed wisdom: designing a replication service for large peer-to-peer data grids,” *IEEE Distributed Systems Online*, vol. 7, no. 3, pp. 2–2, 2006.
- [15] I. Mohiuddin, A. Almogren, M. Al Qurishi, M. M. Hassan, I. Al Rassan, and G. Fortino, “Secure distributed adaptive bin packing algorithm for cloud storage,” *Future Generation Computer Systems*, vol. 90, pp. 307–316, 2019.
- [16] P. Kendrick, T. Baker, Z. Maamar, A. Hussain, R. Buyya, and D. Al-Jumeily, “An efficient multi-cloud service composition using a distributed multiagent-based, memory-driven approach,” *IEEE Transactions on Sustainable Computing*, vol. 6, no. 3, pp. 358–369, 2018.
- [17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pp. 1–10, Ieee, 2010.
- [18] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The google file system,” in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 29–43, 2003.

- [19] A. Hadoop, “Apache hadoop,” URL <http://hadoop.apache.org>, 2011.
- [20] Y. Wang and H. Lv, “Efficient metadata management in cloud computing,” in *2011 IEEE 3rd International Conference on Communication Software and Networks*, pp. 514–519, IEEE, 2011.
- [21] J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson, “A trace-driven analysis of the unix 4.2 bsd file system,” in *Proceedings of the tenth ACM symposium on Operating systems principles*, pp. 15–24, 1985.
- [22] D. R. Karger and M. Ruhl, “Simple efficient load balancing algorithms for peer-to-peer systems,” in *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pp. 36–43, 2004.
- [23] A. Hadoop, “Url <http://hadoop.apache.org>,” Accessed on September, vol. 2, p. 2017, 2014.
- [24] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen, “G-hadoop: Mapreduce across distributed data centers for data-intensive computing,” *Future Generation Computer Systems*, vol. 29, no. 3, pp. 739–750, 2013.
- [25] C. Vorapongkitipun and N. Nupairoj, “Improving performance of small-file accessing in hadoop,” in *2014 11th international joint conference on computer science and software engineering (JCSSE)*, pp. 200–205, IEEE, 2014.
- [26] S. V. Patil, G. A. Gibson, S. Lang, and M. Polte, “Giga+ scalable directories for shared file systems,” in *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing’07*, pp. 26–29, 2007.
- [27] B. Welch, M. Unangst, Z. Abbasi, G. A. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, “Scalable performance of the panasas parallel file system,” in *FAST*, vol. 8, pp. 1–17, 2008.
- [28] M. Cai, M. Frank, B. Yan, and R. MacGregor, “A subscribable peer-to-peer rdf repository for distributed metadata management,” *Journal of Web Semantics*, vol. 2, no. 2, pp. 109–130, 2004.

- [29] P. H. Lensing, T. Cortes, and A. Brinkmann, “Direct lookup and hash-based metadata placement for local file systems,” in *Proceedings of the 6th International Systems and Storage Conference*, pp. 1–11, 2013.
- [30] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, “Scalable and adaptive metadata management in ultra large-scale file systems,” in *2008 The 28th International Conference on Distributed Computing Systems*, pp. 403–410, IEEE, 2008.
- [31] A. Koçi and B. Çiço, “Distributed lock management in cloud computing: Performance and challenges,” in *International Scientific Conference Computer Science2018*, vol. 1, pp. 67–74, 2018.
- [32] Y. Zhu, H. Jiang, J. Wang, and F. Xian, “Hba: Distributed metadata management for large cluster-based storage systems,” *IEEE transactions on parallel and distributed systems*, vol. 19, no. 6, pp. 750–763, 2008.
- [33] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, “Scalability in the xfs file system,” in *USENIX Annual Technical Conference*, vol. 15, 1996.
- [34] J. Xiong, S. Wu, D. Meng, N. Sun, and G. Li, “Design and performance of the dawning cluster file system,” in *2003 Proceedings IEEE International Conference on Cluster Computing*, pp. 232–239, IEEE, 2003.
- [35] A. R. Butt, T. A. Johnson, Y. Zheng, and Y. C. Hu, “Kosha: A peer-to-peer enhancement for the network file system,” *Journal of Grid Computing*, vol. 4, no. 3, pp. 323–341, 2006.
- [36] B. Li, Y. He, and K. Xu, “Distributed metadata management scheme in cloud computing,” in *2011 6th International Conference on Pervasive Computing and Applications*, pp. 32–38, IEEE, 2011.
- [37] T. Von Eicken *et al.*, “Acm operating systems review, vol. 29,” *Issue*, vol. 5, pp. 109–126, 1995.
- [38] S. A. Brandt, E. L. Miller, D. D. Long, and L. Xue, “Efficient metadata management in large distributed storage systems,” in *20th IEEE/11th NASA Goddard*

- Conference on Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings.*, pp. 290–298, IEEE, 2003.
- [39] J. Wang, D. Feng, F. Wang, and C. Lu, “Mhs: A distributed metadata management strategy,” *Journal of Systems and Software*, vol. 82, no. 12, pp. 2004–2011, 2009.
 - [40] K. T. Pollack and S. A. Brandt, “Efficient access control for distributed hierarchical file systems,” in *22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST’05)*, pp. 253–260, IEEE, 2005.
 - [41] O. Rodeh and A. Teperman, “zfs-a scalable distributed file system using object disks,” in *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings.*, pp. 207–218, IEEE, 2003.
 - [42] M. Factor, K. Meth, D. Naor, O. Rodeh, and J. Satran, “Object storage: The future building block for storage systems,” in *2005 IEEE International Symposium on Mass Storage Systems and Technology*, pp. 119–123, IEEE, 2005.
 - [43] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, “Crush: Controlled, scalable, decentralized placement of replicated data,” in *SC’06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pp. 31–31, IEEE, 2006.
 - [44] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, “Ceph: A scalable, high-performance distributed file system,” in *Proceedings of the 7th symposium on Operating systems design and implementation*, pp. 307–320, 2006.
 - [45] W. Li, W. Xue, J. Shu, and W. Zheng, “Dynamic hashing: Adaptive metadata management for petabyte-scale file systems,” in *23rd IEEE/14th NASA Goddard Conference on Mass Storage System and Technologies*, pp. 93–98, Citeseer, 2006.
 - [46] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Voshall, and W. Vogels, “Dynamo: Amazon’s highly available key-value store,” *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007.
 - [47] A. W. Leung, E. L. Miller, and S. Jones, “Scalable security for petascale parallel file systems,” in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*,

pp. 1–12, 2007.

- [48] S. Androutsellis-Theotokis and D. Spinellis, “A survey of peer-to-peer content distribution technologies,” *ACM computing surveys (CSUR)*, vol. 36, no. 4, pp. 335–371, 2004.
- [49] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, “A survey and comparison of peer-to-peer overlay network schemes,” *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.
- [50] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi, “Peer-to-peer resource discovery in grids: Models and systems,” *Future generation computer systems*, vol. 23, no. 7, pp. 864–878, 2007.
- [51] H. Zhang, A. Goel, and R. Govindan, “Improving lookup latency in distributed hash table systems using random sampling,” *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, pp. 1121–1134, 2005.
- [52] X. Liu, S. Wang, and H. Ji, “Double-layer p2p networks supporting semantic search and keeping scalability,” *International Journal of Communication Systems*, vol. 27, no. 12, pp. 3956–3970, 2014.
- [53] R. Gupta and Y. N. Singh, “Reputation aggregation in peer-to-peer network using differential gossip algorithm,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 10, pp. 2812–2823, 2015.
- [54] R. Gupta, N. Singha, and Y. N. Singh, “Reputation based probabilistic resource allocation for avoiding free riding and formation of common interest groups in unstructured p2p networks,” *Peer-to-Peer Networking and Applications*, vol. 9, pp. 1101–1113, 2016.
- [55] A. Goswami, R. Gupta, and G. S. Parashari, “Reputation-based resource allocation in p2p systems: A game theoretic perspective,” *IEEE Communications Letters*, vol. 21, no. 6, pp. 1273–1276, 2017.

- [56] R. Sylvia, “A scalable content-addressable network,” in *ACM SIGCOMM*, vol. 13, pp. 161–172, 2001.
- [57] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, “Tapestry: A resilient global-scale overlay for service deployment,” *IEEE Journal on selected areas in communications*, vol. 22, no. 1, pp. 41–53, 2004.
- [58] A. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems,” in *Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12–16, 2001 Proceedings 2*, pp. 329–350, Springer, 2001.
- [59] P. Maymounkov and D. Mazières, “Kademlia: A peer-to-peer information system based on the xor metric,” in *International Workshop on Peer-to-Peer Systems*, pp. 53–65, Springer, 2002.
- [60] D. Malkhi, M. Naor, and D. Ratajczak, “Viceroy: A scalable and dynamic emulation of the butterfly,” in *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pp. 183–192, 2002.
- [61] Q. Lv, S. Ratnasamy, and S. Shenker, “Can heterogeneity make gnutella scalable?,” in *Peer-to-Peer Systems: First International Workshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1*, pp. 94–103, Springer, 2002.
- [62] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, “Making gnutella-like p2p systems scalable,” in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 407–418, 2003.
- [63] D. Qiu and R. Srikant, “Modeling and performance analysis of bittorrent-like peer-to-peer networks,” *ACM SIGCOMM computer communication review*, vol. 34, no. 4, pp. 367–378, 2004.
- [64] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, “A five-year study of file-system metadata,” *ACM Transactions on Storage (TOS)*, vol. 3, no. 3, pp. 9–es, 2007.

- [65] E. L. Miller, S. A. Brandt, and D. D. Long, “Hermes: High-performance reliable mram-enabled storage,” in *Proceedings Eighth Workshop on Hot Topics in Operating Systems*, pp. 95–99, IEEE, 2001.
- [66] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson, “Cooperative caching: Using remote client memory to improve file system performance,” in *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, pp. 19–es, 1994.
- [67] G. M. Voelker, E. J. Anderson, T. Kimbrel, M. J. Feeley, J. S. Chase, A. R. Karlin, and H. M. Levy, “Implementing cooperative prefetching and caching in a globally-managed memory system,” in *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pp. 33–43, 1998.
- [68] P. Braam, “The lustre storage architecture (tech. rep.),” *Technical report*, 2004.
- [69] E. L. Miller and R. H. Katz, “Rama: An easy-to-use, high-performance parallel file system,” *Parallel Computing*, vol. 23, no. 4-5, pp. 419–446, 1997.
- [70] P. F. Corbett and D. G. Feitelson, “Design and implementation of the vesta parallel file system,” in *Proceedings of IEEE Scalable High Performance Computing Conference*, pp. 63–70, IEEE, 1994.
- [71] P. F. Corbett and D. G. Feitelson, “The vesta parallel file system,” *ACM Transactions on Computer Systems (TOCS)*, vol. 14, no. 3, pp. 225–264, 1996.
- [72] J. Piernas, J. Nieplocha, and E. J. Felix, “Evaluation of active storage strategies for the lustre parallel file system,” in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pp. 1–10, 2007.
- [73] A. Aviles-Gonz, J. Piernas, P. Gonz, *et al.*, “A metadata cluster based on osd+ devices,” in *2011 23rd International Symposium on Computer Architecture and High Performance Computing*, pp. 64–71, IEEE, 2011.
- [74] A. Avilés-González, J. Piernas, and P. González-Férez, “Scalable metadata management through osd+ devices,” *International Journal of Parallel Programming*,

vol. 42, no. 1, pp. 4–29, 2014.

- [75] D. Skeen and M. Stonebraker, “A formal model of crash recovery in a distributed system,” *IEEE Transactions on Software Engineering*, no. 3, pp. 219–228, 1983.
- [76] N.-H. Sun, J. Xing, Z.-G. Huo, G.-M. Tan, J. Xiong, B. Li, and C. Ma, “Dawning nebulae: a petaflops supercomputer with a heterogeneous structure,” *Journal of Computer Science and Technology*, vol. 26, no. 3, pp. 352–362, 2011.
- [77] W. Wu, C.-s. Xie, J.-z. Huang, and C.-f. Zhang, “Highly scalable metadata distribution algorithm in mass storage system,” *Journal of Shanghai University (English Edition)*, vol. 12, no. 5, pp. 419–425, 2008.
- [78] L. Ran and H. Jin, “An efficient metadata management method in large distributed storage systems,” in *Proceedings of the International Conference on Human-centric Computing 2011 and Embedded and Multimedia Computing 2011*, pp. 375–383, Springer, 2011.
- [79] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, “Nfs version 3: Design and implementation,” in *USENIX Summer*, pp. 137–152, Boston, MA, 1994.
- [80] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, and F. D. Smith, “Andrew: A distributed personal computing environment,” *Communications of the ACM*, vol. 29, no. 3, pp. 184–201, 1986.
- [81] J. J. Kistler and M. Satyanarayanan, “Disconnected operation in the coda file system,” in *Proceedings of the thirteenth ACM symposium on Operating systems principles*, pp. 213–225, 1991.
- [82] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, “Coda: A highly available file system for a distributed workstation environment,” *IEEE Transactions on computers*, vol. 39, no. 4, pp. 447–459, 1990.
- [83] M. Nelson, B. Welch, and J. Ousterhout, “Caching in the sprite network file system,” *ACM SIGOPS Operating Systems Review*, vol. 21, no. 5, pp. 3–4, 1987.

- [84] S. V. Anastasiadis, S. Gadde, and J. S. Chase, "Scale and performance in semantic storage management of data grids," *International Journal on Digital Libraries*, vol. 5, no. 2, pp. 84–98, 2005.
- [85] J. Yu, W. Wu, D. Yang, and N. Huang, "Mrfs: A distributed files system with geo-replicated metadata," in *International Conference on Algorithms and Architectures for Parallel Processing*, pp. 273–285, Springer, 2014.
- [86] O. Tatebe, K. Hiraga, and N. Soda, "Gfarm grid file system," *New Generation Computing*, vol. 28, no. 3, pp. 257–275, 2010.
- [87] H. Jin, M. Xiong, S. Wu, and D. Zou, "Replica based distributed metadata management in grid environment," in *International Conference on Computational Science*, pp. 1055–1062, Springer, 2006.
- [88] C. Yi, H. Jin, and Y. Jia, "Metadata management in global distributed storage system," in *Current Trends in High Performance Computing and Its Applications*, pp. 175–184, Springer, 2005.
- [89] W. Xu, Y. Lu, Q. Li, E. Zhou, Z. Song, Y. Dong, W. Zhang, D. Wei, X. Zhang, H. Chen, *et al.*, "Hybrid hierarchy storage system in milkyway-2 supercomputer," *Frontiers of Computer Science*, vol. 8, no. 3, pp. 367–377, 2014.
- [90] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, "{FARSITE}: Federated, available, and reliable storage for an incompletely trusted environment," in *5th Symposium on Operating Systems Design and Implementation (OSDI 02)*, 2002.
- [91] H. Yu, F. Zhang, and Y. Wu, "Granary: A sharing oriented distributed storage system," *Future Generation Computer Systems*, vol. 38, pp. 47–60, 2014.
- [92] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.
- [93] H. Bin and P. Yuxing, "A novel metadata management scheme in cloud computing," in *2010 2nd International Conference on Software Technology and Engineering*, vol. 1, pp. V1–433, IEEE, 2010.

- [94] A. Verma, S. Venkataraman, M. Caesar, and R. H. Campbell, “Scalable storage for data-intensive computing,” in *Handbook of Data Intensive Computing*, pp. 109–127, Springer, 2011.
- [95] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, and A. Chord, “A scalable peer-to-peer lookup service for internet applications,” *Lab. Comput. Sci., Massachusetts Inst. Technol., Tech. Rep. TR-819*, 2001.
- [96] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide-area cooperative storage with cfs,” *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 202–215, 2001.
- [97] D. Yang, W. Wu, Z. Li, J. Yu, and Y. Li, “Ppms: a peer to peer metadata management strategy for distributed file systems,” in *IFIP International Conference on Network and Parallel Computing*, pp. 435–445, Springer, 2014.
- [98] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
- [99] A. Broder, “Network applications of bloom filters: A survey,” in *Allerton Conference, 2002*, 2002.
- [100] R. Anitha and S. Mukherjee, “‘maas’: fast retrieval of data in cloud using metadata as a service,” *Arabian Journal for Science and Engineering*, vol. 40, no. 8, pp. 2323–2343, 2015.
- [101] Y. Hua, H. Jiang, Y. Zhu, D. Feng, and L. Tian, “Semantic-aware metadata organization paradigm in next-generation file systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 2, pp. 337–344, 2011.
- [102] Y. Hua, H. Jiang, and D. Feng, “Real-time semantic search using approximate methodology for large-scale storage systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1212–1225, 2015.
- [103] A. F. Díaz, M. Anguita, H. E. Camacho, E. Nieto, and J. Ortega, “Two-level hash/table approach for metadata management in distributed file systems,” *The Journal of Supercomputing*, vol. 64, pp. 144–155, 2013.

- [104] Q. Xu, R. V. Arumugam, K. L. Yang, and S. Mahadevan, “Drop: Facilitating distributed metadata management in eb-scale storage systems,” in *2013 IEEE 29th symposium on mass storage systems and technologies (MSST)*, pp. 1–10, IEEE, 2013.
- [105] J. Chen, Q. Wei, C. Chen, and L. Wu, “Fsmac: A file system metadata accelerator with non-volatile memory,” in *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–11, IEEE, 2013.
- [106] M. Huang, Z. Liu, L. Qiao, Y. Wang, and Z. Shao, “An endurance-aware metadata allocation strategy for mlc nand flash memory storage systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 4, pp. 691–694, 2015.
- [107] Q. Wei, J. Chen, and C. Chen, “Accelerating file system metadata access with byte-addressable nonvolatile memory,” *ACM Transactions on Storage (TOS)*, vol. 11, no. 3, pp. 1–28, 2015.
- [108] Q. Wei, C. Chen, M. Xue, C. Wang, and J. Yang, “Extending ssd lifetime with persistent in-memory metadata management,” in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 308–311, IEEE, 2016.
- [109] M. Xue, C. Wang, Q. Wei, J. Yang, and C. Chen, “Nvm-accelerated metadata management for flash-based ssds,” in *2016 International Conference on Cloud Computing Research and Innovations (ICCCRI)*, pp. 134–139, IEEE, 2016.
- [110] H. Duan, X. Xiang, Y. Zhang, G. Min, W. Zhan, and P. Lv, “A multi-channel architecture for metadata management in cloud storage systems by binding cpu-cores to disks,” *Concurrency and Computation: Practice and Experience*, vol. 28, no. 16, pp. 4215–4229, 2016.
- [111] P. H. Lensing, T. Cortes, J. Hughes, and A. Brinkmann, “File system scalability with highly decentralized metadata on independent storage devices,” in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 366–375, IEEE, 2016.

- [112] X. Liu, Y. Lu, C. Wu, J. Wu, and Y. Shao, “Ugsd: scalable and efficient metadata management for eb-scale file systems,” in *Proceedings of the International Conference on Compute and Data Analysis*, pp. 81–90, 2017.
- [113] H. Won, M. C. Nguyen, M.-S. Gil, Y.-S. Moon, and K.-Y. Whang, “Moving metadata from ad hoc files to database tables for robust, highly available, and scalable hdfs,” *The Journal of Supercomputing*, vol. 73, no. 6, pp. 2657–2681, 2017.
- [114] J. Wang, X. Zhang, J. Zhang, J. Yin, D. Han, R. Wang, and D. Huang, “Deister: A light-weight autonomous block management in data-intensive file systems using deterministic declustering distribution,” *Journal of Parallel and Distributed Computing*, vol. 108, pp. 3–13, 2017.
- [115] H. Tang, F. Liu, G. Shen, Y. Jin, and C. Guo, “Unidrive: Synergize multiple consumer cloud storage services,” in *Proceedings of the 16th Annual Middleware Conference*, pp. 137–148, 2015.
- [116] G. Wu, F. Liu, H. Tang, K. Huang, Q. Zhang, Z. Li, B. Y. Zhao, and H. Jin, “On the performance of cloud storage applications with global measurement,” in *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, pp. 1–10, IEEE, 2016.
- [117] Y. Gao, X. Gao, X. Yang, J. Liu, and G. Chen, “An efficient ring-based metadata management policy for large-scale distributed file systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 9, pp. 1962–1974, 2019.
- [118] H. J. Singh and S. Bawa, “Scalable metadata management techniques for ultra-large distributed storage systems—a systematic review,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–37, 2018.
- [119] S. Patil and G. Gibson, “Scale and concurrency of {GIGA+}: File system directories with millions of files,” in *9th USENIX Conference on File and Storage Technologies (FAST 11)*, 2011.
- [120] Q. Xu, R. V. Arumugam, K. L. Yong, and S. Mahadevan, “Efficient and scalable metadata management in eb-scale file systems,” *IEEE Transactions on Parallel and*

Distributed Systems, vol. 25, no. 11, pp. 2840–2850, 2013.

- [121] S. Saeed, M. Asim, T. Baker, and Z. Maamar, “A location-sensitive and network-aware broker for recommending web services,” *Computing*, vol. 101, pp. 455–475, 2019.
- [122] R. Wang, Y. Zhang, L. Peng, G. Fortino, and P.-H. Ho, “Time-varying-aware network traffic prediction via deep learning in iiot,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 11, pp. 8129–8137, 2022.
- [123] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30–37, 2009.
- [124] T. Huang, D. Zhang, and L. Bi, “Neural embedding collaborative filtering for recommender systems,” *Neural Computing and Applications*, vol. 32, no. 22, pp. 17043–17057, 2020.
- [125] Y. He, C. Wang, and C. Jiang, “Correlated matrix factorization for recommendation with implicit feedback,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 3, pp. 451–464, 2018.
- [126] M. H. Aghdam, M. Analoui, and P. Kabiri, “Collaborative filtering using non-negative matrix factorisation,” *Journal of Information Science*, vol. 43, no. 4, pp. 567–579, 2017.
- [127] M. Yu, T. Quan, Q. Peng, X. Yu, and L. Liu, “A model-based collaborate filtering algorithm based on stacked autoencoder,” *Neural Computing and Applications*, vol. 34, no. 4, pp. 2503–2511, 2022.
- [128] G. Zhang, Y. Liu, and X. Jin, “A survey of autoencoder-based recommender systems,” *Frontiers of Computer Science*, vol. 14, no. 2, pp. 430–450, 2020.
- [129] J. Zhao, X. Geng, J. Zhou, Q. Sun, Y. Xiao, Z. Zhang, and Z. Fu, “Attribute mapping and autoencoder neural network based matrix factorization initialization for recommendation systems,” *Knowledge-Based Systems*, vol. 166, pp. 132–139, 2019.

- [130] H. Wang, X. Shi, and D.-Y. Yeung, “Relational stacked denoising autoencoder for tag recommendation,” in *Twenty-ninth AAAI conference on artificial intelligence*, 2015.
- [131] M. De Bois, M. A. El Yacoubi, and M. Ammi, “Adversarial multi-source transfer learning in healthcare: Application to glucose prediction for diabetic people,” *Computer Methods and Programs in Biomedicine*, vol. 199, p. 105874, 2021.
- [132] M. De Bois, M. A. E. Yacoubi, and M. Ammi, “Glyfe: review and benchmark of personalized glucose predictive models in type 1 diabetes,” *Medical & Biological Engineering & Computing*, vol. 60, no. 1, pp. 1–17, 2022.
- [133] M. De Bois, M. A. E. Yacoubi, and M. Ammi, “Prediction-coherent lstm-based recurrent neural network for safer glucose predictions in diabetic people,” in *International Conference on Neural Information Processing*, pp. 510–521, Springer, 2019.
- [134] M. Alloghani, T. Baker, D. Al-Jumeily, A. Hussain, A. Kaky, and J. Mustafina, “Early detection and prediction of lung cancer using machine learning algorithms applied on a secure healthcare data system architecture,” *Machine Learning for Computer and Cyber Security: Principle, Algorithms, and Practices*, pp. 233–257, 2019.
- [135] R. K. Tripathy, D. K. Dash, S. K. Ghosh, and R. B. Pachori, “Detection of different stages of anxiety from single channel wearable ecg sensor signal using fourier-bessel domain adaptive wavelet transform,” *IEEE Sensors Letters*, 2023.
- [136] S. S. Rathore and S. Kumar, “An empirical study of some software fault prediction techniques for the number of faults prediction,” *Soft Computing*, vol. 21, pp. 7417–7434, 2017.
- [137] Z. Ji, H. Pi, W. Wei, B. Xiong, M. Woźniak, and R. Damasevicius, “Recommendation based on review texts and social communities: a hybrid model,” *IEEE Access*, vol. 7, pp. 40416–40427, 2019.

- [138] W. Wei, Z. Wang, C. Fu, R. Damaševičius, R. Scherer, and M. Woźniak, “Intelligent recommendation of related items based on naive bayes and collaborative filtering combination model,” in *Journal of Physics: Conference Series*, vol. 1682, p. 012043, IOP Publishing, 2020.
- [139] A. Pujahari and D. S. Sisodia, “Pair-wise preference relation based probabilistic matrix factorization for collaborative filtering in recommender system,” *Knowledge-Based Systems*, vol. 196, p. 105798, 2020.
- [140] A. Pujahari and D. S. Sisodia, “Handling dynamic user preferences using integrated point and distribution estimations in collaborative filtering,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 10, pp. 6639–6651, 2022.
- [141] A. Pujahari and D. S. Sisodia, “Aggregation of preference relations to enhance the ranking quality of collaborative filtering based group recommender system,” *Expert Systems with Applications*, vol. 156, p. 113476, 2020.
- [142] A. Pujahari and D. S. Sisodia, “Model-based collaborative filtering for recommender systems: An empirical survey,” in *2020 First International Conference on Power, Control and Computing Technologies (ICPC2T)*, pp. 443–447, IEEE, 2020.