A Technique for Optimization of Semantic Query in XML Databases

Idayana Alabere, & E. O. Bennett
Department of Computer Science,
Rivers State University,
Port Harcourt,
Nigeria.
alabereidayana@gmail.com, bennett.okoni@ust.edu.ng

Abstract
The increasing volume of XML data reduces the degree of efficiency in the retrieval of data from XML databases. It is therefore necessary to devise efficient methodologies and tools to transform, store and retrieve data making them as efficient as possible to minimize the time the users wait for response or the time application programs are delayed. The direct indexing and multilevel indexing techniques were produced using the constructive research methodology and object-oriented design methodology. Firstly, the XML nodes at each level were classified according to their data types. Secondly, from the index of each classification, new nodes were generated and the indexes were stored as an array, enabling access to the content of a node through the array. Java programming language was used for the implementation of the system. Experiments were conducted to evaluate the performance of the direct and multilevel indexing techniques in XML query optimization before and after applying the techniques. The results gotten after comparing the performance of the direct and multilevel indexing techniques and the performance where these techniques are not applicable in an XML database system shows the efficiency of these techniques in the retrieval of data in XML database system.

Key words: *Optimization, Query, Semantic, XML*

1. Introduction

The extensible Markup Language (XML) is a recommended standard of the World Wide Web Consortium (W3C) that is increasingly being used for various forms of data exchange between differing data sources (Kurita, et.al, 2007).

XML has gained a lot of attention as a result its powerful capability of storing and transporting data and the fact that it doesn't require a defined schema for storing information in a database. Compared to DTD, XML produces more semantic information which is useful for XML document processing, primarily in formulation, optimization and evaluation of XML queries. Unfortunately, existing query processing systems and XML query languages use a little of this valued semantic information. Conventional techniques for optimizing query try to ascertain the most efficient sequence of operation for physical database system to function based on the syntax of a specific query and access techniques available (Lin, 1999). Query optimization is a method of reviewing a query to generate an execution strategy for the known query from the space of likely execution strategies. It is simply a way of finding the best execution strategy for a query given amongst possible queries. The aim of the work is to produce a mechanism for optimization of semantic query for the efficient retrieval of data in XML database system.

In this paper, a system that uses the direct and multilevel indexing techniques is developed for the efficient retrieval of data from XML databases.

2. Review of Related Works

Several query optimization techniques have been developed for optimizing queries in XML databases using its semantic constraints, among them are; Extended Dewey Labeling Scheme, Semantic Query Transformation (SQT) and Prefix on Demand (PoD), Map Reduce-based algorithm, Twig Table algorithm and Stack-based algorithm.

The Extended Dewey Labeling Scheme (Lu, et.al, 2010) was developed to improve the efficiency of XML tree pattern matching, which broadens the existing Dewey labeling scheme to link the types and identifiers of elements in a label, and to prevent the scan of labels for internal query nodes from increasing query processing in (I/O cost).
The Semantic Query Transformation (SQT) and Prefix on Demand (PoD) were developed to optimize XML query processing by making the most of the semantic constraints marked out in the XML schemas to optimize structural join and Twig queries (Le, et.al, 2013).
Map Reduce-based algorithm, a dynamic algorithm that alters the label task procedure by focusing on a part of the XML data at the time of the labeling process, thereby enabling the decrease in the size of the label was developed for repetitive prime number labeling of XML data (Ahn, et.al, 2016).

The Twig Table algorithm for twig pattern query processing was developed by to deal with the issues resulting from the absence of semantics on object, attribute and principles in approaches that are already in existence, semantics-based relational tables integrated disordered list of tags to help in the processing of the twig pattern query (Wu, et.al, 2011).
Stack-based algorithm was designed to adequately respond to queries with the use of materialized views that closely encodes in polynomial time and space all the homomorphism (transforms a set into another that preserves in the second set the relations between elements of the first) from a view to a query (Wu, et.al, 2009). Space and time optimization was later developed by them with the use of bitmaps for encoding view materializations and applying bitmap operations in order to reduce the cost of evaluating the queries.

3. Methodology and Design

The methodologies used for this work are constructive research methodology and object-oriented design methodology. The constructive research methodology was adopted because the processes involved in constructive research methodology (Lehtiranta, et.al, 2017); Selecting a practically relevant problem, obtaining a comprehensive understanding of the study area, designing one or more applicable solutions to the problem, demonstrating the solution's feasibility, linking the results back to the theory and demonstrating their practical contribution and examining the general ability of the results were most appropriate for the research. The object-oriented design methodology (White, 1994) was used because the steps involved; Requirements analysis, Domain analysis, System design and Implementation were best suitable in achieving the aim of this work. Fig 1 shows the detailed architecture of the system.

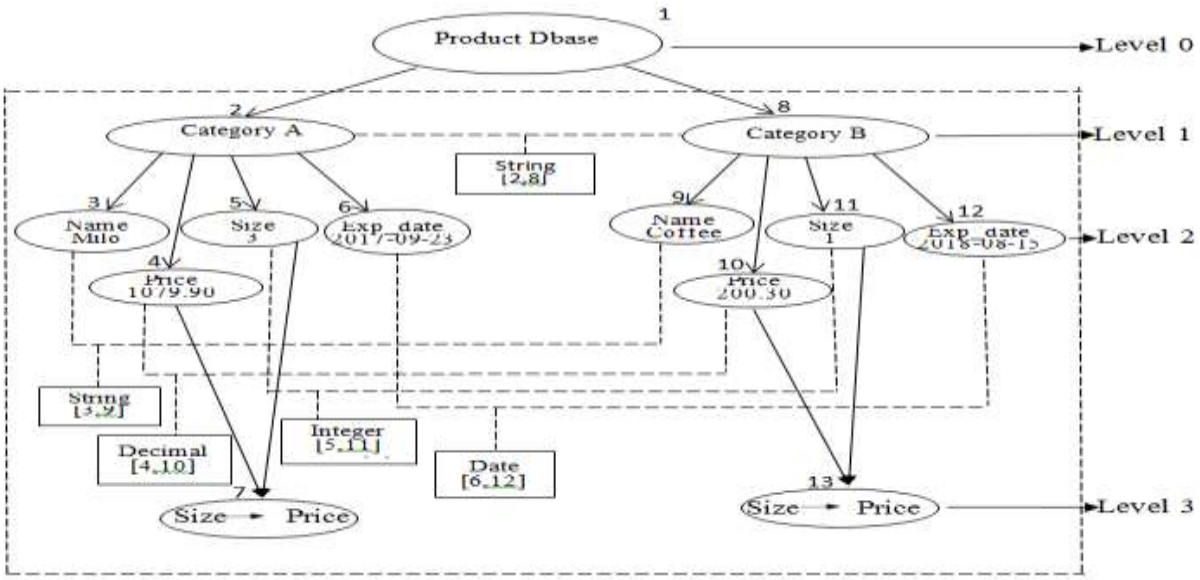

Figure 1: Detailed System Architecture

In Figure 1, there are 4 levels of classification. The oval shapes represent the nodes, and each node has a unique index at the top of the node beginning from 1-13. The dotted line link the nodes, classifies them according to their data types and stores them as an array on each level of the architecture as shown in the rectangles. In level 1, Category A with index 2 and Category B with index 8 are classified as string with an array S[2,8]. In level 2, the node (Name, Milo) with index 3 and the node (Name, Coffee) are classified as Strings (S) with an array S[3,9], the node (Price, 1079.90) with index 4 and the node (Price, 200.30) with index 10 are classified as Decimal (DL) with an array DL[4,10], the node (Size, 3) with index 5 and the node (Size, 1) with index 11 are classified as integer (I) with an array I[5,11], the node (Exp_date, 2017-09-23) with index 6 and the node (Exp_date, 2018-08-15) with index 12 are classified as date (D) with an array D[6,12]. In level four, the nodes (Size Price) with index 7 and 13 simply means that the Price is functionally dependent on the Size which means that the Price of a product is determined by the size of the product.

The system does two major things

- It classifies the nodes at the same level of the architecture according to their data types.
- From the index of each classification, new nodes are generated and stored as an array, thereby enabling access to the content of a node through the array.

4. Implementation and Results

The direct indexing and the multilevel indexing techniques described in this paper are fully implemented into an XML database. The implementation was done using Java 7. Experiments were run on an Intel® Pentium® dual CPU T3200 @ 2.00GHz machine with 4.00GB memory based on three aspects; retrieval of all the items in each category, retrieval of a particular item from each category and retrieval of different sizes of an item.

Table 1: Average time of all the categories in each Indexing Pattern

INDEXING PATTERN	AVERAGE ACCESS TIME (SECS)
Without Indexing	0.0116
Direct Indexing	0.0062
Multilevel Indexing	0.0062

Table 1 shows the average access time of the total listing of all the categories without indexing, using direct indexing and multilevel indexing. The average time taken to access all the categories without indexing was 0.0116secs. In the direct indexing and multilevel indexing, the average time taken to access all the categories were 0.0062secs and 0.0062secs respectively.

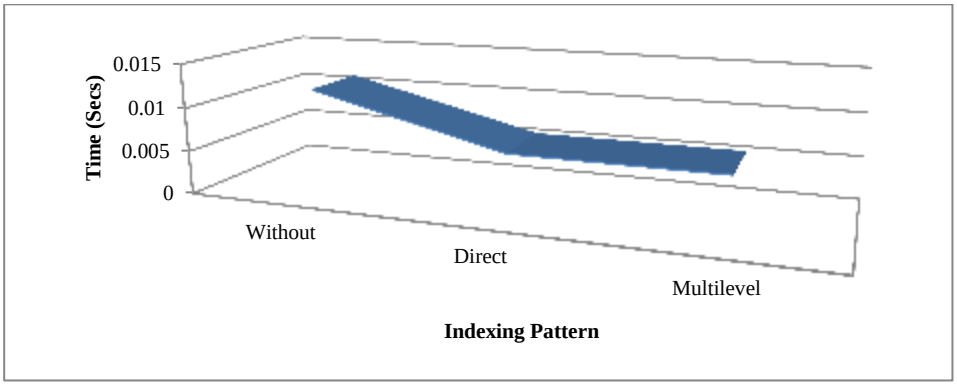

Figure 2: Graph showing the average time taken to access all the categories in each Indexing Pattern

Figure 2 shows the graphical representation of the time taken in access all the categories in each indexing pattern. For without indexing, the average access time was 0.0116secs, the line dropped drastically in the direct and multilevel indexing to 0.0062secs.

Table 2: Average time taken to access items from different categories using the Indexing Patterns

INDEXING PATTERNS	AVERAGE ACCESS TIME(SECS) CATEGORY A	AVERAGE ACCESS TIME(SECS) CATEGORY B	AVERAGE ACCESS TIME(SECS) CATEGORY C	AVERAGE ACCESS TIME(SECS) CATEGORY D	AVERAGE ACCESS TIME(SECS) CATEGORY E
Without Indexing	0.0038	0.0034	0.0036	0.0036	0.0034
Direct Indexing	0.0013	0.0016	0.0014	0.0015	0.0011
Multilevel Indexing	0.0018	0.0020	0.0018	0.0020	0.0017

Table 2 shows the average time taken to access items from Categories A, B, C, D and E, without indexing, using direct indexing and using multilevel indexing. Without indexing, the average access time for categories A.

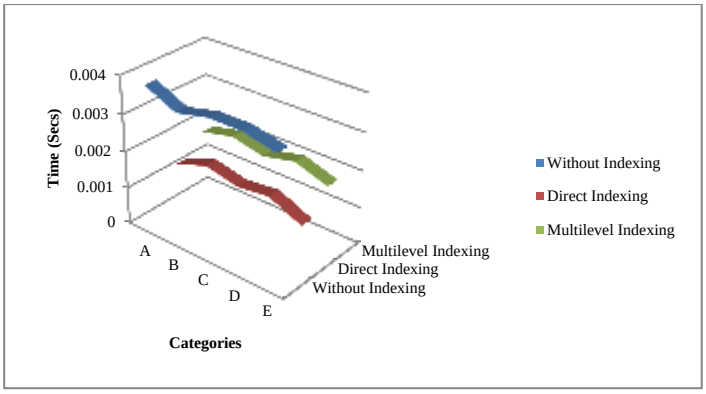

Figure 3: Graph showing the average access time of all items from different categories using the Indexing Patterns

Figure 3 shows the graphical representation of the average access time of all the items from different categories using the different indexing patterns. For Category A, the average access time without indexing was 0.0038secs, for direct indexing, the average access time was 0.0013secs and 0.0018secs for multilevel indexing. The average access time for category B were 0.0034secs without indexing, 0.0016secs using direct indexing and 0.0020secs using multilevel indexing of all the items in category B. Category C has the average time without indexing as 0.0036secs, 0.0014secs for direct indexing and 0.0018secs using multilevel indexing to access all the items in category C. For Category D, the average access time without indexing was 0.0036secs, for direct indexing, the average access time was 0.0015secs and 0.0020secs for multilevel indexing. The average access time for category E were 0.0034secs without indexing, 0.0011secs using direct indexing and 0.0017secs using multilevel indexing of all the items in category E.

Table 3: Average time taken to access different sizes of an item using indexing patterns

CATEGORIES	INDEXING PATTERNS	AVERAGE ACCESS TIME(SECS) SIZE 1	AVERAGE ACCESS TIME(SECS) SIZE 2	AVERAGE ACCESS TIME(SECS) SIZE 3
Category A	Without Indexing	0.0247	0.0250	0.0247
	Direct Indexing	0.0012	0.0012	0.0013
	Multilevel Indexing	0.0018	0.0017	0.0018
Category B	Without Indexing	0.0163	0.0160	N/A
	Direct Indexing	0.0012	0.0012	N/A
	Multilevel Indexing	0.0021	0.0021	N/A
Category C	Without Indexing	0.0183	0.0177	N/A
	Direct Indexing	0.0013	0.0012	N/A
	Multilevel Indexing	0.0020	0.0021	N/A

Table 3 shows the average access time taken to access different sizes of an item without indexing, using direct indexing and multilevel indexing patterns category A, B and C.

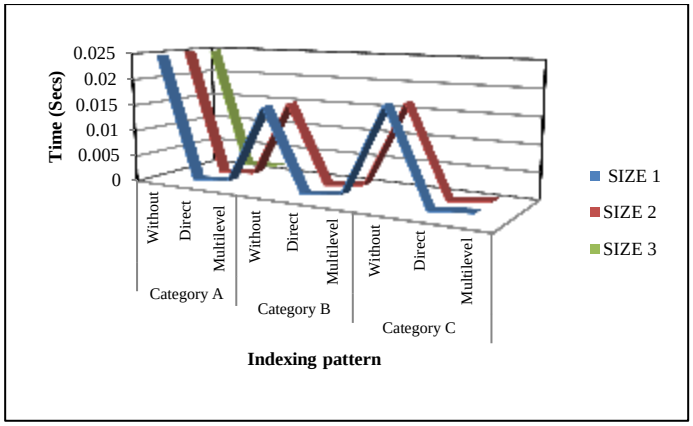

Figure 4: Graph showing average time taken to access different sizes of an item using indexing patterns

Figure 4 has 3 lines representing the size 1, 2 and 3. Category A has 3 sizes and categories B and C has 2 sizes. From figure 4.12 in category A, the average time taken to access size 1 without indexing, using direct indexing an multilevel indexing were 0.0247secs, 0.0012secs and 0.0018ssecs respectively. The average access time for size 2 (Category A) without using indexing, direct indexing and multilevel indexing were 0.0250secs, 0.0012secs and 0.0017secs and size 3 average access time were 0.0247secs, 0.0013secs and 0.0018secs respectively. Category B and C has 2 sizes, 1 and 2. In category B, the average access time for size 1 without indexing, using direct indexing and multilevel indexing were 0.0163secs, 0.0012secs and 0.0021secs and size 2 average access time without indexing, using direct indexing and multilevel indexing were 0.0160secs, 0.0012secs and 0.0021secs respectively. In category C, the average access time for size 1 without indexing, using direct indexing and multilevel indexing were 0.0183secs, 0.0013secs and 0.0020secs and size 2 average access time without indexing, using direct indexing and multilevel indexing were 0.0177secs, 0.0012secs and 0.0021secs respectively.

5. Discussion

The direct indexing and multilevel indexing techniques were produced for optimizing semantic query to enable the efficient retrieval of data in XML database systems to reduce the time overhead in such a way that the memory allocation will be reduced. The direct indexing technique enables the user to access data from the database using its index id, and the multilevel indexing techniques enables the user to access data from the database using the index path. Experiments were conducted to evaluate the performance of the direct indexing and the multilevel techniques before and after applying the techniques. The results show that the direct indexing and multilevel indexing techniques are both efficient for the retrieval of data from XML databases with respect to time and memory allocation. However, there is a substantial increase in the run time of the multilevel indexing technique over the direct indexing technique when items are accessed from different categories and when different sizes of an item is accessed as a result of its index path.

6. Future Works

This paper has produced the direct and multilevel indexing techniques for the efficient retrieval of data from XML databases. Despite the fact that these techniques were able to reduce the overhead time and memory allocation, some areas are subject to further work.

i. Modifying the multilevel indexing technique to make the memory allocation as reduced as the direct indexing technique

ii. Devising ways to modify the multilevel indexing technique to make the access time as efficient as the direct indexing technique

References

Ahn, J., HyukIm, D., Lee, T. & Kim, H. (2016). A dynamic and parallel approach for repetitive prime labeling of XML with MapReduce. The Journal of Supercomputing. DOI 10.1007/s11227-016-1803-y.

Kurita, H., Hatano, K., Miyazaki, J. & Uemura, S. (2007). Efficiency Query Processing for Large XML Data in Distributed Environments. IEEE International Conference on Advanced Networking and Application (AINA'07).

Le, D., Maghaydah, M., Orgun, M. & Zhong, Y. (2013). Optimization of XML Queries by Using Semantics in XML Schemas and the Document Structure. WISE , Part I, LNCS 8180, pp. 343–353.

Lehtiranta, L., Junnonen, J., Karna, S. & Pekuri, L. (2017). Designs, Methods and Practices for Research of Project Management. ISBN 978-1-4094-4880-8.

Lin, L. (1999). Design and Implementation of Semantic Query Technique Join elimination in IBM DB2. Cornputer Science depsrtment, York University North York, Ontario Dec.

Lu, J., Meng, X. & Ling, T. (2010). Indexing and querying XML using extended Dewey labeling scheme. Data Engineering and Knowledge Engineering, MOE, Renmin University of China, China doi:10.1016/j.datak.

White, I. (1994). Booch Method: A Rational Approach. Benjamin-Cummings Pub Co (March 1).

Wu, H., Ling, T., Chen, B. & Xu, L. (2011). TwigTable: using semantics in XML twig pattern query processing. School of Computing, National University of Singapore.

Wu, X., Theodoratos, D. & Wang, W. (2009). Answering XML Queries Using Materialized Views Revisited. New Jersey Institute of Technology, USA. DOI: 10.1145/1645953.1646015 · Source: DBLP