International Journal of Advanced Smart Convergence Vol.13 No.3 125-134 (2024)
http://dx.doi.org/10.7236/IJASC.2024.13.3.125

IJASC 24-3-14

Fabricator based on B+Tree for Metadata Management in Distributed Environment

Chae-Yeon Yun*, Seok-Jae Moon**

** The master's course, Graduate School of Smart Convergence, KwangWoon University, Seoul, Korea*
*** Professor, Graduate School of Smart Convergence, KwangWoon University, Seoul, Korea*
E-mail: {dbscodus, msj8086}@kw.ac.kr

Abstract

In a distributed environment, data fabric refers to the technology and architecture that provides data management, integration, and access in a consistent and unified manner. To build a data fabric, it is necessary to maintain data consistency, establish a data governance system, reduce structural differences between data sources, and provide a unified view. In this paper, we propose the Fabricator system, a technology that provides data management and access in a consistent and unified manner by building a metadata registry. Fabricator manages the addition and modification of metadata schemas and matching processes by designing a matching tool called MetaSB Manager that applies B+Tree. This allows real-time integration of various data sources in a distributed environment, maximizing the flexibility and usability of data.

Keywords: *B+Tree, BigData, Cloud, Data Fabric, Data Integration, Distributed Environment, Metadata*

1. INTRODUCTION

In a distributed environment, a data fabric is a technology that integrates various data sources and formats to provide consistent data management and access [1,2]. This helps companies effectively manage and utilize data in distributed data environments to derive insights and maximize business value [3]. To build a data fabric, it is important to maintain data consistency and establish a data governance system when integrating data sources owned by geographically dispersed companies. Designing a metadata-based schema for all data sources can be a very challenging task, and the cleanup process is particularly difficult [4]. It is necessary to analyze the schemas of each data source and match them with the integrated schema to provide an integrated view, and also pay attention to the performance and matching quality of metadata schema management tools [5].

In this paper, we build a metadata registry to centrally manage the metadata of each data source, improve performance using the B+Tree index structure, and check the structure of the schema before matching to enable

Manuscript Received: July. 16. 2024 / Revised: July. 21. 2024 / Accepted: July. 27. 2024
Corresponding Author: msj8086@kw.ac.kr
Tel: 02-940-8283, Fax: 02-940-5443
Author's affiliation: Professor, Graduate School of Smart Convergence, KwangWoon University, Seoul, Korea

faster addition of new metadata or updates. To demonstrate performance, we designed a matching tool called MetaSB Manager and describe its architecture component by component. The MetaSB Manager proposes a structure that stores metadata schema elements in a repository by applying an index using B+Tree logic. The structure of this paper is as follows. Chapter 2 describes related research, Chapter 3 describes the configuration of the B+Tree-based Fabricator model for metadata management in a distributed environment, Chapter 4 describes the application case of the system and comparative analysis with other systems. And finally, Chapter 5 describes the conclusion and future research.

2. RELATED WORK

The data fabric architecture is a framework that integrates distributed data sources and heterogeneous data into a consistent interface to provide users [6, 7], and in this process, the role of metadata is very important. According to several studies, metadata management in a distributed environment plays a crucial role in maintaining consistency in data sources, structures, and transformation rules. The main challenges of metadata management are the diversity and complexity of metadata, frequent schema changes, and maintaining consistency in distributed data sources. The global schema is an important component that integrates distributed data sources to provide a consistent data view [8]. However, if there is a failure in the global schema component, the entire system can be affected. The main challenges of the global schema include complexity and the absence of a rapid recovery strategy in case of failure. Fault-tolerant mechanisms such as duplexing and backup systems are proposed, and these strategies can increase the stability of the system. B+Tree is a tree structure that can efficiently manage large volumes of data in databases and file systems, with insertion, deletion, and search operations having a logarithmic time complexity [9]. Various techniques such as node splitting, page size adjustment, and caching have been proposed for performance optimization. In a distributed system, B+Tree ensures data scalability and high availability, and research is ongoing in distributed storage, consistency maintenance, and fault recovery. The potential of B+Tree is further expanded through integration with cloud environments and NoSQL databases.

3. PROPOSED SYSTEM

3.1. System Overview

In this paper, we propose a B+Tree-based Fabricator design system for metadata management in a distributed environment. The Fabricator design system consists of two layers.

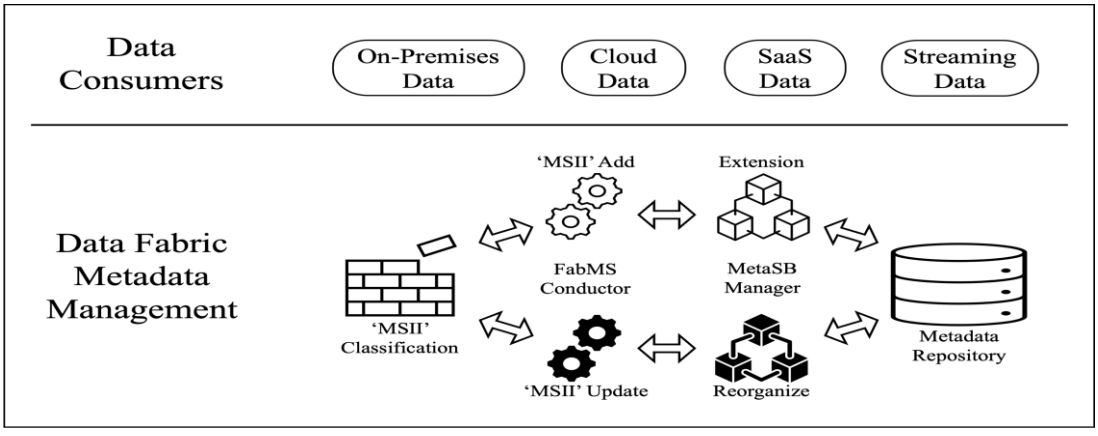

Figure 1. Fabricator Overview

The first layer, the Data Consumers layer, refers to applications or services that use the metadata generated by Fabricator, and requests and retrieves the metadata.

- On-Premises Data: This data is managed on local servers or data centers located within a company, and is directly managed within the company, making it easy to secure and control.

- Cloud Data: This data is managed by cloud service providers and is stored and managed by companies or individuals who use cloud services.

- SaaS Data: This data is managed by a specific service provider and is stored and managed by companies or individuals who use SaaS services.

- Streaming Data: This is data that occurs in real-time and continuously enters, and streaming technology is used for data that needs to be processed in real-time.

The second layer, the Data Fabric Metadata Management layer, aims to effectively manage the metadata of various Data Sources in a data fabric environment, allowing data consumers to provide the information they need accurately and quickly.

- 'MSII' Classification: Data is classified and delivered to Fabricator.

- FabMS Conductor: This central module consists of 'MSII' Add and 'MSII' Update, which create new metadata items or add existing metadata based on the classified data.

- MetaSB Manager: It is in charge of overall management of the entire metadata and interacts with the Extension and Reorganize modules when it needs to reconstruct the metadata.

- Metadata Repository: This is the database that ultimately stores all metadata.

3.2. Metadata Schema Management Applying B+Tree

Metadata is information that describes the structure, attributes, and relationships of data. As shown in Figure 2, "SyncResearcherInformation" consists of "dataArea," "extension," "researcher," "profile," "activity," and "publication," and in "UpdateResearcherInformation," "extension" is deleted and "reorganize" is added.

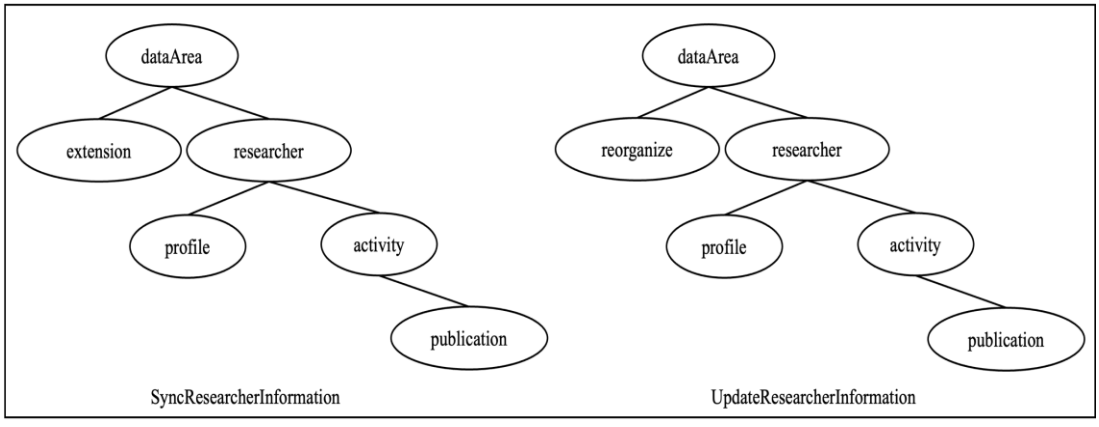

Figure 2. Examples of Subset from 'MSII' Schemas

In this paper, the metadata attribute (name, data type, size, etc.) of the metadata is used as the B+Tree key value for indexing. In addition, the hierarchical structure of B+Tree can be used to represent the various

relationships and hierarchies of the metadata. This allows complex metadata structures to be accessed and managed effectively. The data uses the search algorithm of B+Tree, starting from the root node based on the search key, moving to the appropriate child node, and then returning the metadata item corresponding to the key when it reaches the leaf node. Figure 3 shows the path of each node used as an index in B+Tree. The path is represented by a sequence of numbers corresponding to the position of each node in the hierarchical structure.

Nodes	*Path of SyncResearcherInformation*	*Path of UpdateResearcherInformation*
dataArea	1 / 1	2 / 1
extension	1 / 1 / 2	
reorganize		2 / 1 / 2
researcher	1 / 1 / 3	2 / 1 / 3
profile	1 / 1 /3 / 4	2 / 1 / 3 / 4
activity	1 / 1 / 3 / 5	2 / 1 / 3 / 5
publication	1 / 1 / 3 / 5 / 6	2 / 1 / 3 / 5 / 6

Figure 3. The paths of each node used as indexes in the B+Tree

For example, in the case of the "publication" node, the path in the "SyncResearcherInformation" schema is "1/1/3/6," which indicates that the "publication" node is the sixth child of the third child of the first child of the first node. In the "UpdateResearcherInformation" schema, the path is "2/1/3/6," which indicates that the "publication" node is the sixth child of the third child of the first child of the second node. Figure 4 represents the node structure when the metadata was initially inserted.

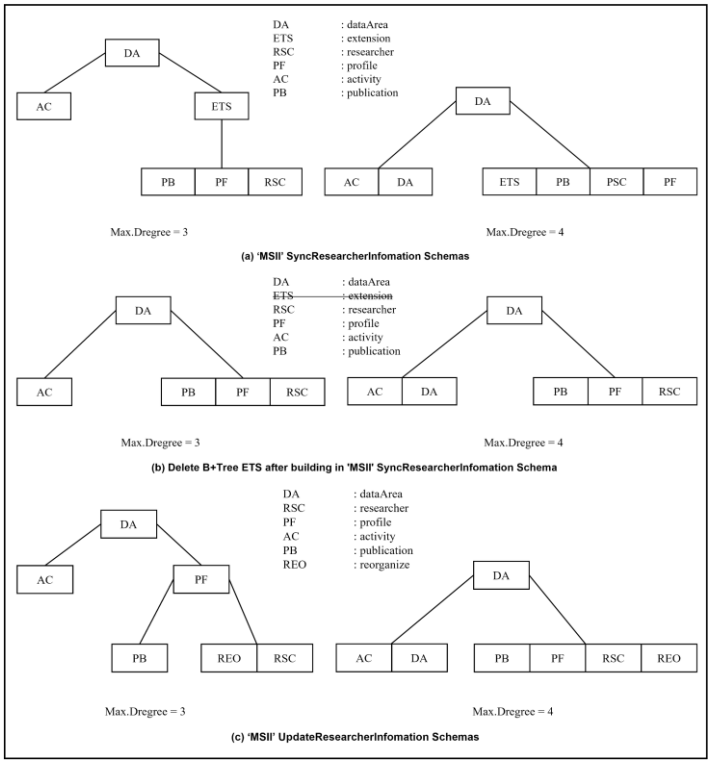

Figure 4. The B+Tree after building it from 'MSII' Schemas

In Figure 4(a), in step 1, the "AC (activity)" metadata is inserted first, followed by the insertion of "DA (dataArea)" in step 2. Then, in step 3, "ETS (extension)" is inserted. In step 4, "PB (publication)" is inserted, and node splitting occurs in a node with a maximum degree of 3, with "DA" becoming the root node, and AC, ETS, and PB are divided into sub-nodes. In step 5, "PF (profile)" is inserted, and in the final step 6, "RSC (researcher)" is inserted, and node splitting occurs in a node with a maximum degree of 3, with PB, PF, and RSC divided into sub-nodes of ETS. Figure 4(b) represents the node structure that changes when the ETS metadata is removed. A node with a maximum degree of 4 is simply deleted, but in a node with a maximum degree of 3, ETS is removed, and the node including PB, PF, and RSC becomes an orphan, so it is merged into the new parent node, the DA node, and repositioned. Figure 4(c) represents the nodes that change when a new metadata "REO (reorganize)" is inserted. In a node with a maximum degree of 3, REO is inserted into the PB, PF, and RSC nodes, but this exceeds the maximum degree, so the intermediate key PF is promoted to the root, and PB and REO, RSC are divided into two new nodes.

3.3. System Flow Mechanism

Figure 5 visually shows how the ADD system of MetaSB Manager connects from the 'MSII' Classification layer to the Metadata Repository layer.

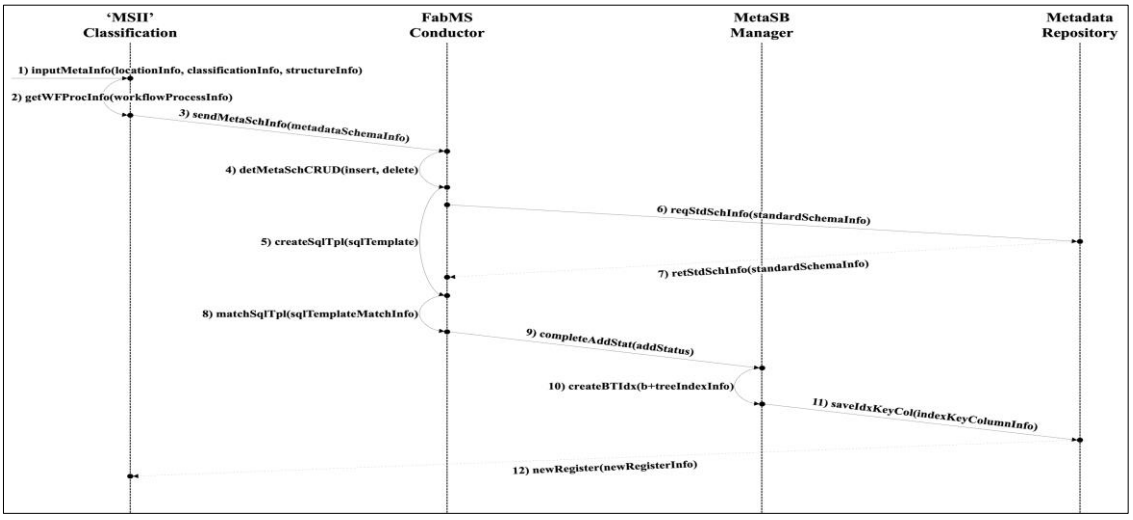

Figure 5. MetaSB Manager-ADD system detailed design flow chart

(1) inAdminMetaInfo(): This starts in the 'MSII' Classification, and is the step where the administrator enters metadata information, which is the process of registering the metadata information needed by the system. Management meta information refers to the metadata of the data that the system will process.

(2) getWFProcInfo(): This retrieves workflow process information from the 'MSII' Classification. This information includes information related to the flow of work, allowing the system to check the necessary workflow-related information.

(3) sendMetaSchInfo(): This is the step of sending metadata schema information from the 'MSII' Classification to FabMS Conductor, delivering the previously entered metadata information to the system. Metadata schema information defines the structure and format of the data.

(4) detMetaSchCRUD(): This determines CRUD (Create, Read, Update, Delete) operations on metadata

schema information in FabMS Conductor. It provides basic CRUD functions for managing metadata information.

(5) createSqlTpl(): This creates SQL templates in FabMS Conductor. This template defines the basic structure of SQL queries to be used in the database and generates the necessary SQL queries based on the metadata information.

(6) reqStdSchInfo(): MetaSB Manager requests standard schema information from the Metadata Repository that is needed by the system. Standard schema information refers to the standard format of the database structure.

(7) retStdSchInfo(): This is the step of returning standard schema information from the Metadata Repository, providing the previously requested standard schema information to the system. MetaSB Manager performs subsequent operations based on this information.

(8) matchSqlTpl(): This matches the SQL templates generated in FabMS Conductor with the standard schema information. It is a task to check the relationship between the generated SQL template and the standard schema.

(9) completeAddStat(): This is the step of completing the status of the ADD operation in MetaSB Manager. It informs the system that all previous operations have been successfully completed.

(10) createBTIdx(): This creates a B+Tree index in MetaSB Manager. B+Tree indexing is one of the database indexing methods used to increase search speed.

(11) createIdxKeyCol(): This creates index key columns in the Metadata Repository. This is a task to add index keys to the database table to improve data search performance, defining the key columns necessary for index creation.

(12) newRegister(): This registers a new register in the 'MSI' Classification. You can register a new data item in the system.

Figure 6 visually shows how the UPDATE system of MetaSB Manager connects from the 'MSII' Classification layer to the Metadata Repository layer.

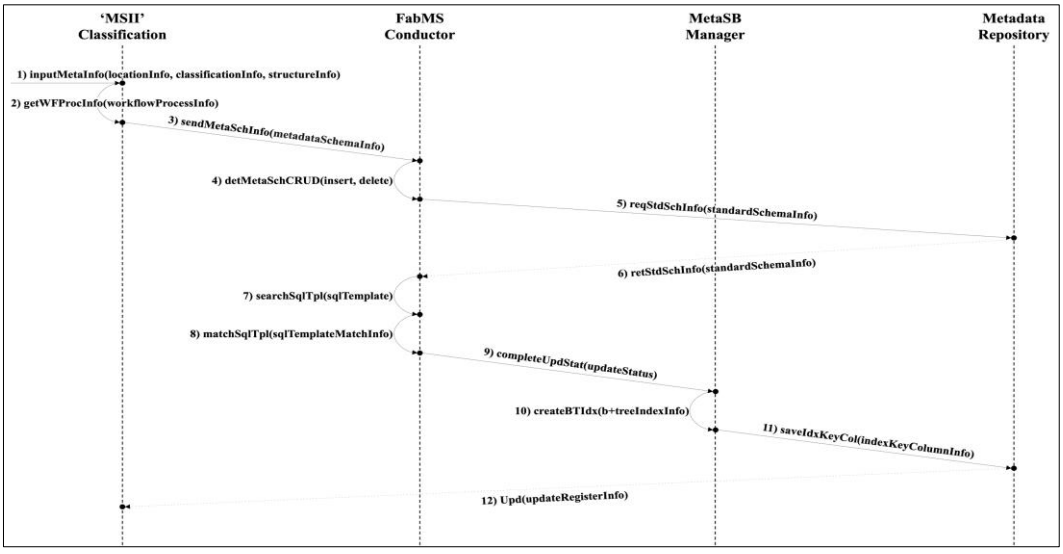

Figure 6. MetaSB Manager-UPDATE system detailed design flow chart

(1) inAdminMetaInfo(), (2) getWFProcInfo(), (3) sendMetaSchInfo(), (4) detMetaSchCRUD(), (5) reqStdSchInfo(), (6) retStdSchInfo(), 11) saveIdxKeyCol(): It is the same as MetaSB Manager-ADD system.

(7) searchSqlTpl(): In FabMS Conductor, it searches for SQL templates. This template defines the basic structure of SQL queries to be used in the database and searches for the necessary SQL queries based on the metadata information.

(8) matchSqlTpl(): It matches the SQL templates generated in FabMS Conductor with the standard schema information. It checks the relationship between the generated SQL template and the standard schema.

(9) completeUpdStat(): This is the step of completing the status of the UPDATE operation in MetaSB Manager. It informs the system that all previous operations have been successfully completed.

(10) createBTIdx(): MetaSB Manager creates a B+Tree index. B+Tree indexing is one of the database indexing methods used to increase search speed.

(12) Upd(): This registers additional register information in the 'MSII' Classification. You can register a new data item in the system.

4. COMPARATIVE ANALYSIS

4.1. Experiments And Results

The performance evaluation in this paper was conducted in a virtual environment with the same conditions, with the number of metadata schemas set to 20, 40, 80, 160, and 320.

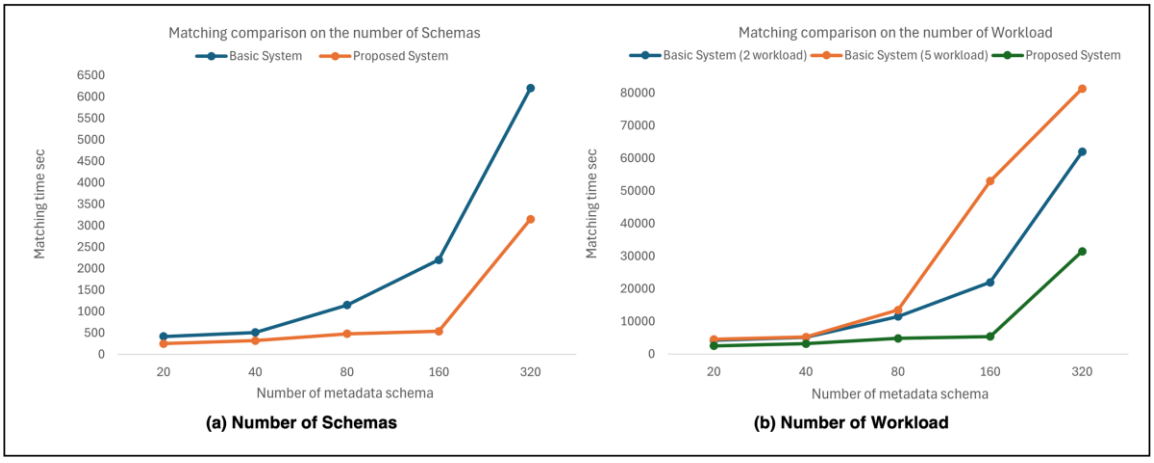

Figure 7. Matching comparison on the number of Schemas and Workload

In Figure 9(a), the system aimed at improving the matching time workload-based process in a specific company's workload compared the results of the matching time based on the presence or absence of the proposed system. The depth of the schema documents in Figure 9(a) ranges from 8 to 21, and the number of elements in each metadata schema ranges from 1900 to 6300. When the number of schemas is 20, the proposed system significantly reduces the matching time from 420 sec to 250 sec compared to the existing system. When the number of schemas is 40, the proposed method is still 320 sec faster than the existing system. At 80 schemas, the proposed system has a matching time of 480 sec, which is significantly increased by 1150 sec compared to the existing system, indicating a large gap. And when the number of schemas is 160, the proposed system

continues to perform better with a matching time of 540 sec compared to the existing system's 2200 sec. At 320 schemas, the proposed system is still efficient with 3150 sec, while the existing system is 6200 sec, indicating that the performance gap narrows compared to when the number of schemas is smaller, but it is still significant. In Figure 9(b), the proposed system compares the matching time of the schema matching based on the workload 2 and 5, where the proposed system is applied and not applied. For 20 schemas, the proposed system significantly reduces the matching time from 4200 sec and 4500 sec to 2500 sec, showing a significant improvement. As the number of schemas increases, the proposed system is 3200 sec faster than the other two existing systems at 40 schemas, 5100 sec and 5200 sec faster. At 80 schemas, the proposed system continues to increase with 4800 sec, while the two existing systems are 11500 sec and 13500 sec. At 160 schemas, the proposed system shows a significant efficiency improvement with 5400 sec compared to the two systems' 22000 sec and 53000 sec. At 320 schemas, the proposed system is still efficient with 31500 sec, while the two existing systems are 62000 sec and 81350 sec, indicating that the performance gap narrows slightly, but still shows a significant improvement. This suggests that the proposed system consistently performs better overall across all tested numbers of schemas, indicating that it is more efficient and better scalable as the number of schemas increases. The efficiency improvement of the proposed system is particularly noticeable when the number of schemas is large, demonstrating its ability to handle larger datasets more effectively.

4.2. Comparison of Systems

In Table 1, we compared the Binary Search Tree (BST) [9], Hash Table, and the proposed system based on items such as data integration method, data processing process, data access method, data structure, scalability, and application field.

Table 1. Comparison of Systems

	BST [10]	Hash Table	Proposed System
Data Integration	Store data in leaf nodes	Distribution according to hash function	Store data in leaf nodes
Data Access	Fast search with binary search	Fast access based on hash function	Sequential access possible (Easy to look up range)
Data Processing	Better than typical BST	Insertion and deletion operations are fast	Efficient relocation during insertion and deletion
Scalability	Suitable for typical small-scale data	Depends on conflict resolution	Suitable for large datasets
Application Field	Sorted data search	Memory management, caching	Database, File System

First, BST stores data in leaf nodes and allows for fast search through binary search. Data processing is superior to general BSTs and is generally suitable for small-scale data. It is mainly used for searching sorted data. Second, Hash Table distributes data according to a hash function and can be accessed quickly through a hash function. Insertion and deletion operations are fast, and scalability varies depending on the collision resolution method. It is mainly used for memory management and caching. Finally, the proposed system, like

BST, stores data in leaf nodes and allows for sequential access, making it easy to query ranges. Efficient relocation is possible during insertion and deletion, and it is suitable for large datasets. It is mainly used in databases and file systems. In other words, BST is suitable for searching sorted data, Hash Table is useful for fast access and memory management, and the proposed system is efficient for range queries and large datasets.

5. CONCLUSION

In this paper, we propose a B+Tree-based Fabricator design system for managing metadata in a distributed environment. To effectively match and improve performance in managing metadata schemas during data integration in companies, we utilized the B+Tree index structure. Fabricator focuses on unstructured metadata and provides high-speed processing of large volumes of data. In this paper, Fabricator can integrate various data sources in real-time to maximize the flexibility and usability of data. In the future, it is necessary to address high performance and optimization issues to support real-time data processing and analysis, and to consider the cost of building and maintaining data fabric accordingly.

ACKNOWLEDGMENT

This paper was supported by the KwangWoon University Research Grant of 2024.

REFERENCES

[1] Patel and N. C. Debnath, Data Science with Semantic Technologies. CRC Press, pp. 267-286, 2023.
DOI: https://doi.org/10.1007/978-3-658-12225-6_4

[2] Underwood, “Continuous Metadata in Continuous Integration, Stream Processing and Enterprise DataOps,” Data Intelligence, vol. 5, no. 1. MIT Press, pp. 275–288, 2023.
DOI: https://doi.org/10.1162/dint_a_00193

[3] Li, M. Yang, X. Xia, K. Zhang, and K. Liu, “A Distributed Data Fabric Architecture based on Metadate Knowledge Graph,” 2022 5th International Conference on Data Science and Information Technology (DSIT). IEEE, Jul. 22, 2022.
DOI: https://doi.org/10.1109/DSIT55514.2022.9943831

[4] Liu, M. Yang, X. Li, K. Zhang, X. Xia, and H. Yan, “M-Data-Fabric: A Data Fabric System Based on Metadata,” 2022 IEEE 5th International Conference on Big Data and Artificial Intelligence (BDAI). IEEE, Jul. 08, 2022.
DOI: https://doi.org/10.1109/BDAI56143.2022.9862807

[5] V. Sharma, B. Balusamy, J. J. Thomas, and L. G. Atlas, Eds., “Data Fabric Architectures.” De Gruyter, May 04, 2023.
DOI: https://doi.org/10.1515/9783111000886

[6] C.-Y. Yun and S.-J. Moon, “A Fabricator Design for Metadata CI/CD in Data Fabric,” International Journal of Internet, Broadcasting and Communication, vol. 16, no. 2, pp. 193–202, May 2024.
DOI: https://doi.org/10.7236/IJIBC.2024.16.2.193

[7] F. Nawab and M. Sadoghi, “Consensus in Data Management: From Distributed Commit to Blockchain,” Foundations and Trends® in Databases, vol. 12, no. 4. Now Publishers, pp. 221–364, 2023.
DOI: https://doi.org/10.1561/1900000075

[8] J. Liang, D. Hu, and J. Feng, “Do We Really Need to Access the Source Data? Source Hypothesis Transfer for Unsupervised Domain Adaptation.” arXiv, 2020.
DOI: https ://doi.org/10.48550/arXiv.2002.08546

[9] Kim, Seon Hwan and Kwak, Jong Wook, “Garbage Collection Method using Proxy Block considering Index Data Structure based on Flash Memory,” Journal of the Korea Society of Computer and Information, vol. 20, no. 6, pp. 1–11, Jun. 2015.

DOI: https://doi.org/10.9708/JKSCI.2015.20.6.001

[10] P. Chalermsook, M. Goswami, L. Kozma, K. Mehlhorn, and T. Saranurak, “Multi-finger binary search trees.” arXiv, 2018.
DOI: https://doi.org/10.48550/ARXIV.1809.01759