Efficient Relational Storage and Retrieval of XML Documents

Albrecht Schmidt, Martin Kersten, Menzo Windhouwer, and Florian Waas

Centre for Mathematics and Computer Science (CWI)
Kruislaan 413, 1098 SJ Amsterdam
The Netherlands
email: firstname.lastname@cwi.nl

Abstract. In this paper, we present a data and an execution model that allow for efficient storage and retrieval of XML documents in a relational database. The data model is strictly based on the notion of binary associations: by decomposing XML documents into small, flexible and semantically homogeneous units we are able to exploit the performance potential of vertical fragmentation. Moreover, our approach provides clear and intuitive semantics, which facilitates the definition of a declarative query algebra. Our experimental results with large collections of XML documents demonstrate the effectiveness of the techniques proposed.

1 Introduction

XML increasingly assumes the role of the *de facto* standard data exchange format in Web database environments. Modeling issues that arise from the discrepancy between semi-structured data on the one hand side and fully structured database schemas on the other have received special attention. Database researchers provided valuable insights to bring these two areas together. The solutions proposed include not only XML domain specific developments but also techniques that build on object-oriented and relational database technology (*e.g.*, see [1, 6, 7, 9–11, 14–16]).

To make XML the language of Web databases, performance issues are the upcoming challenge that has to be met. Database support for XML processing can only find the widespread use that researchers anticipate if storage and retrieval of documents satisfy the demands of impatient surfers.

In this paper, we are concerned with providing effective tools for the management of XML documents. This includes tight interaction between established standards on the declarative conceptual level like the DOM [18] and efficient physical query execution. Starting from the syntax tree representation of a document, we propose a data model that is based on a complete binary fragmentation of the document tree. This way, all relevant associations within a document like parent-child relationships, attributes, or topological orders can be intuitively described, stored and queried. In contrast to general graph databases like Lore [1], we draw benefit from the basic tree structure of the document and incorporate

```
<bibliography>
  <article key="BB88">
    <author>Ben Bit</author>
    <title>How To Hack</title>
  </article>
  <article key="BK99">
    <editor>Ed Itor</editor>
    <author>Bob Byte</author>
    <author>Ken Key</author>
    <title>Hacking & RSI</title>
  </article>
</bibliography>
```

Fig. 1. XML document and corresponding syntax tree

information about the association's position within the syntax tree relative to the root into our data model. References such as IDREFs that escape the tree structure are taken care of by views on the tree structure. Associations that provide semantically related information are stored together in the binary relations of the database repository. Along with the decomposition schema we also present a method to translate queries formulated on paths of the syntax tree into expressions of an algebra for vertically fragmented schemas [3].

Our approach is distinguished by two features. Firstly, the decomposition method is independent of the presence of DTDs, but rather explores the structure of the document at parse time. Information on the schema is automatically available after the decomposition. Secondly, it reduces the volume of data irrelevant to a query that has to be processed during querying. Storing associations according to their context in the syntax tree provides tables that contain semantically closely related information. As a result, data relevant for a given query can be accessed directly in form of a separate table avoiding large and expensive scans over irrelevant data making associative queries with path expressions rather inexpensive. Especially the need for hierarchical projections and semijoins vanishes completely.

Reservations exist that a high degree of fragmentation might incur increased efforts to reconstruct the original document, or parts of it. However, as our quantitative assessment shows, the number of additional joins is fully made up for as they involve only little data volume. Our approach displays distinctly superior performance compared to previous work.

2 Data Model and Algebra

XML documents are commonly represented as syntax trees. With **string** and **int** denoting sets of character strings and integers and **oid** being the set of unique object identifiers, we can define an XML document formally (*e.g.*, see [19]):

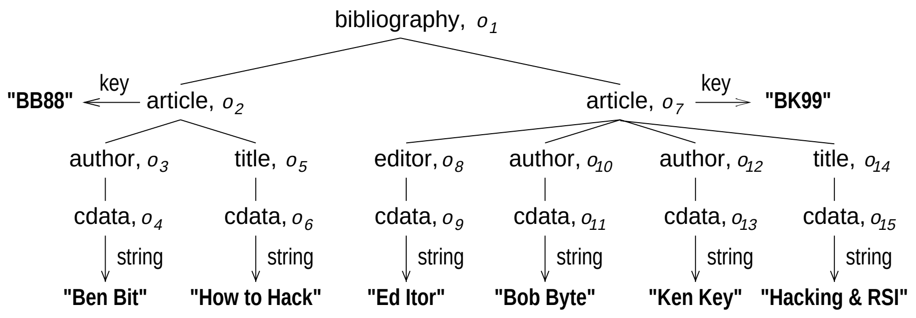

Fig. 2. Syntax tree of example document

Definition 1. *An* XML document $d = (V, E, r, label_E, label_A, rank)$ *is a rooted tree with nodes* V *and edges* $E \subseteq V \times V$ *and a distinguished node* $r \in V$*, the root node. The function* $label_E : V \rightarrow \textbf{string}$ *assigns labels to nodes, i.e., elements;* $label_A : V \rightarrow \textbf{string} \rightarrow \textbf{string}$ *assigns pairs of strings, attributes and their values, to nodes. Character Data (CDATA) are modeled as a special 'string' attribute of cdata nodes,* $rank : V \rightarrow \textbf{int}$ *establishes a ranking between sibling nodes. For elements without any attributes* $label_A$ *maps to the empty set.*

Figure 1 shows an XML document which describes a fragment of a bibliography; the corresponding syntax tree is displayed in Figure 2. The representation is largely self-explanatory, o_i denote object identifiers (OIDs) whose assignment is arbitrary, *e.g.*, depth-first traversal order. We apply the common simplification not to differentiate between PCDATA and CDATA nor do we take rich datatypes into account. Note that OID assigned nodes represent only elements and not attributes.

2.1 Preliminaries

Before we discuss techniques how to store a syntax graph as a database instance, we introduce the concepts of *associations* and *path summaries*. They identify spots of interest and constitute the basis for the Monet XML Model.[1]

Definition 2. *A pair* $(o, \cdot) \in \textbf{oid} \times (\textbf{oid} \cup \textbf{int} \cup \textbf{string})$ *is called an* association.

The different types of associations describe different parts of the tree: associations of type $\textbf{oid} \times \textbf{oid}$ represent edges, *i.e.*, parent-child relationships. Attribute values (including character data, represented by vertices with label *'string'*, that start from *'cdata'* labelled nodes) are modeled by associations of type $\textbf{oid} \times \textbf{string}$, while associations of type $\textbf{oid} \times \textbf{int}$ are used to preserve the topology of a document.

[1] We chose the name Monet XML Model because the home-grown database engine Monet [3] serves as implementation platform.

Definition 3. *For a node o in the syntax tree, we denote the sequence of labels along the path (vertex* and *edge labels) from the root to o with* path(o).

As an example, consider the node with OID o_3 in Figure 2; its path is *bibliography* $\overset{e}{\to}$ *article* $\overset{e}{\to}$ *author*. The corresponding character data string "Ben Bit" has path *bibliography* $\overset{e}{\to}$ *article* $\overset{e}{\to}$ *author* $\overset{e}{\to}$ *cdata* $\overset{a}{\to}$ *string*, where $\overset{e}{\to}$ denotes edges to elements and $\overset{a}{\to}$ to attributes.

Paths describe the schematic position of the element in the graph relative to the root node, and we use $path(o)$ to denote the *type* of the association $(\cdot, o)$. The set of all paths in a document is called the document's *path summary.*

2.2 The Monet XML Model

As we pointed out at the beginning, the question central to querying XML documents is how to store the syntax tree as a database instance that provides efficient retrieval capabilities. Given Definition 1 the tree could be stored using a single database table for the parent-child relations (similar to [17]), another one for the elements labels and so on. Though space effective, such a decomposition makes querying expensive by enforcing scans over large amounts of data irrelevant to a query instance, since structurally unrelated data are possibly stored in the same tables. Even if the query consist of a few joins only, large data volumes may have to be processed (see [9] for a discussion of storage schemes of this kind).

We pursue a rather different approach using the structures defined above, *i.e.*, storing all associations of the same type in the same *binary relation.* A relation that contains the tuple $(\cdot, o)$ is named $path(o)$, and, conversely, a tuple is stored in exactly one relation. This idea results in the following definition:

Definition 4. *Given an XML document d, the* Monet transform *is a quadruple $M_t(d) = (r, \mathbf{R}, \mathbf{A}, \mathbf{T})$ where*

- $\mathbf{R}$ *is the set of binary relations that contain* all *associations between nodes;*
- $\mathbf{A}$ *is the set of binary relations that contain* all *associations between nodes and their attribute values, including character data;*
- $\mathbf{T}$ *is the set of binary relations that contain* all *pairs of nodes and their rank;*

r remains the root of the document.

Encoding the *path* to a component into the name of the relation often achieves a significantly higher degree of semantic fragmentation than implied by plain data guides [10]. In other words, we use *path* to group semantically related associations into the same relation. As a direct consequence of the decomposition schema, we do not need to introduce novel features on the storage level to cope with irregularities induced by the semi-structured nature of XML, which are typically taken care of by NULLs or overflow tables [7]. Moreover, it should be noted, that the complete decomposition is linear in the size of the document with respect to running time. Concerning memory requirements, it is in $O(h)$, h being

$$\begin{aligned}
\mathit{bibliography} \xrightarrow{e} \mathit{article} &= \{\langle o_1, o_2\rangle, \langle o_1, o_7\rangle\},\\
\mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{author} &= \{\langle o_2, o_3\rangle, \langle o_7, o_{10}\rangle, \langle o_7, o_{12}\rangle\},\\
\mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{author} \xrightarrow{e} \mathit{cdata} &= \{\langle o_3, o_4\rangle, \langle o_{10}, o_{11}\rangle, \langle o_{12}, o_{13}\rangle\},\\
\mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{author} \xrightarrow{e} \mathit{cdata} \xrightarrow{a} \mathit{string} &= \{\langle o_4, \text{“Ben Bit”}\rangle, \langle o_{11}, \text{“Bob Byte”}\rangle, \langle o_{13}, \text{“Ken Key”}\rangle\},\\
\mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{title} &= \{\langle o_2, o_5\rangle, \langle o_7, o_{14}\rangle\},\\
\mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{title} \xrightarrow{e} \mathit{cdata} &= \{\langle o_5, o_6\rangle, \langle o_{14}, o_{15}\rangle\},\\
\mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{title} \xrightarrow{e} \mathit{cdata} \xrightarrow{a} \mathit{string} &= \{\langle o_6, \text{“How to Hack”}\rangle, \langle o_{15}, \text{“Hacking \& RSI”}\rangle\},\\
\mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{editor} &= \{\langle o_7, o_8\rangle\},\\
\mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{editor} \xrightarrow{e} \mathit{cdata} &= \{\langle o_8, o_9\rangle\},\\
\mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{editor} \xrightarrow{e} \mathit{cdata} \xrightarrow{a} \mathit{string} &= \{\langle o_9, \text{“Ed Itor”}\rangle\},\\
\mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{a} \mathit{key} &= \{\langle o_2, \text{“BB88”}\rangle, \langle o_7, \text{“BK99”}\rangle\}\}
\end{aligned}$$

Table 1. Monet transform M_t of the example document

the height of the syntax tree, in addition to the space the binary relations in the database engine occupy, *i.e.*, it is not necessary to materialize the complete syntax tree.

Proposition 1. *The above mapping is lossless, i.e., for an XML document d there exists an inverse mapping M_t^{-1} such that d and $M_t^{-1}(M_t(d))$ are isomorphic.*

A sketch of the proof of Proposition 1 is given in the appendix. Table 1 shows the Monet transform of the example document.

In addition to the relational perspective we adhered to so far, the Monet transform also enables an object-oriented perspective, *i.e.*, object being interpreted as node in the syntax tree, which is often more intuitive to the user and is adopted by standards like the DOM [18]. Particularly in querying, approaches that bear strong similarities with object-oriented techniques have emerged. Given the Monet transform, we have the necessary tools at hand to reconcile the relational perspective with the object-oriented view.

It is natural to re-assemble an object with OID o from those associations whose first component is o: *e.g.*, the node with OID o_2 is easily converted into $object(o_2) = \{key\langle o_2, \text{“BB88”}\rangle, author\langle o_2, o_3\rangle, title\langle o_2, o_5\rangle\}$, an instance of a suitably defined class *article* with members *key*, *author* and *title*. However, XML is regarded as an incarnation of the semi-structured paradigm. One consequence of this is that we cannot expect all instances of one type to share the same structure. In the example, the second publication does have an *editor* element whereas the first does not. We therefore distinguish between two kinds of associations: (strong) associations and weak associations. Strong associations constitute the structured part of XML – they are present in every instance of a type; weak associations account for the *semi*-structured part: they may or may not appear in a given instance. Objects o_2 and o_7 reflect this: o_7 has a *editor* member whereas o_2 has not. Therefore, we define the following:

Definition 5. *An object corresponding to a node o in the syntax tree is a set of strong and weak associations $\{A_1\langle o, o_1\rangle, A_2\langle o, o_2\rangle, \ldots\}$.*

The next question we address directly arises from the modeling of objects: How can we re-formulate queries from an object-oriented setting to queries in relational Monet XML?

2.3 Execution Model and Algebra

The unified view provided by the Monet XML model extends directly to querying. For the relational layer, a multitude of operators implementing the relational algebra, including specialties intrinsic to vertical fragmented schemas, have been proposed. Hence, we omit a discussion of technical issues concerning bare, relational query processing in the context of vertical fragmentation and refer the interested reader to [3] for a comprehensive overview.

More interesting is the actual translation of an OQL-like query to match the facilities of the underlying query execution engine. We only outline the translation by an example query. The process bears strong resemblance to mapping techniques developed to implement object-oriented query interfaces on relational databases; thus, we can resort to the wealth of techniques developed in that field. See [4] for a comparative analysis of different query languages for XML.

Consider the following query which selects those of Ben Bit's publications whose titles contain the word 'Hack'; the semantics of the statements are similar to [2]:

select p
from $\mathit{bibliography} \overset{e}{\to} \mathit{article}\ \ p$,
$p \overset{e}{\to} \mathit{author} \overset{e}{\to} \mathit{cdata}\ \ a$,
$p \overset{e}{\to} \mathit{title} \overset{e}{\to} \mathit{cdata}\ \ t$
where $a =$ "Ben Bit" **and** t **like** "Hack";

The query consists of two blocks, a *specification* of the elements involved, which translates to computing the proper binary relations, and *constraints* that define the actual processing. For resolving path expressions, we need to distinguish two types of variables in the **from** clause: variables that specify sets, p in the example, and variables, which specify associations, a and t.

We collapse each path expression that is not available in the database by joining the binary relations along the path specification. This establishes an association between the first and last element of the path. Finally, we take the intersection of the specified elements. Matching the variables against the running example, the **from** clause specifies the following elements:

$$\begin{aligned} p &= \{o_2, o_7\}, \\ \mathit{assoc}(p \to a) &= \{(o_2, \text{“Ben Bit”}), (o_7, \text{“Bob Byte”}), \\ &\quad\ (o_7, \text{“Ken Key”})\}, \\ \mathit{assoc}(p \to t) &= \{(o_2, \text{“How To Hack”}), \\ &\quad\ (o_7, \text{“Hacking \& RSI”})\} \end{aligned}$$

Queries containing regular expressions over paths directly benefit from the availability of the path summary. Standard methods for the evaluation of regular expressions can be applied to the textual representation of the paths and enable the immediate selection of the candidate relations.

The evaluation of the **where** clause is not of particular interest in this context. Though processing of binary tables differs from the conventional relational model in several aspects, these differences have no direct impact on our method.

2.4 Optimization with DTDs

As XML documents are not required to conform to DTDs we do not assume that they do. However, in this section we show that our data model is flexible enough to take advantage of additional domain-knowledge in the form of DTDs or XML Schema specifications. Again, the first-class paths in Monet XML are

```
<!ELEMENT bibliography (article*)>
<!ELEMENT article (editor?, author*, title)>
<!ATTLIST article key CDATA #REQUIRED>
<!ELEMENT editor (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

Fig. 3. DTD for example document

the focal feature necessary for a seamless integration. We present an approach that is similar in spirit to [14].

For motivation, consider again the example document in Figure 1 and 2 suppose we are given the DTD in Figure 3. The DTD says that each publication may only have a single title element. Given this rule, we can collapse each path from the publication nodes to the character data of title elements without losing information; thus,

$$\begin{aligned}\{\ & \mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{title}\{\langle o_2, o_5\rangle, \langle o_7, o_{14}\rangle\},\\ & \mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{title} \xrightarrow{e} \mathit{cdata}\{\langle o_5, o_6\rangle, \langle o_{14}, o_{15}\rangle\},\\ & \mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{title} \xrightarrow{e} \mathit{cdata} \xrightarrow{a} \mathit{string}\{\langle o_6, \text{“How to Hack”}\rangle,\\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle o_{15}, \text{“Hacking \& RSI”}\rangle\}\ \}\end{aligned}$$

may be reduced to

$$\begin{aligned}\{\ & \mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{title}\{\langle o_2, o_5\rangle, \langle o_7, o_{14}\rangle\},\\ & \mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{e} \mathit{title} \xrightarrow{a} \mathit{string}\{\langle o_5, \text{“How to Hack”}\rangle,\\ & \qquad\qquad\qquad\qquad\qquad\qquad \langle o_{14}, \text{“Hacking \& RSI”}\rangle\ \}\}.\end{aligned}$$

That is we take advantage of DTDs by identifying and subsequently collapsing 1 : 1 relationships to reduce storage requirements and the number of joins in query processing. The result of hierarchically joining the associations takes the place of the original data. Some of these 1 : 1 relationships can be inferred from a DTD, others require domain-specific knowledge: our common sense knowledge of bibliographies tells us that in bibliographies the only elements whose order is important are author and editor elements. Thus, we may, on the one hand, drop all rank relations that do not belong to author or editor tags and furthermore reduce the before mentioned path to:

$$\{\ \mathit{bibliography} \xrightarrow{e} \mathit{article} \xrightarrow{a} \mathit{title}\{\langle o_2, \text{“How to Hack”}\rangle, \langle o_7, \text{“Hacking \& RSI”}\rangle\ \}.$$

Note that we apply this technique not to the DTDs themselves to derive a storage schema but rather simplify the paths present in the actual document instance.

Documents	size in XML	size in Monet XML	#Tables	Loading
ACM Anthology	46.6 MB	44.2 MB	187	30.4 s
Shakespeare's Plays	7.9 MB	8.2 MB	95	4.5 s
Webster's Dictionary	56.1 MB	95.6 MB	2587	56.6 s

Table 2. Sizes of document collections in XML and Monet XML format

		Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
1A	Monet XML	1.2	5.6	6.8	8.0	4.4	4.9	5.0	5.0	8.8	12.7
2A	SYU / Postgres	150	180	160	180	190	340	350	370	1300	1040
1B	Monet XML	–	4.4	5.6	6.8	3.2	3.7	3.8	3.8	7.6	11.5
2B	SYU / Postgres	–	30	10	30	40	190	200	220	1150	890

Table 3. Comparison of response times for query set of SYU

3 Quantitative Assessment

We assess the techniques proposed with respect to size of the resulting database, as well as querying and browsing the database. As application domains we chose readily available XML document collections: the ACM SIGMOD Anthology [12], Webster's Dictionary [8], and Shakespeare's Plays [5].

We implemented Monet XML within the Monet database server [3]. The measurements were carried out on an Silicon Graphics 1400 Server with 1 GB main memory, running at 550 MHz. For comparisons with related work, we used a Sun UltraSPARC-IIi with 360 MHz clock speed and 256 MB main memory.

Database Size. The resulting sizes of the decomposition scheme are a critical issue. Theoretically, the size of the path summary can be linear in the size of the document as the worst case – if the document is completely *un*-structured. However, in practical applications, we typically find large structured portions within each document so that the size of the path summary and therefore the number of relations remains small. Table 2 shows the database sizes for our examples in comparison with the size of the original XML code. The third column contains the number of tables, *i.e.*, the size of the path summary. The last column shows the complete time needed to parse, decompose and store the documents.

It leaps out that the Monet XML version of the ACM Anthology is of smaller size than the original document. This reduction is due to the 'automatic' compression inherent in the Monet transform (tag names are stored only once as meta information) and the removal of redundantly occurring character data. For example there are only few different publishers compared to the number of entries in general. In the decomposition, full entries of these fields can be replaced with references; this is done automatically by the DBMS. We can expect similar effects to occur with other decomposition schemas, like object-oriented mappings.

Scaling. In order to inspect the scaling behavior of our technique we varied the size of the underlying document. In doing so, we took care to maintain the

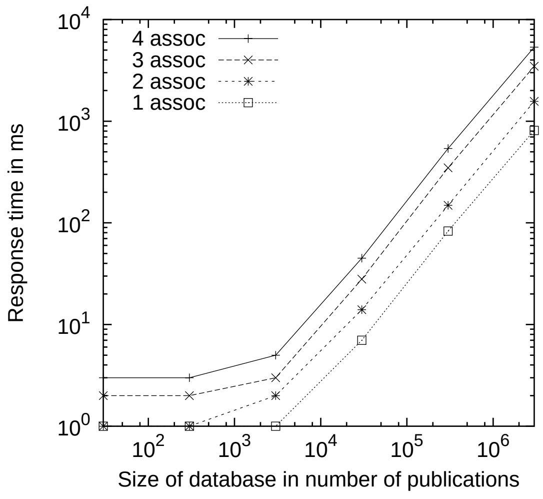

Fig. 4. Scaling of document

ratio of different elements and attributes of the original document. We scaled the ACM Anthology from 30 to $3 \cdot 10^6$ publications which corresponds to XML source sizes between 10KB and 1GB. The database sizes and the insertion times scaled linear in the size of the XML document.

Querying. To test for query performance under scaling we ran 4 queries consisting of path expressions of length 1 through 4 for various sizes of the Anthology. As Figure 4 shows, the response times for each query, given as a function of the size of the document, is linear in the size of the database. Only for small sizes of the database, the response time is dominated by the overhead of the database system. Notice, both axes are logarithmic.

Only few of the performance analyses published so far offer the possibility to reproduce and compare results, which makes meaningful comparison difficult at this time. The results we use to compare Monet XML against were reported in [15] who implemented their algorithms as a front-end to Postgres. In [15], the authors propose a set of 10 queries using Shakespeare's plays [5] as an application domain. We refer to their approach as SYU in the following. In Table 3 we contrasted response times of Monet XML with SYU obtained from experiments on the abovementioned Sun Workstation.

The figures display a substantial difference in response time showing that Monet XML outruns the competitor by up to two orders of magnitude (rows 1A,2A). The times for SYU include a translation of XQL to SQL that is handled outside the database server. To allow for this difference, we additionally computed the response times relative to query 1 for both systems separately, assuming that preprocessing costs have a constant contribution. These figures exhibit actual query processing time only (rows 1B,2B). Monet XML shows an

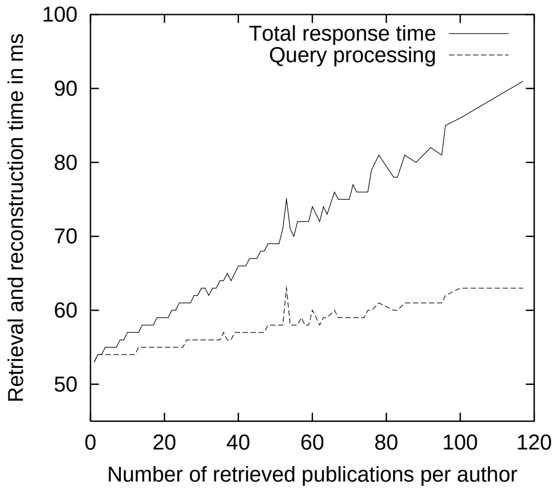

Fig. 5. Response time vs. result size

increase of processing time by less than 12 ms whereas SYU is up to 1150 ms slower than its fastest response time.

An analysis of the figures exhibits the advantages of the Monet model. While SYU store basically all data on a single heap and have to scan these data repeatedly, the Monet transform yields substantially smaller data volumes. In some extreme cases, the query result is directly available in Monet XML without any processing and only needs to be traversed and output. Another noticeable difference concerns the complexity of queries: the straight-forward semantics of the Monet XML model result in relatively simple queries; conversely, the compiled SQL statements that SYU present are quite complex.

The comparison with Lore [13] exhibited essentially the same trends on small document instances. However, we were not able to bulkload and query larger documents like the ACM anthology as Lore requested more than the available 1 GB main memory. In contrast, using Monet XML we engineered a system functionally equivalent to the online DBLP server [12] that operated in less than 130 MB.

Browsing a database. Our last experiments aim at assessing the systems capabilities with respect to browsing. As an example consider a typical query as it is run on the Anthology server several thousand times a day: Retrieve all conference publications for a given author. Clearly, the size of the output may vary drastically and it is of particular interest for a browsing session that response times are kept low independent of the size of the answer.

Figure 5 shows both the total response time including textual rendering and response time of the repository. As expected, the time for rendering the output increases significantly yet linear in the result size. However, the response time of the repository increases at significantly lower rate. This is due to the reconstruction of the associations in form of joins rather than chasing individual chains of pointers. Even for authors with a large number of publications the overall response time is well under one tenth of a second, which makes interactive browsing affordable. Also note that the lower line in Figure 5 could also be interpreted as the cost of constructing a view while the upper line additionally includes rendering the view to textual XML.

The results presented demonstrate the performance potential of our approach deploying fully vertical fragmentation. As the low response times show, reducing the data volume involved in single database operations on the expense of additional joins pays very well not only in terms of overall performance but also when scaling is an issue.

4 Conclusions

We presented a data model for efficient processing of XML documents. Our experiences show that it is worth taking the plunge and fully decompose XML documents into binary associations. The experimental results obtained with a prototype implementation based on Monet underline the viability of our approach: the effort to reduce data volume quickly pays off as gains in efficiency. Overall, our approach combines the elegance of clear semantics with a highly efficient execution model by means of a simple and effective mapping between XML documents and a relational schema.

Concerning future work, we will concentrate on exploring possibilities of parallel processing and efficient handling of multi-query workloads as found in typical interactive Web-based information systems. As we have seen with own experiments, there is also the need for a general, standardized methodology that allows conclusive performance analyses and facilitates comparisons of different approaches.

References

1. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.
2. C. Beeri and Y. Tzaban. SAL: An Algebra for Semistructured Data and XML. In *International Workshop on the Web and Databases*, pages 37–42, Pennsylvania, USA, 1999.
3. P. A. Boncz and M. L. Kersten. MIL Primitives for Querying a Fragmented World. *The VLDB Journal*, 8(2):101–119, 1999.
4. A. Bonifati and S. Ceri. Comparative Analysis of Five XML Query Languages. *ACM SIGMOD Record*, 1(29):68–79, 2000.
5. J. Bosak. Sample XML documents. `shakespeare.1.01.xml.zip`, available at `ftp://sunsite.unc.edu/pub/sun-info/standards/xml/eg/`.

6. P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 505–516, Montreal, Canada, 1996.
7. A. Deutsch, M. F. Fernandez, and D. Suciu. Storing Semistructured Data with STORED. In *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, pages 431–442, Philadephia, PA, USA, 1999.
8. M. Dyck. The GNU version of The Collaborative International Dictionary of English, presented in the Extensible Markup Language. Available at `http://metalab.unc.edu/webster/`.
9. D. Florescu and D. Kossmann. Storing and Querying XML Data Using an RDBMS. *Data Engineering Bulletin*, 22(3), 1999.
10. R. Goldman and J. Widom. Dataguides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 436–445, Athens, Greece, 1997.
11. C. Kanne and G. Moerkotte. Efficient Storage of XML Data. In *Proceedings of the 16th International Conference on Data Engineering*, page 198, 2000.
12. M. Ley. DBLP Bibliography. `http://www.informatik.uni-trier.de:8000/~ley/db/`.
13. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. *ACM SIGMOD Record*, 3(26), 1997.
14. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 302–314, Edinburgh, UK, 1999.
15. T. Shimura, M. Yoshikawa, and S. Uemura. Storage and Retrieval of XML Documents Using Object-Relational Databases. In *Database and Expert Systems Applications*, pages 206–217. Springer, 1999.
16. Software AG. *Tamino – Technical Description*. Available at `http://www.softwareag.com/tamino/technical/description.htm`.
17. R. van Zwol, P. Apers, and A. Wilschutz. Implementing semi-structured data with MOA. In *Workshop on Query Processing for Semistructured data and Non-Standard Data Formats (in conjunction with ICDT)*, 1999.
18. W3C. Document Object Model (DOM). Available at `http://www.w3.org/DOM/`.
19. W3C. Extensible Markup Language (XML) 1.0. Available at `http://www.w3.org/TR/1998/REC-xml-19980210`.

A Appendix

Proof of Proposition 1. Definition 4 introduces the Monet transform $M_t(d) = (r, \mathbf{R}, \mathbf{A}, \mathbf{T})$ of a document d. For a document d the sets $\mathbf{R}, \mathbf{A}$ and $\mathbf{T}$ are computed as follows:

$$\begin{aligned}
&\textit{for elements:}\\
\mathbf{R} = &\bigcup_{(o_i,o_j,s)\in\tilde{E}} [\mathit{path}(o_i) \xrightarrow{e} s]\langle o_i, o_j\rangle,\\
&\textit{for attributes including CDATA:}\\
\mathbf{A} = &\bigcup_{(o_i,s_1,s_2)\in \mathit{label}_A} [\mathit{path}(o_i) \xrightarrow{a} s_1]\langle o_i, s_2\rangle,\\
&\textit{for ranking integers:}\\
\mathbf{T} = &\bigcup_{(o_i,i)\in \mathit{rank}} [\mathit{path}(o_i) \rightarrow \mathit{rank}]\langle o_i, i\rangle\big),
\end{aligned}$$

where E and label_E are combined into one set

$$\tilde{E} = \{(o_1, o_2, s) | (o_1, o_2) \in E, s = \mathit{label}_E(o_2)\},$$

label_A is interpreted as a set $\subseteq \mathbf{oid} \times \mathbf{string} \times \mathbf{string}$ as well as $\mathit{rank} \subseteq \mathbf{oid} \times \mathbf{int}$, and $[\mathit{expr}]$ means that the value of expr is a relation name. To see that the mapping given in definition 4 is lossless we give the inverse mapping. Given an instance of the Monet XML model $M_t(d)$ we can reconstruct the original rooted tree $d = (V, E, r, \mathit{label}_E, \mathit{label}_A, \mathit{rank})$ in the following way ($\mathit{second\text{-}last}(p)$ returns the second-last component of path p).

1. $V = \big\{o_i | (\exists R \in \mathbf{R})(\exists o_j \in \mathbf{oid}) : R\langle o_i, o_j\rangle\big\}$,
2. $E = \big\{(o_i, o_j) | (\exists R \in \mathbf{R}) : R\langle o_i, o_j\rangle\big\}$,
3. r remains,
4. $\mathit{label}_E = \big\{(o_i, s) | (\exists R \in \mathbf{R})(\exists o_j \in \mathbf{oid})(\exists s \in \mathbf{string}) : R\langle o_i, o_j\rangle \wedge \mathit{second\text{-}last}(R) = s\big\}$,
5. $\mathit{label}_A = \big\{(o_i, s_1, s_2) | (\exists A \in \mathbf{A}) : A\langle o_i, s_2\rangle \wedge \mathit{last}(A) = s_1\big\}$,
6. $\mathit{rank} = \big\{(o, i) | (\exists T \in \mathbf{T}) : T\langle o, i\rangle\big\}$.