International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 11
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 November 2023*

Enhancing Xml Data Retrieval Performance with Clustering and Indexing

Wanjari Ravindra Shankar, Dr. F. Rahman

Department of Computer Science, Kalinga University, Naya Raipur,
Chhattisgarh, India

ABSTRACT

Using SQL Server 2005 and Berkeley DBXML (BDBXML) as case studies, this research assesses how well a similarity-based clustering method retrieves XML data. If you're using SQL Server 2005, you can speed up data retrieval speeds by combining clustering and indexing. Applying clustering and indexing simultaneously, for instance, decreases the retrieval time for 10,000 items from 0.88 seconds to 0.46 seconds. The most efficient retrieval for 10,000 entries utilizing both clustering and indexing was 2.389 seconds in BDBXML, which demonstrates better retrieval speeds overall. With BDBXML demonstrating quicker retrieval speeds than SQL Server 2005 for big datasets, the results demonstrate that indexing and clustering improve performance. For big datasets utilized in similarity-based clustering tasks, this study shows that improving XML data retrieval by clustering and indexing is effective.

Keywords: Similarity, Indexing, Retrieval, Database, Server

I. INTRODUCTION

The exponential growth of data in the digital era calls for better methods of storing, retrieving, and managing this information. Because of its adaptability, platform freedom, and hierarchical structure that allows nested components, XML (eXtensible Markup Language) has become a commonly used format for storing and delivering data. From online marketplaces and content management systems to databases used in science and web services, XML's widespread use as a data representation standard is clear. But when XML databases get bigger, data retrieval efficiency becomes a major issue, particularly for complicated queries that demand rapid processing.

Because XML is structured like a tree, there are several peculiarities that make XML data retrieval performance difficult. Tables are the standard data organization method in relational databases, and indexing techniques tailored to tabular data allow for faster retrieval. Unfortunately, conventional indexing methods aren't very good at handling XML because of its tree-like structure, parent-child connections, nested tags, and data nodes that are rich in attributes. One possible cause of increased retrieval time while working with XML documents is the use of path-based searches to traverse nested levels. Additionally, sophisticated methods beyond sequential search are required for computationally costly XML file or dataset processing in the absence of an effective retrieval approach. Here, indexing and clustering algorithms, which are optimized for XML's structure, come into play as performance-enhancing strategies.

One of the best ways to improve XML data retrieval is by indexing. The goal of XML indexing structures is to reduce computing costs and retrieval times by minimizing the traversal of unnecessary parts of the data tree. There are a few common ways to index XML documents: element, path, and content-based indexing. While route indexing maximizes retrieval according to certain hierarchical paths inside the XML structure, element indexing is concerned with indexing particular tags or elements. For documents with a lot of text, another method that might be helpful is content-based indexing, which looks at the text and attributes of XML nodes. For XML, indexes like as B-trees, hash-based indexes, and inverted indexes have been modified to support intricate hierarchical queries. As an example, route indexes can improve the efficiency of retrieving nodes along specified paths, leading to better query performance for nested data. In order to efficiently handle complicated XPath and XQuery queries—typically used to traverse XML structures and retrieve particular data nodes—XML databases make use of these specialized indexes.

By combining XML documents or fragments that are structurally or semantically similar, clustering is another effective method for improving the performance of XML data retrieval. Faster data retrieval is possible with clustering because it reduces the amount of time spent looking through irrelevant data by limiting the search space to selected clusters. Structure-based and content-based clustering are the two main types of XML clustering methods. Data in XML may be organized using structure-based clustering if the documents have common structures, including tag patterns or hierarchies. When dealing with XML datasets that have consistent formats, such medical records or online store product listings, this clustering method works well. When dealing with XML documents that have different structures but comparable content, such news items or research papers, content-based clustering is effective since it clusters XML data according to similar textual or attribute content. Further optimization of performance has also been achieved through the development of hybrid clustering algorithms that combine structure-based and content-based clustering. By focusing searches on the most important clusters and excluding irrelevant data, these clustering approaches speed up retrieval.

There is encouraging evidence that combining clustering and indexing methods can improve the efficiency of XML data retrieval. Retrieval systems can improve query response speeds, even for massive XML datasets, by building efficient cluster indexes or by utilizing clustered indexes that blend the best of both worlds. As an example, XML documents can be grouped together according to their structural similarities. This enables route indexing inside the clusters, which simplifies the navigation of comparable data structures by eliminating repetitive calculations. In addition, indexed clusters provide distributed or simultaneous querying of huge datasets by distributing the retrieval load across the system. Online product catalogs, digital libraries, and real-time data monitoring systems are just a few examples of applications that might greatly benefit from this clustering and indexing combination strategy. Here, indexing improves retrieval inside the chosen clusters while clustering limits the search scope, guaranteeing efficiency and scalability.

Challenges in XML Data Retrieval

1. Complexity of XML Document Structure

The complexity and depth of nesting in XML documents makes data retrieval a tough task. Schemas and several levels of nested components are commonplace in XML data, in contrast to more flat data structures such as tables. To correctly travel between nodes and obtain data, specific parsing and querying techniques are required because to the complexity. Misunderstanding or unsuccessful queries might result from little structural changes, thus it's crucial to understand the actual structure.

2. Lack of Standardization in XML Schemas

XML documents are very adaptable; even when used with the same data type, they can have vastly different structures and schemas. Due to the absence of standards, reliable data retrieval becomes problematic when using distinct XML files to represent same data in varying forms. In order to reconcile differing XML document structures caused by schema variances, processing time and complexity are increased through the use of specialized parsing or transformations.

3. Performance Overhead in Parsing and Processing

The verbose and hierarchical structure of XML data makes its parsing resource-intensive, particularly for big pages. Particularly when employing DOM parsers, which store the complete document in memory, processing XML documents can put a burden on CPU and memory resources. Particularly when searching or querying across several nodes or huge document collections, this might cause performance bottlenecks for large datasets.

4. Limited Query Capabilities with XPath and XQuery

Parsing XML data, especially for large pages, is resource-intensive due to its verbose and hierarchical nature. Processing XML documents may be taxing on CPU and memory resources, especially when using DOM parsers that hold the entire content in memory. Performance bottlenecks for large datasets could be caused by this, especially when searching or querying across numerous nodes or massive document collections.

5. Data Redundancy and Redundant Storage

Due to XML's verbosity, data redundancy—the occurrence of repeated tags and attributes—is a common cause of file size increases. This duplication is essential for human comprehension but wasteful for large-scale data storage and retrieval operations. It may take more time and more computing resources to retrieve data from larger files since read/write operations are slower.

6. Handling Mixed Content and Textual Data

It's not uncommon for XML files to have a mix of text, elements, and attributes. Because text and element nodes might coexist, retrieving mixed material properly is difficult. Data retrieval becomes more complicated and error-prone when parsing such files for text-only or structured-only information or when using complex parsing methods.

7. Scalability Issues with Large XML Files

The amount of time and computing power needed to parse and query XML files grows linearly with their size. Systems that weren't made to manage large XML files may struggle to process them, which can lead to slowdowns, timeouts, and memory shortages. Partitioning or streaming methods are frequently necessary for effective data retrieval from these files, which might make the retrieval process more complicated.

8. Limited Support for XML in Relational Databases

Storage and retrieval issues with XML data are common in many older relational databases since these databases either do not support it natively or only allow it in restricted ways. To fit into relational databases, XML data is typically processed and flattened, removing part of the hierarchical structure. Text blobs are another option for storing XML data; however, successful querying of textual blobs necessitates further indexing or parsing, which makes retrieval more complicated.

9. Difficulty in Version Control and Data Consistency

It gets more difficult to keep consistent data across versions in dynamic systems with often changing XML data. It is difficult to monitor modifications and guarantee data consistency in XML documents due to the absence of native version control tools. When dealing with several XML versions, it's common to need to provide bespoke handling. This increases the risk of retrieval failures caused by data structures that are obsolete or inconsistent.

10. Security and Privacy Concerns

Security flaws in XML documents can allow unauthorized parties to access data through exploits like XML External Entity (XXE) attacks. Sanitizing inputs and limiting characteristics, such as external entity references, is a common practice for securing XML data for retrieval. The extra procedures needed to authenticate and filter the data might make data retrieval more complicated, but implementing these security measures is necessary.

<u>Performance Metrics for XML Data Retrieval</u>

In order to comprehend how efficient clustering and indexing methods are, it is crucial to assess XML retrieval performance. Typical measures of performance consist of:

1. **Query Response Time**: Finds out how long it takes to get results from a search. Faster retrieval processes are indicated by shorter response times.

2. **Scalability**: Determines if the system can sustainably process larger data sets without suffering noticeable performance degradation. Examining how well clustering and indexing handle increasing document sizes and query loads is an important part of scalability testing.

3. **Indexing and Clustering Overhead**: Counts the extra effort and materials needed to construct and update indexes and clusters. In order to be effective, systems must provide fast retrieval with little overhead.

4. **Accuracy and Precision**: The retrieval system's ability to fulfill query criteria is assessed by these measures. The accuracy of clustering shows how well documents are grouped together, while the precision of indexing shows how relevant the results are.

II. REVIEW OF LITERATURE

Klaib, Alhadi. (2021) One of the most important technologies for data transport in the Internet age is Extensible Markup Language, or XML. When dealing with XML data, XML labeling systems are a must-have tool. To label XML data, one must first give labels to each node in the XML document. In order to overcome some of the problems associated with indexing XML data, a hybrid labeling method called CLS was created. Additional use cases for datasets include evaluating XML tagging systems. Nowadays, you may find a plethora of XML datasets to choose from. There are some that come from actual datasets and others that are made up. To evaluate the XML tagging techniques, these datasets and benchmarks are utilized. In order to choose the best dataset and benchmark to evaluate the CLS labelling system, this study examines and takes into account all of them, along with their requirements. Based on the results of this study, the XMark benchmark is the best option for evaluating the CLS labeling scheme's performance.

Lianghui, Cai. (2020) XML, the Comprehensive Markup Language Although Xml takes up more room than binary data, it is incredibly easy to learn and use, and it uses a simple set of tags to describe data in a way that's both convenient and easy to understand. Web Information Retrieval System Structure for Professional Content Design and XML-Organized Management of Distributed Web Information Resources Lets Users Get Their Hands on Content-Specific Data. The Features of an Ideal Data Storage Format: Scalability, High Level of Structure, and Simplicity in Network Transmission Great Performance Is the Determinant of Xml. Library and information retrieval professionals can learn more about XML and its uses in this paper.

Saranya & Zoraida, B.S.E. (2016) to meet consumer demand, it is more vital to extract useful information from massive amounts of online resources nowadays. Semantic information retrieval relies on XML and RDF documents to decipher user queries and retrieve relevant data. Easy value and data interchange and structured information sharing are two benefits of XML documents' lightweight coding and logical structure. To make XML information retrieval more efficient, a lot of mining methods and approaches are applied. Data items that are similar to one another can be preprocessed using methods like classification (Supervised Learning) and clustering (Unsupervised Learning). The research on three clustering algorithms and their similarity metrics on XML datasets is presented in this work. These methods include k-means, EM, and Tree Clustering. Using the identical XML datasets, we analyze and evaluate the three clustering methods to determine which one is the most effective in clustering XML documents.

Desoki, Rehab & Elfatatry, Ahmed. (2014) A difficult research topic is searching huge XML libraries. Prior to doing a search, clustering a huge repository greatly improves the search performance. Clustering is a method that divides a search area into more manageable XML collections. Our work here introduces a mechanism for improved XML clustering by structure. In addition, we provide a novel XML structure representation that preserves all XML structural properties independent of summarization. After that, we compare our enhanced method's search results to those of the SAXON and Qizx XML XQuery processors as a benchmark. Examining search processing times and results accuracy with varying dataset sizes for homogeneous and heterogeneous XML documents is the main focus of the comparison. Improved accuracy at the same performance level is demonstrated by the obtained results.

Costa, Gianni et al., (2013) One way to organize XML documents is by identifying and then grouping those that share common structural components. So far, this has been achieved by inspecting the XML documents for the presence of a single type of predefined structural component. But it's possible that the structural components selected a priori aren't the best ones for efficient grouping. dis addition, the generated clusters probably include some internal structural inhomogeneity from unnoticed variances dis the XML documents' structures caused by other sorts of structural components that were missed. To get over these restrictions, we offer a novel hierarchical method for isolating structurally homogenous clusters of XML documents, which may take into account various types of structural components as needed. As a result, the XML document structures are further differentiated at each level of the resultant hierarchy by taking into account new structural components. A unique method is used to summarize each cluster in the hierarchy, allowing for a distinct and clear comprehension of its structural aspects.Results from tests conducted on both genuine and fake XML data show that the proposed method is more efficient and scalable than its well-established rivals. It has also been demonstrated that cluster summarization is highly representative.

Nayak, Richi. (2008) The XML documents Clustering with Level Similarity (XCLS) incremental clustering technique is introduced in this research. It clusters XML documents based on their structural similarity. For more effective processing, XML documents are represented using a level structure format. In order to determine how closely the new document matches up with preexisting clusters, a global criteria function is created. Because it doesn't calculate the pair-wise similarity between each document, a lot of processing power is spared. In order to study the compromises between precision and efficiency, XCLS is further adjusted to include the semantic meanings of XML elements. According to the results of the empirical study, when grouping structured data like XML, structural similarity is more important than semantic similarity. According to the results of the experiments, the XCLS approach efficiently and accurately groups documents with different structures into clusters.

Dalamagas, Theodore et al., (2006) Common areas of study include methods for handling and analyzing XML data. But there hasn't been much focus on procedures that take XML data structure into account. Among these processes is the grouping of XML documents with similar structures. Using clustering algorithms using distances to evaluate how similar two tree topologies are leads to this grouping. In this work, we offer a system for structurally grouping XML documents. Here, we model the XML documents as rooted ordered labeled trees and investigate how hierarchical clustering methods use structural distance metrics to find groups of XML documents that are structurally similar. If you want a more efficient distance computation that doesn't sacrifice quality, try using structural summaries for trees. We put our method through its paces on a prototype testbed.

III. EXPERIMENTAL SETUP

This similarity-based clustering technique was developed for use in an experimental setting that required a personal computer with a 2.66 GHz Intel Core 2 Duo processor. This CPU is powerful enough to conduct the clustering operations of the algorithm. The system's ability to manage several tasks at once is guaranteed by its dual-core CPU, albeit its speed could fall short when compared to contemporary computing standards. Data is mostly stored on a 250 GB SATA HDD, which is part of the system's storage. The XML data utilized in this experiment, including datasets, temporary files, and intermediate clustering findings, may be adequately stored on SATA HDDs, despite their inferior performance when compared to newer SSDs. The system also has 2 GB of DDR RAM, which isn't a lot by today's standards but was plenty for the massive datasets that were handled in the trials. The available RAM may be insufficient for larger datasets or operations that need a lot of memory.

Although it is a legacy platform, Windows XP offers a reliable environment for executing the software tools and algorithms needed for this setup, despite its age. The system makes use of SQL Server 2005, an RDBMS that allows for the storing and manipulation of structured data, including XML, for handling the XML data. Because it can handle XML queries, this program is great for clustering with big datasets. To store and manage XML data natively, we also use BDBXML, an open-source database engine that is specialized for XML data. In order to process the XML documents involved in clustering activities quickly, BDBXML offers rapid indexing and retrieval capabilities.

Because of its widespread support for data manipulation and algorithmic operations, as well as its cross-platform capabilities, Java was selected to develop the clustering algorithm. This allowed for the flexibility needed to handle the XML data and execute the similarity-based clustering process. Although it is built on outdated technology, this configuration provides a viable environment for testing with moderate-sized datasets and for evaluating the clustering algorithm's performance.

The table below compares the performance of SQL Server 2005 to its competitors. Across all element amounts, retrieval times are longest when clustering or indexing is not used. For 2.5k elements, it starts at 0.328 seconds and increases to 0.88 seconds for 10k elements. Even without indexing or clustering, using clustering alone speeds up retrieval times. Retrieving 5,000 items now takes only 0.48 seconds, down from 0.58 seconds before clustering. Similarly, retrieval time drops from 0.88 seconds to 0.82 seconds when there are 10,000 items. While indexing alone does increase retrieval speeds, it is not nearly as noticeable as clustering, particularly when dealing with larger data quantities. Compared to clustering, which works better with bigger datasets, the retrieval time for 7.5k items lowers to 0.705 seconds, from 0.771 seconds. The best performance improvements are seen when indexing and clustering are used together, since retrieval times are drastically reduced for all data amounts. With 2.5k elements, the time to retrieve is a mere 0.185 seconds, and with 10k elements, it drops to 0.46 seconds. The time needed to obtain XML data from the database using various ways is illustrated in Figure 1. In seconds, the Y-axis shows how long it takes to get XML data out of the database. Number of items obtained is represented on the X-axis.

Table 1: Comparing the performance of SQL server 2005

Strategy	**Number of elements retrieved**			
	2.5.k	**5.0 k**	**7.5k**	**10k**
Without cluster and index	0.328	0.58	0.771	0.88
Cluster	0.312	0.48	0.669	0.82
Index	0.318	0.55	0.705	0.82
Cluster with index	0.185	0.31	0.471	0.46

IV. RESULTS AND DISCUSSION

Performance analysis SQL server 2005

Thousands of XML items are used to evaluate the similarity based grouping approach. We begin by creating a database with the name empfull.mdb. The following fields are generated in the empc table: Sno (int) and Empinfo (xml) in this database. There are components for name, department, designation, and address in the XML column empinfo. The table has ten thousand XML elements that are well-formed. Data may be retrieved from the table using queries.

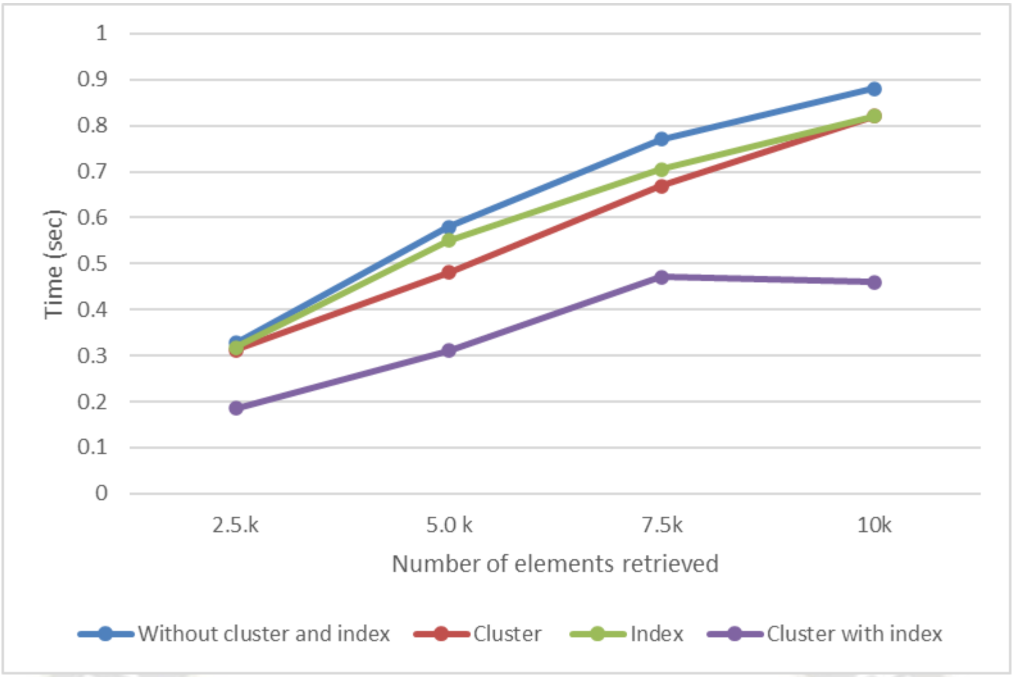

Figure 1: Comparing the performance of SQL server2005

Performance analysis BDBXML

The XML data in BDBXML is stored in containers. The first container is called staff .Created is dbxml. The following parameter is used to add 10,000 components to this container. Name, department, designation, and address are elements, and a serial number is an attribute. To construct a container, use the command create container. Then, to enter components into the container, use put Document. Regarding the container crew. The data may be retrieved using dbxml queries. Consider a data retrieval query that does not make advantage of indexing and clustering. This query might be paraphrased as follows: "from the container called staff Choose all employees with the job title "System Analyst" using dbxml.

Berkeley DBXML without Cluster and Index is compared in the table.

Strategy	Number of elements retrieved			
	2.5.k	5.0 k	7.5k	10k
Without cluster and index	2.312	2.528	2.638	3.448
Cluster	1.943	2.326	2.487	2.940
Index	0.627	1.107	2.126	2.721
Cluster with index	0.459	0.961	1.661	2.389

Table 2: Comparing the performance of Berkeley DBXML

The retrieval times begin at 2.312 seconds for 2.5k items and gradually rise to 3.448 seconds for 10k elements, with a little increase for each additional element. When compared to the "Without Cluster and Index" technique, the retrieval time is reduced when clustering is used alone. The recovery time is 1.943 seconds for 2.5k items and steadily rises to 2.940 seconds for 10k elements, with just a little increase as the number of components increases. With the index technique, retrieval times are improved even more noticeably. With a starting time of 0.627 seconds for 2.5k items, the retrieval time gradually climbs to 2.721 seconds for 10k elements, which is still lower than the other techniques. On every dataset, clustering and indexing work together to get optimal results. Beginning at 0.459 seconds for 2.5k elements, the retrieval time steadily rises to 2.389 seconds for 10k elements.

See how long it takes to get XML data out of the container using various approaches in Fig. 2. In seconds, the Y-axis shows how long it takes to get XML data from the whole document container. Number of items obtained is represented on the X-axis. Without clustering and indexing, it takes 2.313 seconds to get 2500 objects of a certain kind from the container. It takes 0.625 seconds to retrieve the data after indexing. Retrieving the identical data using clustering and indexing takes 0.462 seconds. Clustering with an indexing technique results in superior performance, as shown in this graph.

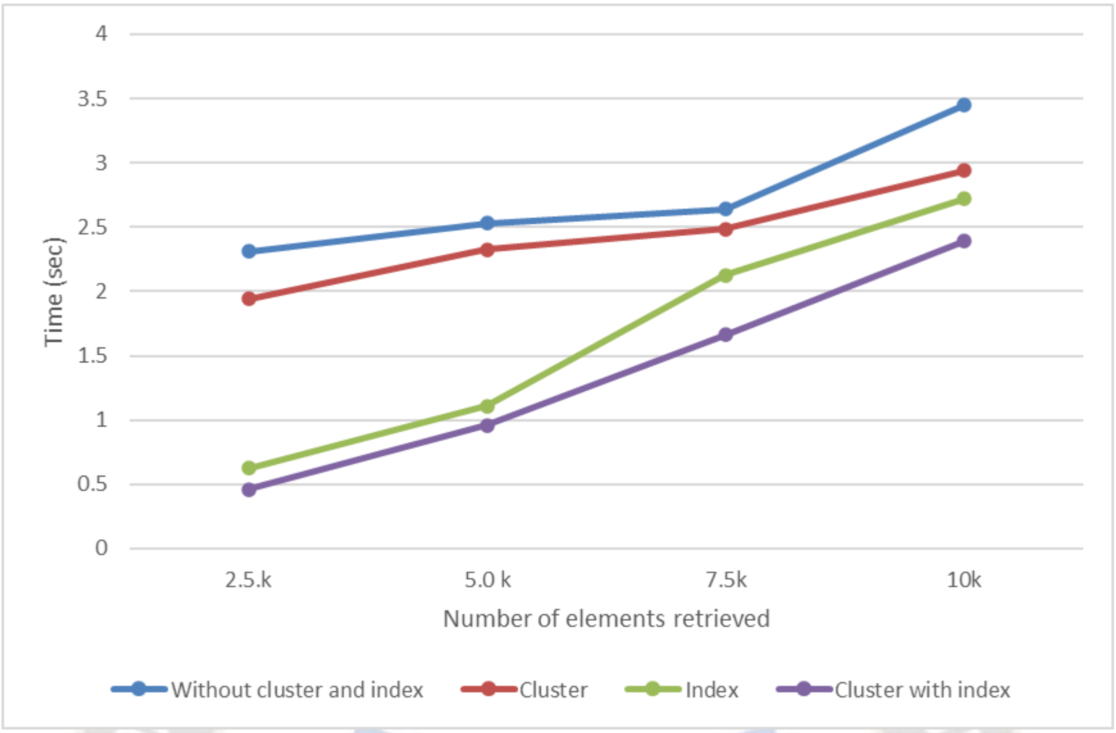

Figure 2: Comparing the performance of Berkely DBXML

V. CONCLUSION

We look at how well SQLServer 2005 and Berkeley DBXML handle XML documents using a similarity-based clustering approach with indexing. The findings show that indexing is not enough to get the desired performance boost. Combining clustering with indexing yields better results than either technique alone. We intend to enhance the efficiency of data retrieval in the future by modifying the current XML indexing algorithms. It is also suggested to use Oracle database management systems to evaluate the clustering strategy.

REFERENCES: -

[1] S.-C. Haw, A. Amin, P. Naveen, and K.-W. Ng, "Performance Evaluation of XML Dynamic Labeling Schemes on Relational Database," *Int. J. Technol.*, vol. 13, no. 5, p. 1055, 2022, doi: 10.14716/ijtech.v13i5.5871.

[2] A. Klaib, "XML Dataset and Benchmarks for Performance Testing of the CLS Labelling Scheme," *Sebha Univ. J. Pure Appl. Sci.*, vol. 20, no. 2, pp. 12-15, 2021, doi: 10.51984/JOPAS.V20I2.1243.

[3] L. Cai, "XML and Its Application in Library and Information Retrieval," *J. Phys. Conf. Ser.*, vol. 1544, no. 1, p. 012091, 2020, doi: 10.1088/1742-6596/1544/1/012091.

[4] Saranya and B. S. E. Zoraida, "A Study on Clustering Algorithms for XML Data Clustering," *IOSR J. Comput. Eng.*, vol. 18, no. 5, pp. 84-89, 2016, doi: 10.9790/0661-1805018489.

[5] R. Desoki and A. Elfatatry, "Towards improving XML search by using structure clustering technique," *J. Inf. Sci.*, vol. 41, no. 2, pp. 146-166, 2014, doi: 10.1177/0165551514560523.

[6] N. Alghamdi, W. Rahayu, and E. Pardede, "Semantic-based Structural and Content Indexing for the Efficient Retrieval of Queries over Large XML Data Repositories," *Future Gener. Comput. Syst.*, vol. 37, no. 2, pp. 1-12, 2014, doi: 10.1016/j.future.2014.02.010.

[7] G. Costa, G. Manco, R. Ortale, and E. Ritacco, "Hierarchical Clustering of XML Documents Focused on Structural Components," *Data Knowl. Eng.*, vol. 84, no. 1, pp. 26–46, 2013, doi: 10.1016/j.datak.2012.12.002.

[8] L. Xu, T. W. Ling, and H. Wu, "Labeling Dynamic XML Documents: An Order-centric Approach," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 1, pp. 100-113, 2012.

[9] R. Nayak, "Fast and Effective Clustering of XML Data Using Structural Information," *Knowl. Inf. Syst.*, vol. 14, no. 2, pp. 197-215, 2008, doi: 10.1007/s10115-007-0080-8.

[10] T. Dalamagas, T. Cheng, K.-J. Winkel, and T. Sellis, "A Methodology for Clustering XML Documents by Structure," *Inf. Syst.*, vol. 31, pp. 187-228, 2006, doi: 10.1016/j.is.2004.11.009.

[11] J.-M. Bremer and M. Gertz, "Integrating Document and Data Retrieval Based on XML," *VLDB J.*, vol. 15,

no. 1, pp. 1-5, 2005, doi: 10.1007/s00778-005-0150-z.

[12] B. Catania, A. Maddalena, and A. Vakali, "XML Document Indexes: A Classification," *IEEE Internet Comput.*, vol. 9, no. 5, pp. 64-71, 2005, doi: 10.1109/MIC.2005.115.

[13] H.-J. Lee, B.-S. Jeong, and D.-H. Kim, "Storage and Retrieval of XML Documents Without Redundant Path Information," *Kips Trans. Part D*, vol. 12D, no. 5, pp. 663-672, 2005, doi: 10.3745/KIPSTD.2005.12D.5.663.

[14] R. Luk, H. Leong, T. Dillon, A. Chan, W. Croft, and J. Allan, "A Survey in Indexing and Searching XML Documents," *J. Am. Soc. Inf. Sci. Technol.*, vol. 53, no. 6, pp. 415-437, 2002, doi: 10.1002/asi.10056.

[15] J. Hoven, "Database Management System and XML: Interchange of Data," *IS Manag.*, vol. 19, no. 1, pp. 94-96, 2002, doi: 10.1201/1078/43199.19.1.20020101/31482.13.