Statistics for Cost-Based XML Query Optimization

Jose de Aguiar Moraes Filho, Theo Härder

University of Kaiserslautern
{ aguiar | haerder}@informatik.uni-kl.de

Abstract. Cost-based query optimization (CBO) is a very important area for data management in general and for XML data management in particular. For native XML database management systems (XDBMS), CBO techniques are harder than for relational databases, because the underlying tree-based data model is much more complex and the relative order (document order) between XML elements (nodes) matters. In this paper, we present our first ideas on statistics data structures supporting CBO, that is, the mapping of a logical access model with algebraic operators to the physical access model. In our prototype system called XTC (XML Transaction Coordinator), the physical access model is embodied by a toolbox of path processing operators from which the best performing operators have to be selected based on statistic information for the construction of the query execution plan.

1 Introduction

The increasing use of XML urges DB research to improve query processing on natively stored XML data. Regarding current XML research, several approaches are focusing on heuristic or algebraic query processing techniques. However, only a few of them are taking into account the so-called cost-based XML query optimization (XML CBO). To the best of our knowledge, there are hardly any published research projects coping specifically with this subject. CBO is a very important area for data management in general and for XML data management in particular. With CBO, a query processor can choose a better query execution plan (QEP) based on information (the so-called statistics) about the data to be queried. Hence, query execution may be improved with respect to response time and/or transaction throughput. On the other hand, we have learned from relational database use that it is not always easy to derive appropriate data statistics and to keep them accurate. For native XDBMSs, this task is even harder because the underlying tree-based data model is much more complex and the relative order (document order) between XML elements (nodes) matters.

This paper is structured as follows. We briefly present in Section 2 the XTC system serving as our runtime environment for all optimization considerations. Then, Section 3 studies XML Query Processing. In section 4, we sketch our proposal to store statistical data in a XDBMS. Related works on XML CBO are referenced in Section 5, before we wrap up the paper.

2 The XTC XDBMS

XTC is a full-fledged native XML database manager already proven as an engine effective for storing and controlling concurrent access to XML documents in a multi-user environment [5, 10]. It allows the use of several XML APIs, such as DOM and SAX, and querying XML data through descriptive languages such as XPATH [14] and XQuery [15]. The XTC design strictly adheres to the well-known layered database architecture [3]. In the following, we briefly sketch the most important XTC components for our work (see Fig. 1). Above a conventional solution for File Services and Propagation Control, the Access Service layer maintains all physical object representations, that is, data records, fields, etc. as well as access paths structures, such as lists, B- and B*-trees, as

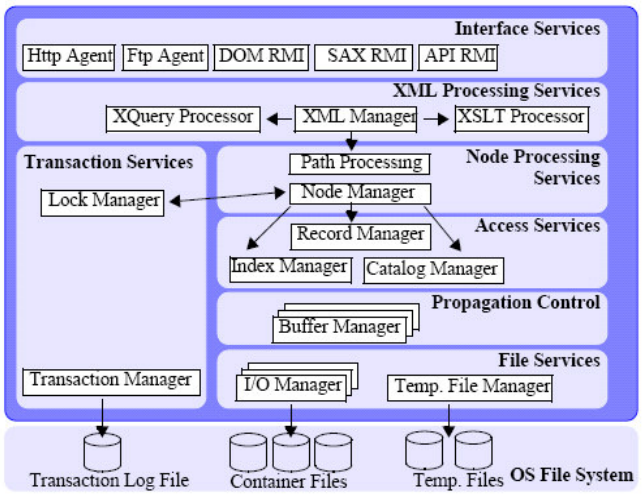

Figure 1. XTC Architecture

well as internal catalog information. The Node Processing Services (NPS) layer provides navigational and set-oriented physical operations (such as structural joins) to be used in the physical access model. The XML Services layer directly supports the various XML language models offered to the applications and internally provides the logical access model together with its mapping to the layer below.

In XTC, an XML document is stored into a B*-tree in which prefix-based labelling, the SPLID scheme (*stable path labeling identifier*) in XTC terminology, is used. The resulting structure is called *document store* (Fig.2a). Thus, XTC controls the document order and enables fast direct and navigational accesses (parent/child/sibling) to XML data. To encode the element names in an XML document, XTC maintains a vocabulary which is represented by an ordered list of (VocID, name) pairs where VocID contains the internal numeric representation of an element name or attribute name. Using B-tree indexes, an element index is additionally created and maintained by XTC. The *element index* keeps a Name Directory with (potentially) all element names – so-called tags – occurring in an XML document. Each tag entry in the Name Directory points to a set of SPLIDs (or DeweyIDs [4, 6]) in document order, which address all corresponding element nodes in the document store (Fig. 2b). In fact, all SPLIDs indexing the nodes of a single element name are organized as a B*-tree which maintains the document order among the SPLIDs and enables direct access to each of them. With these storage mechanisms, XTC provides all "external" input sets for querying data in document order.

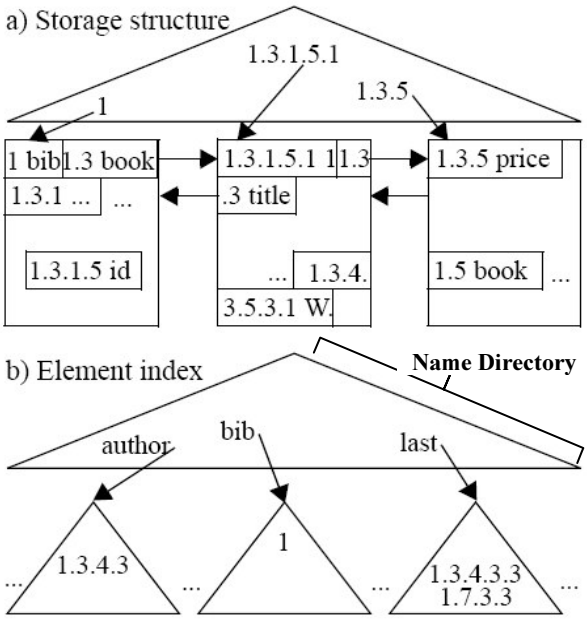

Figure 2. XTC Storage Model

3 XML Query Processing

Declarative query languages, such as XPath and XQuery, are supported by XTC in its XML Services layer. Because this kind of query processing is a complex activity, it requires appropriate levels of abstraction. The top-most level corresponds to the language model, where syntactic and semantic analyses have to be applied to the query statement processed. Furthermore, it is necessary to build an internal graph representation (logical algebra) of each query statement to enable graph transformations towards an equivalent, but more efficient processing structure and to allow for a kind of procedural evaluation.

Once the internal graph representation is built, optimization by rewriting the query can start at the level of the logical access model. Rewriting is primarily accomplished by use of algebraic transformations. Another possible optimization is applying sequence heuristics by which selective algebraic operations can be executed as early as possible. So far, no system-specific issue was taken into account.

For the next level, the physical access model, our NPS layer provides a set of path processing operators performing a variety of structural joins. Hence, each logical algebraic operator is mapped to one or more physical operators, yielding several possible query QEPs. Each physical operator embodies a specific algorithm that relies on the existence of indexes, document structure, element order, etc. In order to choose the optimal QEP, it is necessary to enumerate the set of "promising" plans and, for each of them, to estimate the cost by applying a tailored cost model. The cost model in turn relies on statistical information of XML data. For example, statistics might reflect the cardinality of XML elements, the number of physical pages in which the XML document is stored and the selectivity of paths in the document. The cost estimate takes I/O costs, CPU costs, and communication costs (relevant to distributed settings) into account. In fact, the costs of I/O and CPU usage heavily depend on the underlying storage model. After having computed a set of candidate QEPs, the cheapest QEP is chosen and evaluated.

In general, the use of statistics implies a trade-off between statistics accuracy and storage space / maintenance effort needed. The more accurate the statistics should be, the more space is necessary to store it and the more frequent it has to be adjusted to the existing data structures.

4 Statistics for Cost-Based XML Query Processing

```
<bib>
 <book year="1994">
  <title>TCP/IP Illustrated</title>
  <author>
   <last>Stevens</last>
   <first>W.</first>
  </author>
  <publisher>Addison-Wesley</publisher>
  <price> 65.95</price>
 </book>
 <book year="1992">
  <title>Advanced Programming in the…</title>
  <author>
   <last>Stevens</last>
   <first>W.</first>
  </author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
 </book>
 <book year="2000">
  …
 </book>
……
</bib>
```

Figure 3. Sample XML Document

To illustrate these concepts and to show the use of statistical information, consider an XML document stored in XTC a sample fragment of which is depicted in Figure 3. We can use XPath or XQuery statements to query this document. In Figure 4, we illustrate a query statement in both languages that requests authors of books having the value (name) "Ende" and whose books have the title "Momo". Here, two characteristics of XML query processing are revealed: Queries referring to structure represented by path steps and those referring to values represented by where-clause predicates of an XQuery statement or brackets of an XPath expression. Whatever language in which the query is submitted, the query statement has to be internally translated into logical algebra expressions [8, 9]. In Figure 4, the query graph consists of a set of join operations (⋈) and indicates a kind of logical precedence structure. Based on this logical algebraic expression, the optimizer should generate physical plans by combining existing physical operators.

Queries

XPath) //book[/title="Momo"]//author["Ende"]

XQuery) For $i in //book, $a in $i//author
Where $i/title="Momo" and $a="Ende"
Return ($i, $a)

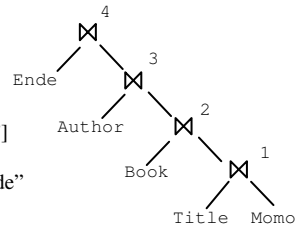

Figure 4. A Query and its Logical Algebraic Plan

For optimization purposes, the NPS layer provides an extensive set of structural joins and other path processing operators in a so-called PPO (*path processing operators*) toolbox. Because there are at least three orthogonal dimensions for building and classifying these operators, for each logical operator we can make several physical operators available whose optimal use is dependent on the specific characteristics of the XML document. The *axis operator* dimension has 8 entries and regards the various XPath axes (e.g. ancestor, descendant, etc.). The *evaluation sequence* dimension considers the PPO use for the optimal selectivity-dependent processing sequence of path expressions: bottom-up (leaf-to-root), top-down (root-to-leaf), or starting in the middle. Note, although a logical precedence structure for the path processing operators is given by the query formulation, query evaluation may deviate from it, as long as the same final result can be guaranteed. Finally, we have the dimension *evaluation type* where two kinds of algorithms are possible to evaluate a path expression: binary structural join and holistic twig join. The former structurally joins each pair of axis operators; whereas the latter evaluates the whole (or larger parts of the) path expression. Each of these approaches can in turn operate in navigational mode (i.e., node at a time) or in set-oriented mode (i.e., sequence at a time). For example, [9] proposes eight structural hash join algorithms in the axes parent, ancestor, child and descendant, whether in bottom-up or top-down fashion, namely ParHashA, AncHashB, ParHashB, AncHashB, ChildHashA, DescHashB, ChildHashB, and DescHashB. The endings A and B define which input sequence is hashed.

In order to drive the logical-to-physical mapping, we also need a data structure to store statistics. Due to the dual characteristics (structure and value) of XML documents and to make a complete coverage, we want to use two kinds of statistics: structural statistics and value statistics. The former is tailored to the evaluation of path expressions, whereas the latter is useful to evaluate predicates. Whatever the kind of statistics, the storage structure should combine the benefits of both: capturing the concept of document order and being compact enough to be stored in memory.

In XTC, we can initially figure out a data structure to store statistics in which we capture distinct XML elements in a structure similar to the Name Directory. Each element points to a set of pairs (*path, count*) for each ancestor and descendant axis. *Path* contains the encoded distinct path rooted by the element, whether ancestor(PathA) or descendant (PathD). *Count*, also called frequency, represents the number of ancestor (descendant) paths. We call it AXS-Stat (An XML Structural Statistics, see Figure 5). This approach has the potential to consume much memory space, since some redundancy is allowed. However, this problem can be overcome by using adaptive techniques. For example, given a memory budget, paths with lowest frequencies could be deleted and PathD-tables with no path should not be stored. If the budget changes more information can be stored in the AXS-Stat. In the worst case, there would be only one encoded PathD-table rooted by document root.

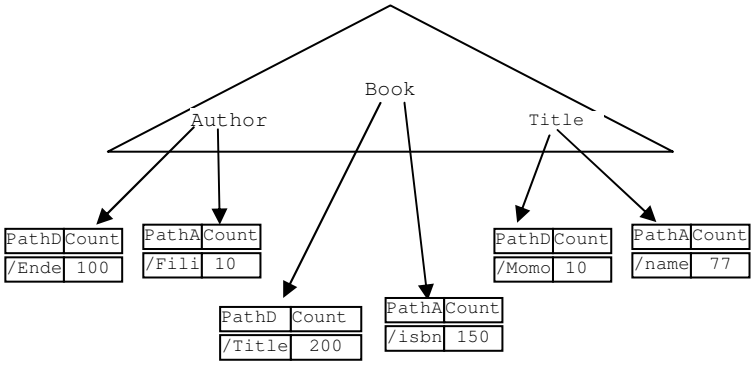

Figure 5. XML Statistical Data Structure

Continuing the query optimization, each join operation in Fig. 4 can be mapped to one or more algorithms yielding several possible query plans (QEP). Based on a cost model and statistics, a cost is computed for each QEP. For illustration, let us pick up the last join operation of the plan (number 1 in Fig. 4). This operation can be performed by a Nested Loop join algorithm [2], a ParHashA algorithm or a ParHashB algorithm. The statistics structure indicates that the cardinality of title is 200 and that of Momo is 10. Thus, Momo´s selectivity is 0.05 under title. For the first algorithm, the cost would be 2000 in the worst case. For the second one (ChildHashA), the cost would be 1000, in case it builds a hash table, because of the lesser selectivity, on title being the parent of Momo, and then probes with the SPLIDs of the Momo values. The third one, ParHashB hashes child (Momo), having much higher selectivity than title, and probes title. Let us assume its cost is 500. Thus, the algorithm of choice to process the join is ParHashB. This optimization process is recursively executed for all operations in the plan. The final optimized QEP would look like Figure 6. Here, the ChildHashA algorithm was selected for join 3, again based on statistics, because the book element is more selective than the author element and this algorithm builds a hash table for parent/ancestor (book).

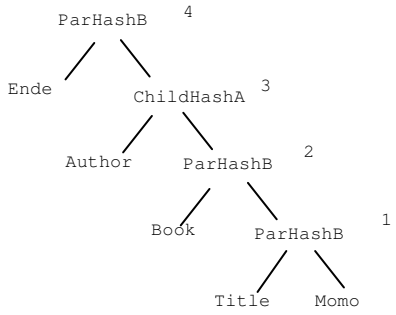

Figure 6. The Optimized QEP

5 Related Work

So far, there are only a few publications coping specifically with XML CBO: Data Guide in [11]; Path Tree (PT) and Markov Tables (MT) [1]; XPathLearner in [7]; XSKETCH in [12] and Bloom Histogram in [13]. All of these proposals summarize XML data by the use of a tree-based structure and once the structure cannot be stored in memory, they prune paths with length up to K, where K is a tuning parameter. All of them propose pruning at level 2 (K=2). Some of them, such as Markov Tables and XPathLearner, use a Markov model to estimate longer paths. Even though, they use pruned paths in table entries. Others use histograms, as in XSKETCH and Bloom Histograms, to model selectivity of paths and element values. Besides, only Data Guides have focused their use on a semi-structured DBMS. Empirical results for these proposals, especially on their quality to guide the query optimization phase, on their memory consumption, as well as on the maintenance overhead to be taken into account, are hardly available.

6 Conclusion

In this paper, we studied important aspects of XML cost-based query processing. We proposed, as our first idea, a specific structure to store statistical information for XML documents to be integrated into our prototype system XTC. This structure called AXS-Stat tries to capture the structural relationships among XML elements within a memory space budget. Moreover, AXS-Stat can be used in an adaptive fashion. We outlined the use of this statistical information in the course of the query optimization process.

Our future work includes the development and implementation of an algorithm for building and maintaining AXS-Stat. Furthermore, we will design and explore appropriate cost models for a cost-based query mechanism. XTC can be considered as an ideal testbed for comparative studies, because we can easily implement alternative approaches and run them under identical conditions. Therefore, by running benchmarks and comparative experiments against competing approaches (see Section 5), we can stepwise improve our approach based on empirical evidence and quantify to what extent it is competitive. Most important for this proceeding is the definition of suitable baseline experiments at the beginning to observe progress in the optimization endeavor at all.

References

1. Aboulnaga, A., Alameldeen, A. R., Naughton, J. F.: Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. In Proceedings of the 27th VLDB Conference. Roma, Italy, 2001, pp.591-600.
2. Al-Khalifa, S., Jagadish, H. V., Patel, J. M., Wu, Y., Koudas, N., Srivastava, D.: Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In Proceedings of the 18th Int. Conf. on Data Engineering. San Jose, 2002, pp. 141-152.
3. Härder, T., Reuter, A.: Concepts for Implementing a Centralized Database Management System. In Proceedings of the Int. Comp. Symposium on Application Systems Development, 1983, Nürnberg, pp. 28-60.
4. Härder, T., Haustein, M. P., Mathis, C., Wagner, M.: Node Labeling Schemes for Dynamic XML Documents Reconsidered. In Data & Knowledge Engineering, Elsevier, 2006.
5. Haustein, M. P.: Feingranulare Transaktionsisolation in nativen XML-Datenbanksystemen, Verlag Dr. Hut, München, Januar 2006.
6. Haustein, M. P., Härder, T., Mathis, C., Wagner, M.: DeweyIDs - The Key to Fine-Grained Management of XML Documents. In Proc. 20th Brazilian Symposium on Databases, Uberlandia, Minas Gerais, Brazil, October 2005, pp.85-99.
7. Lim, L., Wang, M., Padmanabahn, S., Vitter, Jeffrey S., Parr, R.: XPathLearner: An On Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation. In Proceedings of the 28th VLDB Conference. Hong Kong, China, 2002, pp.442-453.
8. Mathis, C., Härder, T.: Hash-Based Structural Join Algorithms. In Proceedings of the 2nd Int. Workshop on Database Technologies for Handling XML Information on the Web, Munich, March 2006.
9. Mathis, C., Härder, T., Haustein, M.: Locking-Aware Structural Join Operators for XML Query Processing. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, June 2006.
10. Mathis, C., Härder, T.: A Query Processing Approach for XML Database Systems. In Proc. 17th Workshop "Grundlagen von Datenbanken", Wörlitz, Sachsen-Anhalt, Mai 2005, pp. 89-93.
11. McHugh, J., Widow, J.: Query Optimization for XML. In Proceedings of the 25th VLDB Conference. Edinburgh, Scotland, 1999, pp. 315-326.
12. Polyzotis, N., Garofalakis, M.: Structure and Value Synopses for XML Data Graphs. In Proceedings of the 28th VLDB Conference. Hong Kong, China, 2002, pp. 466-477.
13. Wang, W., Jiang, H., Lu, H., Yu, Jeffrey Xu.: Bloom Histogram: Path Selectivity Estimation for XML Data with Updates. In Proceedings of the 30th VLDB Conference. Toronto, Canada, 2004, pp.240-251.
14. XPATH XML Path Language 2.0. W3C Candidate Release (Nov. 2005).
15. XQuery 1.0: An XML Query Language. W3C Candidate Release (Nov. 2005).