OrientStore: A Schema Based Native XML Storage System

Xiaofeng Meng† Daofeng Luo† Mong Li Lee‡ Jing An†

† Information School
Renmin University of China
Beijing 100872, China
xfmeng@mail.ruc.edu.cn

‡School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
leeml@comp.nus.edu.sg

1. Introduction

The increasing number of XML repositories has provided the impetus to design and develop systems that can store and query XML data efficiently. Research to improve system performance has been largely concentrated on indexing paths and optimizing XML queries. In fact, the storage configuration of XML data on disk also has an impact on the efficiency of an XML data management system.

Existing XML storage strategies can be classified into two categories: native XML storage and non-native XML storage. The main distinction between them is their data model. The former is based on the XML Data Models such as Document Object Model (DOM), and Object Exchange Model (OEM), while the latter is based on the traditional relational data model, or object-oriented data model. An evaluation of the alternative non-native storage strategies has been given in [6]. Here, we will focus on native XML storage strategies.

Several native storage strategies have been developed in [1,2,3,5,8,11]. These can be classified into Element-Based (EB), Subtree-Based (SB) and Document-Based (DB). Both the Lore system [3] and TIMBER [1] utilize the classic EB strategy, where each element is an atomic unit of storage and is organized in a pre-ordered manner. Natix [2] is a well-known SB strategy. It divides the XML document tree into subtrees according to the physical page size, such that each subtree is a record. The sizes of the subtrees are kept as close as possible to the size of the physical page. A split matrix is defined to ensure that correlated element nodes remain clustered. Similar to the EB strategy, the records are stored in a pre-ordered way. The storage module in the Apache Xindice system [8] employs the DB strategy, whereby the entire XML document constitutes a single record.

Other variations of storage strategies can be found in NeoCore XMS [11] where the XML data is first flattened to expose only the pure XML information, before they are passed on to a digital pattern process to create icons.. Tamino [5] is a leading commercial native XML database, but details of its storage structure are fairly sketchy.

All the above native storage strategies are schema-independent, when schema information in the form of XML Schema or DTD is usually available or even indispensable. In order to facilitate data exchange, a standard schema (or DTD) is typically defined on the underlying XML files and published. Examples of available standard schema or DTDs include Chemical Markup Language, Mathematical Markup Language, News Markup Language, etc. Popular XML datasets such as the DBLP [9], Movie database [10], Shakespeare' Play [12] and XMark [4] come with its own DTD.

The availability of schema information is crucial to data exchange applications, and query optimizations. We observe that schema information also has a key role to play in designing efficient and effective storage strategies for XML management systems.

In this work, we develop a prototype native XML storage system, called OrientStore. OrientStore implements two schema-guided storage strategies, namely Element-Based Clustering (EBC), and Logical Partition-Based Clustering (LPC) strategies.

In contrast with the present storage systems for XML data, OrientStore has the following unique features:

a. It concretely investigates how schema information can be utilized to reduce the storage requirement and the response time of queries.

b. It implements two schema-guided storage strategies: EBC and LPC. These strategies cluster correlated data in different ways to reduce the number of I/Os required during retrieval.

**Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003**

2. System Overview

Figure 1 shows the architecture of OrientStore. It has the following components:

a. The underlying file manager communicates with file system to create and delete data files, in units of fixed size such as 8 MB.

b. The storage manager manages the storage space of the file in units of a physical page, which is set to 8 KB. This is equivalent to the physical page size of the operating system.

c. The buffer manager employs the standard Least Recently Used (LRU) replacement policy. All read and write requests are sent to the buffer manager.

d. The external data processor communicates with other database systems to retrieve the data stored in these systems.

e. The access manager provides a uniform access interface to record manager, index manager, and meta-data manager. Details of the buffer manager and storage manager are hidden from them.

f. The record manager formats a record into (and from) a byte-stream.

g. The data manager provides functions for importing, exporting, and retrieving the root of a document, etc.

h. The index manager keeps track of indices built on the XML data. OrientStore supports two types of indices: value index and path index [7].

i. The meta-data manager maintains the meta-data of the system, one of which is the schema information. Schema information can be viewed as a graph, or schema graph. The schema graph is encoded such that each node in the graph is represented by a unique 4-bytes code, referred to as the schemaNodeID.

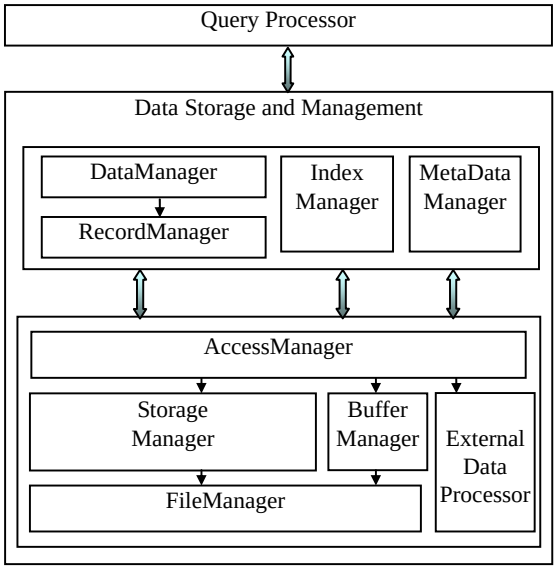


Figure1: Architecture of OrientStore.

3. Utilizing Schema Information

As mentioned earlier, we can take advantage of the information available in an XML schema to reduce storage space consumption and improve query processing. Specifically, OrientStore utilizes the schema information in the following four ways:

a. Since the schema graph is encoded, we can replace the element tag names by their corresponding schemaNodeIDs. Each schemaNodeID takes up 4-bytes, which is much smaller than the space requirement for a typical tag name, thereby reducing the overall space consumption.

b. Relative paths "//" and paths with wildcards "*" are common in XPath and XQuery. Based on the schema information, we can translate the relative path into one or more absolute paths. Path expressions in XML queries can also be validated before it is evaluated by the system.

c. Tag names in a query are also mapped to their corresponding schemaNodeID. Queries are subsequently processed using the schemaNodeIDs. Based on the encoded schemaNodeIDs, the system can quickly determines whether two elements have ancestor-descendant relationship or not. A node (and its descendant nodes) will not be further evaluated if it is not found to be the ancestor of target nodes. As a result, the query evaluation process can be dramatically accelerated since it avoids the full traversal of the entire document tree.

d. The schema also serves as a guide to how the XML data files can be stored such that the number of I/Os required for their retrieval is reduced. The following section elaborates on the proposed schema-guided storage strategies.

4. Schema-Guided Storage Plan

OrientStore exploits schema information in the design and implementation of two storage strategies: *Element-Based Clustering (EBC)*, and *Logical Partition-Based Clustering (LPC)* strategies.

4.1 Element-Based Clustering

The Element-Based Clustering (EBC) storage strategy is similar to element-based storage plan as used in Lore and TIMBER in that each element node is a record. However, instead of storing the records in a pre-order fashion, EBC clusters the element records such that records with the same schemaNodeID are placed close together.

Figure 2 shows a sample XML document and its corresponding DTD. The EBC storage plan will cluster all the *title* elements together with their text values together. Note that pointers (either physical or logical) are needed to link the parent records to their children records. Given a query wants to find the titles of books published by a

specified publisher, OrientStore is able to retrieve all the relevant books directly, without involving other unnecessary element nodes, thus reducing the number of I/Os required.

```
<vendor>
  <name>Star</name>
  <book>
    <publisher> ABC</publisher>
    <title>C++ </title>
  </book>
  <book>
    <publisher>DEF</publisher>
    <title>Java </title>
  </book>
</vendor>

<!ELEMENT vendor (name,book*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT book (publisher, title)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

Figure 2: Sample XML document and its DTD.

4.2 Logical Partition-Based Clustering

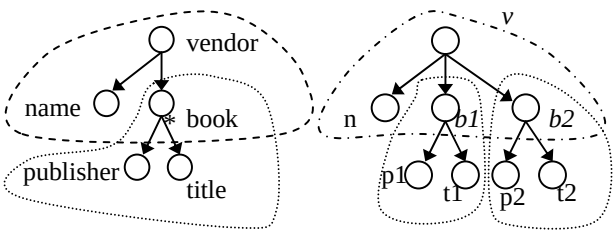
The Logical Partition-Based Clustering (LPC) storage strategy partitions the schema graph into *semantic block*s. A *semantic block* describes a relatively integrated logical unit. Consider Figure 2 where the element *book* together with its children *title* and *publisher* constitutes a semantic block. We use the following heuristic to obtain semantic blocks:

A node in a schema graph is the start (or the root) of a semantic block if:

a. it is a root of the schema graph, or

b. it has a cardinality of '*' or '+', and it has child nodes.

Figure 3(a) shows the schema graph and the semantic blocks obtained for the XML document in Figure 2: *vendor (name, book)* and *book (publisher, title).*



a) Semantic Blocks b) Records

Figure 3: LPC Storage

A record in the LPC storage strategy is an instance of a semantic block. In Figure 3(b), the records *b1 (p1, t1)* and *b2 (p2, t2)* are instances of the semantic block *book (publisher, title),* while the record *v(n, b1, b2)* is an instance of *vendor (name, book).*

The records obtained by the LPC storage strategy has the following characteristics and advantages over those obtained by existing schema-independent storage strategies, particularly the Subtree-Based strategy:

a. The size of a LPC record is typically much smaller than the size of a physical page. That is, the granularity of a record obtained from the LPC strategy lies between the records obtained by the Element-Based and Subtree-Based storage strategies.

b. A LPC record is an instance of a semantic block, while a record in the Subtree-Based storage plan is a physical unit that is determined by the page size. Hence, the element nodes in a LPC record are more correlated than those in the Subtree-Based strategy. In fact, the element nodes in a semantic block are more likely to be retrieved together in a query.

Further, all the instances of the same semantic block are clustered together. Thus the records *b1 (p1, t1)* and *b2 (p2, t2)* in Figure 2(b) will be stored in a physical page, while *v (n, b1, b2)* may be stored in another physical page. This implies fewer I/Os for a query.

For comparison purposes, we also implemented the schema-independent Element-Based (EB) and Subtree-Based (SB) storage strategies in OrientStore. Figure 4 shows the results of our experiments. We use XMark dataset and run 4 kinds of queries (Q1:point query, Q2:aggregate query, Q3:join, Q4:ordered access query). We found that on average, the number of I/Os incurred by the proposed LPC and EBC storage strategies is 30% less than that incurred by EB and SB.

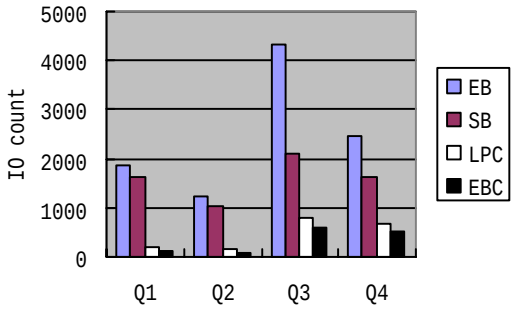


Figure 4: Comparative Experiment

5. Plan of Demonstration

For this demonstration, we will show the performance of these schema-independent storage plans, and the proposed schema-guided strategies, EBC and LPC. A wide range of queries will be issued on the schemas of datasets such as

XMark, DBLP, Shakespeare, and the response time recorded.

6. Acknowledgements

This research was partially supported by the grants from 863 High Technology Foundation of China under grant number 2002AA116030, the Natural Science Foundation of China (NSFC) under grant number 60073014, 60273018, the Key Project of Chinese Ministry of Education (No.03044) and the Excellent Young Teachers Program of Chinese Ministry of Education (EYTP).

7. References

[1] H. V. Jagadish, Shurug AL-Khalifa, et al. TIMBER: A Native XML Database. Technical Report, University of Michigan, April 2002.

[2] C.-C. Kanne and G. Moerkotte.. Efficient Storage of XML data. In Proceedings of 16th ICDE, page 198. San Diego, California, USA, February 2000.

[3] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. SIGMOD Record, Vol.26(3):54-66, September 1997.

[4] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, R. Busse. XMark: A Benchmark for XML Data Management. In Proceedings of 28th VLDB, pages 974-985. Hong Kong, China, August 2002.

[5] H. Schoning. Tamino – A DBMS Designed for XML. In Proceedings of 17th ICDE, pages 149-154. Heidelberg, Germany, April 2001.

[6] F. Tian, D. J. DeWitt, J. Chen, C. Zhang. The Design and Performance Evaluation of Alternative XML Storage Strategies. SIGMOD Record, Vol.31(1), March 2002.

[7] J.Wang. SUPEX: A Schema-Guided Path Index for XML Data. In Proceedings of 28th VLDB. Hong Kong, China, August 2002.

[8] Apache Xindice. http://XML.apache.org/xindice/

[9] DBLP dataset. http://dblp.uni-trier.de/xml/

[10] Internet Movie Database. http://imdb.com

[11] NeoCore XMS. http://www.neocore.com/

[12] Shakespeare dataset. http://sunsite.uncedu/pub/sun-info/standards/xml/eg/