Efficient Fragmentation of Large XML Documents

Angela Bonifati[1] and Alfredo Cuzzocrea[2]

[1] ICAR Inst., National Research Council, Italy
bonifati@icar.cnr.it
[2] DEIS Dept., University of Calabria, Italy
cuzzocrea@si.deis.unical.it

Abstract. Fragmentation techniques for XML data are gaining momentum within both distributed and centralized XML query engines and pose novel and unrecognized challenges to the community. Albeit not novel, and clearly inspired by the classical *divide et impera* principle, fragmentation for XML trees has been proved successful in boosting the querying performance, and in cutting down the memory requirements. However, fragmentation considered so far has been driven by semantics, i.e. built around query predicates. In this paper, we propose a novel fragmentation technique that founds on structural constraints of XML documents (size, tree-width, and tree-depth) and on special-purpose structure histograms able to meaningfully summarize XML documents. This allows us to predict bounding intervals of structural properties of output (XML) fragments for efficient query processing of distributed XML data. An experimental evaluation of our study confirms the effectiveness of our fragmentation methodology on some representative XML data sets.

1 Introduction

An imminent development of XML processing is undoubtly making it as fast and efficient as possible. Query engines for XML are being designed and implemented, with the specific goal of employing indexes to improve their performance [10]. Others [23] employ statistics to cost the most frequently asked queries, or use classical algebraic techniques [16] to optimize query plans.

On the other hand, XML query processors suffer from main-memory limitations that prevent them from processing large XML documents. While content-based predicates can be used to project down parts of documents, an XML query engine which is parsimonious in resources, may still enable a further resizing of the obtained projection/query results. This may also happen in many resource-critical contexts, such as a distributed database, or a stream processor. The advantages of XML fragmentation are already being proved in an XML query engine [4,5] or in a distributed setting [3]. Fragmentation of XML documents as proposed by the previous works has been based on semantics, whereas in this paper we work out a novel kind of fragmentation, which is orthogonal to the first and is only guided by the structural properties of an XML document.

Given an XML document, modeled w.l.g. as a tree, there exist several ways of splitting it into subtrees, which may be semantically driven or structurally driven. Usually, query processors decides to apply projections and selections beforehand in order

R. Wagner, N. Revell, and G. Pernul (Eds.): DEXA 2007, LNCS 4653, pp. 539–550, 2007.

to reduce the amount of data to be manipulated in memory during query evaluation. Notwithstanding the effectiveness of pushing algebraic operators within the query plan, it may happen that the size of intermediate results are still too large to fit in memory. If for instance we consider a 100MB XMark document, and a query Q_1 asking for open auctions sold by people owning a credit card (`creditcard` being an optional element) and for closed auctions sold by any people, the result query plan would look like the one shown in Fig. 1 (a). The two branches of the join operator(s) would in such a case be of size 29.6MB(6.1MB) and 17.09MB(11.8MB), respectively. Along with the size, the intermediate results can be also resized w.r.t. tree-width and tree-depth constraints that may affect the query processing time as well. These can be prohibitively large for the join branches above, i.e. 70 (38) and 9 (2) respectively for the left-hand-side join operator. If structure-driven fragmentation is employed, the subtrees output by the selections and projections can be resized to smaller pieces according to the structural constraints and can be then processed per piece. Fig. 1(b) pictures the result of fragmentation on the two operands of the join(s).

A suitable application of structure-driven fragmentation is streaming XML processing (e.g. [8]). Stream query processors are mainly memory-based, thus motivating the use of smaller fragments given for instance by top-down navigation of the original document. Our fragmentation could be employed to obtain smaller XML messages input to the stream, carefully designed to not exceed specified memory requirements at query runtime. Thus, using the three structural constraints altogether allows us to obtain approximately uniform fragments, e.g. to be used in a uniform stream. Finally, distributed and parallel query processing may leverage the fragmentation of the original documents, in order to improve their performance. This issue will be further discussed in our experimental study on *XP2P* [3], a P2P-based infrastructure we have developed.

Coming back to our problem, we can state it as follows. Let t be an XML tree, w a tree-width constraint, d a tree-depth constraint and s a tree-size constraint, we split t into valid fragments f of t such that $size(f) <= s$, $tw(f) <= w$ and $td(f) <= d$ and $\nexists f' \neq f$ such that $f \cap f' \neq \emptyset$, $size$, tw and td being functions returning the size of f, the maximum width of f, and the maximum depth of f, and f' being a valid fragment of t, respectively. Specifically, we consider performance issues of an arbitrary XML processor for what concerns (*i*) the aspect of fragmenting a given XML document, and (*ii*) the aspect of querying the fragmented representation of a given XML document. To this end, we propose an innovative approach for efficiently supporting XML document fragmentation via *structural constraints*, according to which a given XML document is fragmented by imposing "range-shaped" constraints to size, tree-width and tree-depth of output fragments. We name the resulting fragmentation technique as *structure-driven fragmentation of XML documents*.

Although a set of heuristics performing this kind of fragmentation can be easily devised, a key problem is determining the values of structural constraints input to the above heuristics, given that the search space is prohibitive at large. To alleviate the problem, we introduce special-purpose *structure histograms* that report the constraint values for the fragments of a given document. We then present a *prediction algorithm* that probes those histograms to output the expected number of fragments, when fixed input values of the constraints are used. This number is obtained in dependence on

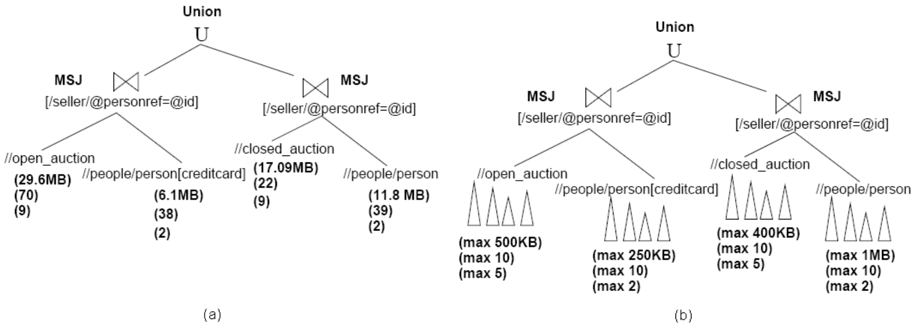

Fig. 1. Query plan of query Q_1 without (a) and with (b) the fragmentation operator applied

structural properties of the input document, thus constituting a value that "summarizes" these properties. Furthermore, we study how to relax the fixed constraints by means of classical distributions. The overall approach we propose is codified within a novel set of heuristics, called *SimpleX*, which, to the best of our knowledge, is the first proposal addressing the XML data fragmentation problem via structural constraints. Finally, we also provide an experimental evaluation of *SimpleX* that clearly shows the effectiveness of our fragmentation methodology in a relevant real-life scenario drawn by a P2P setting and against some representative XML data sets.

The rest of the paper is organized as follows: Section 2 shows the *SimpleX* heuristics for structure-driven fragmentation; Section 3 describes the structure histograms and their use in-support-of the prediction task; Section 4 presents a variety of experiments that probe the effectiveness of our techniques; Section 5 discusses the related work; finally, Section 6 states conclusions and further research.

2 *SimpleX*: Simple Top-Down Heuristics for Shredding an XML Document

The fragmentation problem stated above is a problem with linear cost function and integer constraints, which is intrinsically exponential. To effectively explore the search space, we have designed a set of simple top-down heuristics for document fragmentation, *SimpleX*. They all have in common the fact that they start at the root of the document and proceed in a top-down fashion. At each step the current subtree width, depth and size are checked against the constraints w, d, s. If the constraints are satisfied, the subtree becomes a valid fragment and is pruned from the document to constitute a separate valid XML document. A new node containing as PC-data the path expression of the obtained fragment will then replace the given subtree in the original document. If instead the constraints are not satisfied, the algorithm inspects the next subtree in the XML tree according to the criteria assessed by the heuristic.

A first criterion to select the next subtree is for instance given by the order of visit, i.e. depth-first or breadth-first. We call these variants *in-depth* and *in-width*. Fig. 2

Table 1. Sizes of subtrees of Fig. 2

Node	*Size* (KB)	*Node*	*Size* (KB)	*Node*	*Size* (KB)
site	145	people	100	catgraph	45
person1	20	person2	50	person3	30
edge1	15	edge2	10	edge3	20

represents an XML tree compliant to the XMark DTD, whose subtree sizes [1] are reported in Table 1 as absolute numbers (dots in Fig. 2 represent PC-data elements whose sizes appear in Table 1).

Applying for instance the in-depth heuristics with constrained size $s = 100$ and depth $d = 2$ and unconstrained width w, the XML tree gets fragmented as in Fig. 2 (a), whereas with $s = 100$, $d = 2$ and $w = 1$, it gets fragmented as shown in Fig. 2 (b). The application of the other heuristics on the sample tree is omitted for conciseness.

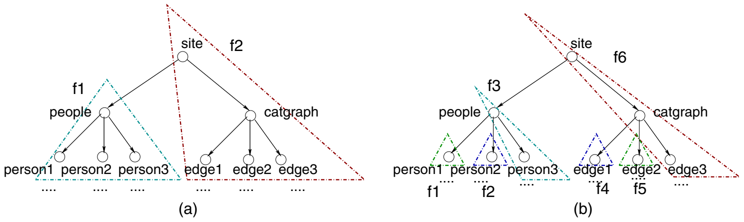

Fig. 2. A sample XMark tree fragmented with one of the *SimpleX* heuristics and two (three) constraints in (a) (in (b))

SimpleX [2] is one possible set of simple heuristics among the various ones that can be applied for shredding a document (e.g. bottom-up or random-access). In principle, there is no better heuristics than any other, as it actually depends on the structure of the document. Our aim in this paper is not finding the best heuristic, but instead to show how to tune the fragmentation constraints for *SimpleX* heuristics, if summary data structures are employed. In fact, note that the constraints of the problem statement introduced in Section 1 may turn to be incompatible if randomly specified, thus possibly leading to empty solutions.

Being the search space prohibitively large, a key problem is determining the values of structural constraints input to the above heuristics. To alleviate the problem, we have designed the structure histograms, which let determine correct combinations of the constraint values, without actually doing the fragmentation beforehand. The histograms have been implemented within an analysis module that uses algorithms to predict an interval for the number of fragments produced by the heuristics. This is particularly

[1] The subtree depth and width can be easily inferred from Fig. 2 and are omitted for space reasons.

[2] In the remainder, and in the experiments, we will simply indicate the set of heuristics as *SimpleX*. We mean that we apply all the heuristics in the set and pick the results of the most efficient one at each run.

interesting for large XML data sets and query results, as it offers a visual summarization tool that can be inspected at any time for prediction. We introduce the structure histograms next.

3 Structure Histograms

Given an XML document X, the structure histograms present X as *summarized* by counting the fragments (i.e., the sub-trees) in X such that these fragments hold the following structural properties: (*i*) the fragment size s, (*ii*) the fragment tree-depth d, and (*iii*) the fragment tree-width w.

Formally, let X be an XML document, let p be a structural property defined on X, let $D_p = [p_{min}, p_{max}]$ be the value domain of p, a *class* Δp is defined on D_p as follows: $\Delta p = [p'_{min}, p'_{max}]$, such that $p_{min} \leq p'_{min} \leq p'_{max} \leq p_{max}$. Then, a structure histogram built on X, denoted by $H_S(X, p, \Delta p)$, grouping p by an aggregation function f = COUNT (thus, reporting the *frequency* of the fragments) over Δp-wise steps, is a tuple $\langle D_p, H_p \rangle$, such that each bucket $b(\Delta p)$ in the co-domain H_p counts the fragments in X having a value of the (structural) property p ranging $\Delta p = [p'_{min}, p'_{max}]$. We call H_S a one-dimensional histogram computed over p. Moreover, to support parametric summarization of XML data and thus improve the fragmentation prediction, we introduce the *parametric structure histogram* $H_S^{\mathcal{P}}(X, p, \Delta p)$, which is an extension of the previous histogram, where $\mathcal{P}$ is a fixed structural property w.r.t. which the histogram over p is computed. Specifically, $H_S^{\mathcal{P}}$ is a two-dimensional histogram computed over $\langle \mathcal{P}, p \rangle$.

In our fragmentation framework, we make use of the following structure histograms summarizing a given XML document X: (*i*) the *Tree-Size Structure Histogram* H_S $(X, s, \Delta s)$, which summarizes X w.r.t. the size s (i.e., $p = s$); (*ii*) the *Tree-Depth Structure Histogram* $H_D(X, d, \Delta d)$, which summarizes X w.r.t. the tree-depth d (i.e., $p = d$); (*iii*) the *Tree-Width Structure Histogram* $H_W(X, w, \Delta w)$, which summarizes X w.r.t. the tree-width w (i.e., $p = w$); (*iv*) the *Max-Tree-Size Parametric Structure Histogram* $H_D^S(X, s, \Delta d)$, which, fixed the size s by computing the max value (i.e., $\mathcal{P} = \texttt{MAX}(s)$), summarizes X w.r.t. the tree-depth d (i.e., $p = d$).

More precisely, given an input XML document X, and a structural property p, we build the output structure histogram $H_S(X, p, \Delta p)$ by setting the input parameters D_p and Δp as follows (it should be noted that the input parameter p is directly set by the target user/application): (*i*) $D_p = [0, MaxValue]$, such that $MaxValue$ is the maximum value of the structural property p among all the fragments in X, (*ii*) $\Delta p = \mathcal{N} \cdot |D_p|$, such that $0 \leq \mathcal{N} \leq 1$ is an empirically set parameter, and $|D_p|$ is the cardinality of D_p. Examples of such structure histograms for the subtree in Fig. 2 are sketched in Table 2. Note that building the histograms for an arbitrary XML tree is necessarily exponential in the worst case, but our heuristics can significantly trim the number of inspected fragments.

A user (or application) willing to partition a document who knows how many fragments he/she wants to obtain, may want to know the values of constraints that let exactly obtain that number of fragments. In other words, he/she would like to properly tune the constraints values. Moreover, constraints as specified by the user may not be compatible among each other or the final results may be biased to the data set

Table 2. H_D, H_S (partial) and H_W for the sample XMark tree of Fig. 2

H_D

D	f
2	1
1	2
0	6

H_S

S	f
145	1
100	1
45	1
20	1
...	...

H_W

W	f
2	1
3	2
0	6

inherent structure. In order to automatize the task of deciding the constraint values, we have devised algorithm *predictInterval* that predicts the range of frequencies by inspecting the structure histograms. The algorithm pseudocode is shown in Fig. 3. For space reasons, we limit ourselves to discuss the algorithm on the XMark sample of Fig. 2 and show that it lets predict the range of frequencies quite sharply. Earlier, we have pointed out that if we disregard the width w in (a), we obtain a rather different fragmentation w.r.t. (b), where w has a non-null value. We start by looking at the histogram H_D reported in Table 2 and we remark that for a value of depth $d = 1$, we would obtain two fragments. If we look at the histogram H_W, this in turn tells that there are two nodes with width $w = 3$, and these nodes cannot be part of the same fragment if w is chosen to be 1. In such a case we would generate 6 fragments out of those nodes. Thus, only by looking at H_D and H_W, we learn that the number of fragments shall be in the range $[2, 6]$. If we further add the third constraint s, the upper bound of the above range may raise or not, depending on whether the fragments so far obtained satisfy or not the value of s. This leads to choose a value of s from histogram H_S, that pessimistically corresponds to the size of the largest subtree located at depth $d = 1$ (e.g. subtrees rooted in nodes `people` and `catgraph` in Fig. 2), information that we learn from an H_D^S histogram. In this particular example, we can choose for instance a size s equal to 100, thus obtaining the fragmentation shown in Fig. 2 (b) quite straightforwardly.

As we have seen, choosing correct values for the input constraints of the fragmentation algorithm is a non trivial task. An incorrect value for such constraints would lead to too many fragments or too few of them, or even to an empty solution in some cases. Indeed, there may exist random values of w, d and s, which turn out to be incompatible among each other. In order to alleviate this problem, we let the constraints vary along classical distributions (such as Uniform, Gauss, Zipf), thus relaxing constraints with such distributions. Thus, along with choosing fixed bounds for s, w, d, we assign ranges to them according to those distributions. This is further motivated by the fact that an XML document contains “unbreakable” pieces of text (such as PC-data, entities etc.) that needs to be taken into account in the choice of the constraint values (especially for the size constraint). By empirically comparing the output of *SimpleX* against the baseline case given by a constant distribution, we will show below that non-constant distributions have in general a better behavior.

```
algorithm $predictInterval(H_D$: tree-depth structure histogram,
    $H_S$: tree-size structure histogram,
    $H_W$: tree-width structure histogram,
    $s_0, d_0, w_0$: size, depth, width constraints): return $[f_{min}, f_{max}]$
  Let $d_0$ be the chosen depth in $H_D$ // alternatively, $w_0$ in $H_W$
    such that $H_D(d_0, \Delta d_0, f_0)$
  Pick the max width $w_{max}$ in $H_W$ //alternatively, $d_{max}$ in $H_D$
  Let $f_{min}$ = $f_0$, $f_{max}$ = sum($f_0$,$w_{max} - w_0$)
  Let $s_{max}$ the max size at depth $d_0$ in $H_D^S$
  Let $f_{s_{max}}$ the corresponding frequency in $H_S$
    If $s_0 >= s_{max}$
      return $[f_{min}, f_{max}]$
    else {
      $f_{max}$ += $f_{s_{max}}$
      for each $w_i$ in the interval $w_{max} - w_0$ in $H_W$
        $f_{max}$ += $f_{w_i} * (w_i - w_0)$
    }
  return $[f_{min}, f_{max}]$
```

Fig. 3. Algorithm *predictInterval*

4 Experimental Assessment

We have conducted an experimental study aimed at showing the effectiveness of our structure-driven fragmentation methodology. The experiments are divided into three classes. First, we build the structure histograms for representative XML data sets, and show their use to decide the final values of constraints. Secondly, we define and measure the accuracy error of fragmentation using the *SimpleX* set of heuristics, when fixed constraints are employed against the cases (baseline) in which the constraints vary with classical distributions (e.g. Uniform, Gauss, Zipf). Finally, we demonstrate the utility of fragmentation in a distributed setting, such as a DHT-based P2P network. In such a case, we measure the impact of fragmentation on network performance against the expected ideal behavior.

All the experiments have been performed on a machine with a 1.2 GHz processor, 512 MB RAM, and running Linux Suse 9.1. We uniquely identify each fragment with its absolute root-to-leaf path expression. Each fragment stores with extra `sub` nodes the path expression of subfragments and separately the path expression of its parent fragment. We have presented this data model in [3]. Notice that any data model (such as [5] for instance) other than ours can be adopted here to represent the fragments. Finally, the data sets and queries employed in the study are summarized in Table 3.

Fig. 4 shows the structure histograms for the Nasa data set. In order to improve readability, we separately report the complete histograms in Fig. 4 and some of the frequency values in Table 4. Note that such histograms have a size of the order of KB, thus being reasonably small. For instance, considering a triple d_0 = 5, $w_0 = 1200$ and $s_0 = 230KB$, and applying the Algorithm in Fig. 3, we obtained a prediction range equal to $[1200, 2500]$. Similar results for the other data sets of Table 3 and other values of the constraints are omitted for space reasons. Moreover, notice that the histograms

Table 3. XML documents and queries used

Document d (MB)	# *elems.*	*maxDepth*	*maxWidth*	*provenance*
XMARK (113)	3,332,129	11	25,500	[20]
XMARK (30)	501,705	11	7649	[20]
NASA (24)	476,645	7	2434	[19]
FourReligiousWorks-Bom (1.5)	7,656	5	79	[9]

Query	*Description*
QD_i	`FOR $p IN XPathExpr RETURN $p`

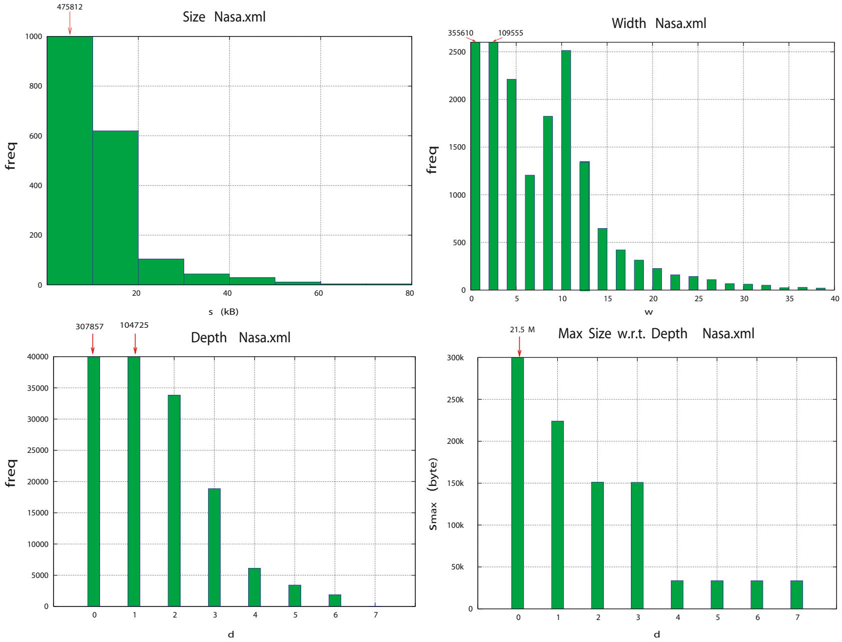

Fig. 4. Tree-Size (H_S), Tree-Width (H_W), Tree-Depth (H_D) and, finally, Max-Tree-Size Parametric (H^S_D) structure histograms for the Nasa data set

also allow a user to quickly discard incompatible values of constraints as they summarize only valid constraints values.

We have put at work the proposed heuristics on the data sets of Table 3. We have considered various values of parameters s, w and d and run the heuristics with fixed values of these parameters and with their variations as given by classical distributions (e.g. Uniform, Gauss, Zipf). We define the accuracy error e_c of fragmentation w.r.t. constraint c as follows. Let c_a be the average value of the size (tree-depth and tree-width, resp.) obtained with *SimpleX* heuristics, c_0 the fixed value of constraint input to algorithm *predictInterval* (see Fig. 3), c_{min} and c_{max} the interval of the constraint value as obtained via the particular distribution, then the accuracy error of fragmentation is

Table 4. Window frames of histograms in Figure 4 depicting some of the frequency values

$[s_1 - s_2]$ (KB)	$freq$
[50 - 100]	20
[100 - 150]	8
[150 - 200]	6
[200 - 250]	1

w	$freq$
674	1
730	1
1188	1
2434	1

d	$freq$
4	6100
5	3430
6	1854
7	1

Table 5. Application of *SimpleX* to NASA with/without distribution

$Distribution$	$\# \, fragments$	$Avg. \# \, nodes$	e_s	e_d	e_w
None	1879	253	0.97	0.6	0.97
Uniform	61	7813	0.02	0.57	0.89
Gauss	57	8362	0.09	0.42	0.88
Zipf	71	6713	0.1	0.57	0.9

given by the formula $\frac{|c_0-c_a|}{c_0}$ for fixed constraints, and by the formula $\frac{|\frac{c_{min}+c_{max}}{2}-c_a|}{|\frac{c_{min}+c_{max}}{2}|}$ for non-fixed constraints.

As an example, Table 5 shows the obtained results with the NASA data set and value of constraints: s = 230KB, w = 1200 and d = 5. The lower is the accuracy error, the better is the matching of the heuristics with the fragmentation constraints. It can be noticed that the case when the constraints are strict upper bounds leads to fairly more fragments than the cases when distributions are applied. On average, fragmentation via distributions obtains lower accuracy errors than the case when distributions are not used. Results with other data sets and other values of constraints showed the same trend.

As we already discussed, there exist several applications of our fragmentation strategy, which justify its effectiveness. Here, we present some experiments that have been performed on *XP2P*, our DHT-based P2P simulator [3]. For each experiment, we have scattered a certain number of fragments in the network obtained with our structure-driven fragmentation. We then measured the network scalability when both varying the number of peers and the number of queries.

Fig. 5 (a) shows the nr. of hops versus the number of peers when XMark(30) has been divided into 1000 fragments with the constraint values predicted by our analysis tool. Here we have considered exactly as many queries of kind QD_i (see Table 3) as the number of fragments, each query being propagated to the successor peers as dictated by the current peer list of successors (i.e. at logarithmic distance). It can be noticed that the case where XML structure-driven fragmentation is used closely tracks the original Chord [3] logarithmic curve. Finally, Fig. 5 (b) shows the nr. of hops when varying the nr. of fragments for XMark(30) data set within a network of 500 peers. The fragmentation slightly increases the number of hops, if compared with the constant curve that represents no fragmentation.

[3] The original Chord simulator only stores on each peer an identifier of resources. In *XP2P*, we have extended it to store XML fragments.

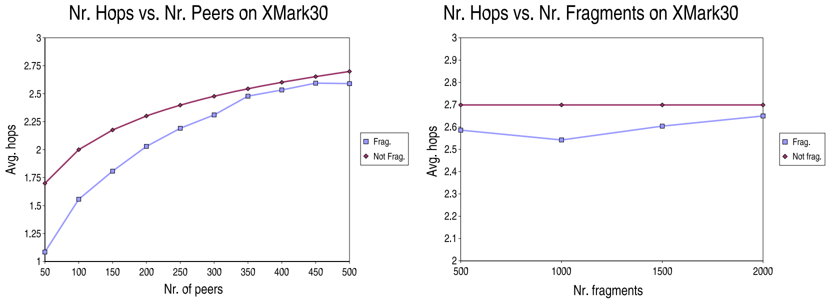

Fig. 5. Nr. of hops w.r.t. nr. of peers with 1000 fragments (a); Nr. of hops w.r.t. nr. of fragments with 500 peers (b). In both cases, the number of queries of kind QD_i equals the number of fragments.

5 Related Work

The advantages of fragmenting relational databases are well established [17] as both horizontal fragmentation [7], which splits a given relation into disjoint sub-relations, and vertical fragmentation [13], which projects a given relation onto a subset of its attributes. More recently, fragmentation techniques have been adapted to object-oriented [1], semi-structured [14], and native XML [4] databases.

[15] proposes an innovative approach for supporting the distribution of XML databases via horizontal fragmentation. It employs a query-oriented cost model taking into account the most frequently asked queries, and uses heuristics to optimize the fragmentation based on the efficiency of such queries. Being query-driven, this approach does not consider the structural properties of XML data, as we do in our proposal. *XFrag* [4] is a framework for processing XQuery-formatted queries on XML fragments in order to reduce memory processing. This work focuses on how to employ fragments to make query optimizations, thus strengthening our proposal. Several query optimization techniques have been presented for XML data, among which [16] and [11]. While the former relies on algebraic projections, the latter is based on tree-automata. These techniques can be combined with ours to let the processor evaluate queries in parallel on multiple fragments.

[6] and [18] propose summary data structures for XML twigs and paths that let derive an estimation of queries selectivity by using statistical methods, such as histograms or wavelets. Notwithstanding the importance of the above data structures for query optimization, our histograms are instead aimed at predicting the number of fragments of an XML document when applying *SimpleX* heuristics. The latter prediction could not be inferred by looking at the above data structures. [22] proposes the *position histograms*, which allow the estimation of both simple and complex pattern query answers. Furthermore, when XML schema information is available, they employ the so-called *coverage histograms* that extend the former and allow the target XML database to be better summarized. Differently from ours, those histograms help estimating the sizes of child and descendant steps in path expressions. Histograms are used for a rather different purpose in our framework, as stated above.

Finally, *XRel* [21] is a path-based approach to store XML documents into RDBMS and retrieve them afterwards. While the path identification of fragments is similar to ours, the focus of the paper is on building an extension of relational databases for XML data.

6 Conclusions and Future Work

We have presented a fragmentation strategy for XML documents that is driven by structural constraints. To the best of our knowledge, this is the first work addressing such a problem. We further offer the user or the application a prediction of the "outcome" of the fragmentation, by means of the so-called structure histograms. By means of distributions, we are able to vary the constraint values thus improving the fragmentation performance.

We are currently developing new classes of heuristics. The first one uses additional data structures in combination with histograms in order to make the prediction more precise. The other one considers schemas of XML documents (when available) during the prediction. Moreover, another research direction we are considering consists in providing full support to *XML join queries* via devising ad-hoc heuristics that focus on the fragment size, which is a critical parameter affecting the computational cost due to evaluating such queries. Finally, we plan to embed our fragmentation tool and its analysis module in an existing XQuery engine.

References

1. Bellatreche, L., Karlapalem, K., Simonet, A.: Algorithms and Support for Horizontal Class Partitioning in Object-Oriented Databases. Distributed and Parallel Databases 8 (2000)
2. Bohannon, P., Freire, J., Roy, P., Simeon, J.: From XML Schema to Relations: A Cost-based Approach to XML Storage. In: Proc. of ICDE (2002)
3. Bonifati, A., Cuzzocrea, A.: Storing and Retrieving XPath Fragments in Structured P2P Networks. Data & Knowledge Engineeering 59 (2006)
4. Bose, S., Fegaras, L.: XFrag: A Query Processing Framework for Fragmented XML Data. In: Proc. of WebDB (2005)
5. Bremer, J.M., Gertz, M.: On distributing xml repositories. In: Proc. of WebDB (2003)
6. Chen, Z., Jagadish, H.V., Korn, F., Koudas, N., Muthukrishnan, S., Ng Raymond, T., Srivastava, D.: Counting Twig Matches in a Tree. In: Proc. of ICDE (2001)
7. Ezeife, C., Barker, K.: A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object based System. Distributed and Parallel Databases 3 (1995)
8. Florescu, D., Hillery, C., Kossman, D., et al.: The BEA/XQRL Streaming XQuery Processor. In: Proc. of VLDB (2003)
9. Ibiblio.org web site (2004), Available at `http://www.ibiblio.org/xml/books/biblegold/examples/baseball/`
10. Jagadish, H.V., Al-Khalifa, S., Chapman, A., Lakshmanan, L.V., Nierman, A., Paparizos, S., Patel, J., Srivastava, D., Wiwatwattana, N., Wu, Y., Yu., C.: Timber: a Native XML Database. VLDB Journal 11 (2002)
11. Koch, C.: Efficient Processing of Expressive Node-Selecting Queries on XML Data in Secondary Storage: A Tree Automata-based Approach. In: Proc. of VLDB (2003)

12. Krishnamurthy, R., Chakaravarthy, V.T., Naughton, J.F.: On the Difficulty of Finding Optimal Relational Decompositions for XML Workloads: A Complexity Theoretic Perspective. In: Proc. of ICDT (2003)
13. Lin, X., Orlowska, M., Zhang, Y.: A Graph-based Cluster Approach for Vertical Partitioning in Databases Systems. Data & Knowledge Engineeering, 11 (1993)
14. Ma, H., Schewe, K.D.: Fragmentation of XML Documents. In: Proc. of SBBD (2003)
15. Ma, H., Schewe, K.D.: Heuristic Horizontal XML Fragmentation. In: Proc. of CAiSE (2005)
16. Marian, A., Simeon, J.: Projecting XML Documents. In: Proc. of VLDB (2003)
17. Ozsu, M., Valduriez, P.: Principles of Distributed Database Systems. Alan Apt (1999)
18. Polyzotis, N., Garofalakis, M.N.: Statistical synopses for graph-structured XML databases. In: Proc. of SIGMOD (2002)
19. University of Washington's XML repository (2004),
Available at http://www.cs.washington.edu/research/xml/datasets
20. Xmark: An XML Benchmark Project (2002),
Available at http://monetdb.cwi.nl/xml/
21. Yoshikawa, M., Amagasa, T., Shimura, T., Uemura, S.: XRel: A Path-based Approach to Storage and Retrieval of XML Documents Using Relational Databases. ACM Transactions on Internet Technology 1 (2001)
22. Wu, Y., Patel, J., Jagadish, H.: Using Histograms to Estimate Answer Sizes for XML Queries. Information Systems 28 (2003)
23. Zhang, N., Haas, P., Josifovski, V., Lohman, G., Zhang, C.: Statistical Learning Techniques for Costing XML Queries. In: Proc. of VLDB (2005)