

Distributing XML with Focus on Parallel Evaluation

Marcel Waldvogel, Marc Kramis, and Sebastian Graf

University of Konstanz, Distributed Systems Laboratory,
78457 Konstanz, Germany

Abstract. In contrast to relational databases, the distribution of semi-structured document-centric XML data is not well researched. While there are some suggestions on how to split and distribute large XML documents, these approaches do not consider the parallel query evaluation. In this paper, we present and compare five different algorithms to split a large XML document into a fixed number of XML fragments. We then describe how to distribute the resulting XML fragments over a set of peers and how to query these peers in parallel to retrieve the final result. In addition, we analyse the impact of our splitting algorithms with respect to scalability for two different XPath expression classes on the well-known XMark dataset. We conclude this paper with an outlook on future work, including result ordering during parallel query execution and dynamic re-distribution of XML fragments to new peers due to updates.

1 Introduction

Despite its reputation as being too verbose as well as slow and inefficient with respect to processing, XML established itself as a first-class citizen throughout the modern computer world. As it expands and is adopted for a growing number of document formats, people *do* actually value features such as self-descriptiveness and data-before-schema as well as XML's rich toolset and universal interchangeability including long-term archival.

Initially, XML was just used to exchange or store small amounts of data in a unified way. Nowadays, even large data- and document-centric data sets are stored in the XML format within simple XML files or (native) XML databases. However, a single system facing such enormous amounts of XML data quickly comes to its limits due to insufficient amounts of storage space, processing power, or available main memory. When it comes to massive concurrent access or to single-system failures, distribution to multiple systems becomes the only feasible option.

The distribution of data-centric XML is straightforward and extensively researched with relational databases where the aim is to distribute columns and rows in a reasonable way. Here, reasonable means with minimal effort and optimally suited to the workload to which the database is exposed. In stark contrast,

the distribution of document-centric XML is much more difficult as the straightforward approaches for distributing the XML fragments will likely not result in a balanced system. E.g., if a splitting algorithm splits the XML tree at the first level, i.e., distributes all children of the root node, it may happen, that some peers will store a large number of XML fragments containing one more child while other peers will have to store large and deep sub-trees.

Automatically finding the appropriate split nodes in the tree is a challenging task as the system dynamically has to adapt to every single XML document and therein to a quickly changing topology when moving from one to the next sub-tree. Given a fixed number of peers available to store XML fragments the question arises how to split the tree to make sure every single peer gets its fair share of XML fragments of comparable size. In other words, the question is whether there exists a split algorithm producing an optimal distribution for any document-centric XML data. Only when the XML fragments are evenly distributed over multiple peers, each one has an equal chance of being involved in a parallel query evaluation. Still, chances are that some peers get more involved due to a specific query workload but if the distribution is already skewed due to a bad split algorithm, even average queries will contribute to some peers becoming a bottleneck.

During our quest for the optimal split algorithm, we evolved five different approaches and analysed each one on the well-known XMark benchmark. The analysis is based on both a measure which gives an idea of the alluded fair share for each peer as well as an evaluation of two different XPath expression classes. Extensive benchmarks were used to verify our findings and select the optimal split algorithm.

The rest of this paper is organised as follows. Section 2 describes the related work in detail. Section 3 contains our main contribution, i.e., the five split algorithms. Section 4 shows how we distributed the XML fragments to different peers. Section 5 describes how we evaluate queries in parallel. Section 6 provides our findings in the form of benchmark results. Finally, Section 7 concludes this paper and gives an extensive outlook on future work.

2 Related Work

Research started to look into distributed semi-structured XML data only a few years ago. Many approaches considering distributed queries are based on the assumption that the XML is already distributed [18, 4, 6, 1, 5]. Based on the well-known distribution techniques of relational databases, i.e., the horizontal [7] and vertical fragmentation [11], some take this straightforward concept of fragmentation into account [9, 12, 14, 15]. The suggested algorithms work well for data-oriented XML because of their regular structure.

However, these approaches are not suitable with respect to parallel evaluation of document-centric XML. The resulting XML fragments have different structural characteristics due to their inherent irregularities. To our knowledge, there are only a few approaches, which take the structure of a XML into account when partitioning and distributing it. [3] presented an approach which is

directly based on the structure, i.e., the given XML is partitioned based on several structural constraints. These constraints are defined by the width, the size, and the depth of sub-trees which can be extracted. In addition, the parameters have to be manually set before-hand to obtain a fragmentation. Depending on the parameters, a good fragmentation with respect to a parallel evaluation is possible.

A completely different approach with the same focus on parallel queries is described in [13]. The parallel evaluation takes place either on a distributed XML which was partitioned with the help of graph-partitioning algorithms [10] or on a variable fragmentation depending on the executed query. The variable fragmentation is based on the operations performed by the issued query. In this case, the fragments are represented by DOMs. This usage reduces the usability of the variable fragmentation because the DOMs have to be adapted each time the query changes.

3 Five Split Algorithms

Our approach works in two independent steps: First, suitable nodes to split are identified. These nodes (and their respective sub-trees) are extracted from the original tree structure as described with [3] and in a second step combined to get a desired number fragments.

In this chapter, five different split algorithms are presented. Each split algorithm aims at finding the optimal split-nodes, especially for document-centric data with respect to a parallel evaluation of the resulting sub-trees. To evaluate and compare the presented approaches two main objectives are set:

- A parallel evaluation is supported by a tree-walking query if the bulk of split-nodes have non-unique tag names. Every sub-tree of each split-node with the same tag name can be evaluated in parallel.
- The sub-structures should be as big as possible considering the desired similarity of the tag names in the set of split-nodes. This distribution generates an overhead on the queries itself. Regarding parallel evaluation, the handling of this overhead can only be justified by the evaluation of large structures.

3.1 Level Split

This approach marks all nodes on a given level as split-nodes. The algorithm is described in algorithm 1. An example is shown in Figure 1(a). This approach works perfectly for data-oriented XML where the structure is quite similar as described with current approaches based on horizontal fragmentation techniques inherited from the distribution of relational data. In contrast, it is not guaranteed to get a good fragmentation result with this approach when it comes to document-centric data due to the irregularity of the document structure. As shown in Figure 1(a) all relevant sub-trees with the corresponding split-nodes a are identified. So the condition of split-nodes with equal tag names is satisfied. The size of the identified sub-structures are quite small regarding the other

```

Data: level  $l$  where the designated split-nodes are located on
Result: split-nodes  $s$ 
foreach node  $n$  in pre order do
  | if  $n$  is on level  $l$  then
  | | insert  $n$  in  $s$ 
  | end
end

```

Algorithm 1: Level Split Algorithm

possible split with the split-nodes a, a, b, a, c where the extracted sub-structures would be much larger. In this case the structure with the split-node denoted with c would be extracted as well. This is not wanted because this tag name only occurs once in the set of split-nodes. Consequently, this algorithm is not sufficient for partitioning document-centric XML.

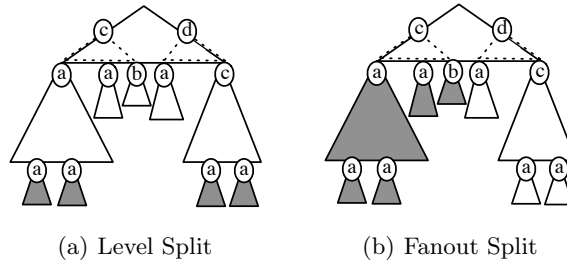


Fig. 1.

3.2 Fanout Split

To get only large sub-structures, the assumption is, that a node with a large sub-tree also has a large number of children as well. Therefore, according to a given number of children, a node in the XML structure is marked as a split-node if this number reaches a given threshold. The corresponding algorithm is described in algorithm 2. This threshold must be set before the identification process and can be determined with a suitable exploration [8] for example. An example of the result with the split algorithm is shown in Figure 1(b). Again, this approach works well for data-centric XML. For document-centric structures, the identification of the split-nodes should not only be based on the fanout of a node due to the irregularity of the document structure. Here, this results in nodes with a large fanout but relatively small sub-trees. Additional to this possible violation of the above heuristic, there is no assertion that there are no unique tag names in the set of split-nodes. In Figure 1(b), e.g., is the split-node named b set, but

Data: threshold t of needed number of children
Result: split-nodes s
foreach *node* n *in pre order* **do**
 $c \leftarrow$ number of children;
 if $c \geq t$ *AND* $n.ancestors \notin s$ **then**
 insert n in s ;
 end
end

Algorithm 2: Fanout Split Algorithm

a parallel evaluation of the underlying structure is not possible because of the document-centric structure.

3.3 Semantic Split

To consider similar tag names of split-nodes, this approach is working on the occurrence of different tag names on the sibling axis. The algorithm is showed in algorithm 3. On each level, the occurrence of different tag names on the

Result: split-nodes s
foreach *node* n *in pre order* **do**
 if n *has no left sibling AND* n *has a right sibling* **then**
 $t \leftarrow$ compute relative tag name occurrence on sibling axis;
 if $t == 1$ *AND* $n.ancestors \notin s$ **then**
 insert n in s ;
 end
 end
end

Algorithm 3: Semantic Split Algorithm

sibling axis is explored. If there is more than one node in the sibling axis and each node has the same tag name, these nodes are identified as split-nodes. An example is given in Figure 2(a). Again, the grey sub-structures can be queried in parallel. Unfortunately, as with the Level Split, the identified sub-structures are quite small and therefore not optimal. The partitioning of sub-structures as labeled with a make more sense with respect to a parallel evaluation. Still, another approach with respect to the size of the sub-structures is needed.

3.4 Postorder Split

To tackle the need for large extractable sub-structures, we developed a novel split algorithm. By means of a designated number of fragments, two thresholds are defined. With the help of a post order traversal through the original tree structure, the sub-tree size to each node is computed. If one of the thresholds

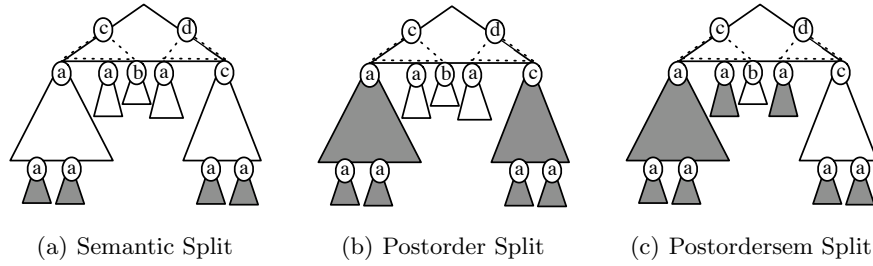


Fig. 2.

is reached, the current node is marked as a splitnode. The two thresholds are represented by the *delegateThreshold*- and the *splitThreshold*-variable in algorithm 4. The normal threshold to mark a node as a splitnode is represented by the *delegateThreshold*. If none of the traversed sub-structures could reach the *delegateThreshold*, the *splitThreshold* inhibits the extraction of a very large sub-structure according to the father node of these sub-structures. If no node has the potential to work as a split-node, the children on the first level are selected to achieve at least a basic fragmentation. This behaviour is given with the check for the root node. Figure 2(b) shows an example. The biggest possible sub-structures according to a given threshold are identified. The identified sub-structures have unique tag names. This complicates parallel evaluations on this data, even if the corresponding sub-structures contain a large amount of nodes. This split-node obviously generates sub-structures with a similar load. Regarding data-centric data, this approach also works well because the identified split-nodes have similar tag names. With document-centric XML, this split algorithm can result in split-nodes with different tag names.

3.5 PostorderSem Split

To get rid of the possibly different tag names of the Postorder Split, this approach combines the post order traversal with the node count as well as the Semantic Split. The tree structure is traversed in a post order traversal similar to the Postorder Split. Instead of computing the size of some sub-tree according to a given node, the size of the sub-trees according to a given tag name is computed. A detailed description is given with algorithm 5. With this approach, the two main objectives are satisfied. First, the extracted sub-trees have a suitably large size. Because the computation of split-nodes takes place on the number of traversed nodes according to the tag name, the sub-structures themselves have to be quite big. In addition to the size, the second objective, i.e., the similarity of the split-nodes according to their tag name, is given. This algorithm extracts only relevant sub-structures which guarantee the parallel evaluation.

Data: number of available peers p , $p \geq 2$
Data: number of nodes in XML m
Result: split-nodes s
 $delegateThreshold = m/(p * 2)$;
 $splitThreshold = (m/p) * 2$;
foreach node n in post order **do**
 $s_n \leftarrow$ size of sub-tree of n only with respect to not extracted nodes;
 if $splitThreshold \leq s_n$ **then**
 | insert $n.children$ in s ;
 end
 else
 if $delegateThreshold \leq s_n$ **then**
 if n is root **then**
 | insert $n.children$ in s ;
 end
 else
 | insert n in s ;
 end
 end
 end
end

Algorithm 4: Postorder Split Algorithm

4 Distribution and Combination of Relevant Substructures

After the identification of suitable split-nodes in context to a parallel evaluation of the corresponding sub-structures, these sub-structures have to be extracted from the original structure. The distribution of the identified structures takes place *round-robin* alike. Algorithm 6 shows the fragmentation. A suitable number of wanted fragments is set. Additional to these structures, a root fragment is initialized. Afterwards, the original XML is pre order traversed. Each node is inserted in the root fragment until the first split-node is reached. If this occurs, a proxy-node with a unique id is inserted on the root fragment and a children fragment is selected in *round-robin* order. On the children fragment, a corresponding proxy-node is inserted which has also an unique id. With this ids a direct access between the proxy-node in the root fragment and the proxy-node in the children fragment is supported. After inserting this proxy-node, the split-node itself is inserted under this proxy-node. Afterwards, each following node is inserted on this children fragment until the sub-structure, which was identified to be extracted, is left. Then, the following nodes are inserted on the root fragment until the next split-node is reached and so on.

With this approach, even document-centric data is, depending on suitable split-nodes, very well fragmented. Because similar tags are following each other, these similar tags and their sub-structure is distributed in a way, that support parallel query evaluation.

Data: number of available peers p , $p \geq 2$
Data: number of nodes in XML m
Result: split-nodes s
 $delegateThreshold = m / (p * 2)$;
 $namesSubtreeSizesHashtable = name \rightarrow \#nodes$ **foreach** node n in post order **do**
 $s_n \leftarrow$ size of sub-tree of n only with respect to not extracted nodes;
 if n has no leftsibling **then**
 foreach node o in sibling axis **do**
 if $tagname_o \notin namesSubtreeSizesHashtable$ **then**
 | insert $tagname_o$ in $namesSubtreeSizesHashtable$;
 end
 end
 end
 adapt # nodes in $namesSubtreeSizesHashtable[tagname_n]$;
end
foreach node n in level order **do**
 $occur_n \leftarrow namesSubtreeSizesHashtable[tagname_n]$ **if**
 $splitThreshold \leq occur_n$ **then**
 | insert n in s ;
 end
end

Algorithm 5: PostorderSem Split Algorithm

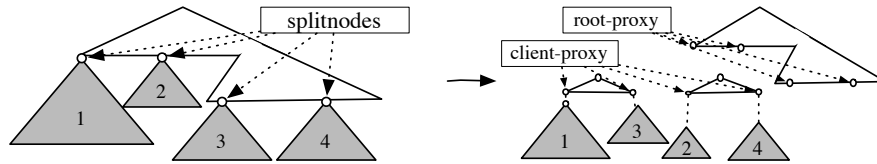


Fig. 3. proxyLayout

Figure 4 shows a fragmented XML. The following sub-structures 1,2 and 3,4 are, with 2 desired fragments, distributed. Results on common benchmarks show that this distribution approach leads to good practical result even if there is the possibility that a parallel query is fully not supported with this fragmentation regarding document-centric data.

5 Query distributed XML Documents

XPath 2.0 [2] is a common way to select parts of an XML document. It is used as a base for many common query and transformation languages such as XQuery and XSLT. The XPath 2.0 expressions are split into axis steps, which navigate the tree in a way and perform node-tests to filter the required nodes.

With respect to a given fragmentation, expression evaluation can be used in a straightforward way. First of all, we have to modify the expression to only use

Data: number of designated fragments n
Result: Set of children-fragments c
Result: Root-fragment r
Initialize s ;
 $i = 0$;
 $currentFragment \leftarrow r$;
foreach node n in pre order **do**
 if n is a splitnode **then**
 $currentFragment \leftarrow r$
 $currentFragment.appendInPreorder(proxyOnRoot)$;
 $currentFragment \leftarrow s[i\%n]$
 $currentFragment.appendInPreorder(proxyOnClient)$;
 $i = i + 1$
 end
 if $n.ancestors$ is not a splitnode **then**
 $currentFragment \leftarrow r$;
 end
 $currentFragment.appendInPreorder(n)$;
end

Algorithm 6: Distribute XML

the forward axes. Because of the symmetries in XPath [16], this modification is not a restriction. We did not implement the *following* axis with our prototype because this kind of axis step is not commonly used in practical environments. Furthermore, querying the *following* axis cannot be used just like that because of the distribution scattering the nodes of interest throughout multiple peers.

A given XPath expression is executed on the root-fragment. The expression traverses the tree in the designated order and is, after reaching a proxy node on the root-fragment, handed over to the appropriate fragment. From this point on, the expression is continuing its work on the root fragment without waiting for the result. The expression on the child fragments is executed at the same time in parallel. Care has to be taken when the expression is handed over to the child fragment. To match the sub-tree of the fragment, the expression might have to be adapted, e.g., a query containing just children-steps is pruned to perform only those children-steps, which are executable on the subtree. After the evaluation of the root fragment, the query processor waits for the result from the fragments and the result, which has to be combined, is returned to the querying system.

6 Benchmark Results

The described approach was evaluated in detail. Based on an environment of 4 servers each with 8GB RAM and two DualCore Opteron Processors each with 2Ghz, an evaluation on 1 until 12 peers has taken place because one server has to work as the root-peer handling the root-fragment.

The bench takes place on a XML from the XMark-project [17] with the size of 1GB. Regarding two queries of the XMark-project, the XPath-part was extracted, which is not in need of joining the data.

- /site/closed_auctions/closed_auction/annotation/description/parlist/listitem/parlist/listitem/text/emph/1
- count(/site//item)

A detailed description of these queries is given in [17]. The queries are executed five times and the average and the corresponding 95%-confidence intervals are computed and plotted. Because the fragmentation of PostorderSem, Semantic and Fanout Split are the same in this example, the results are combined in one chart.

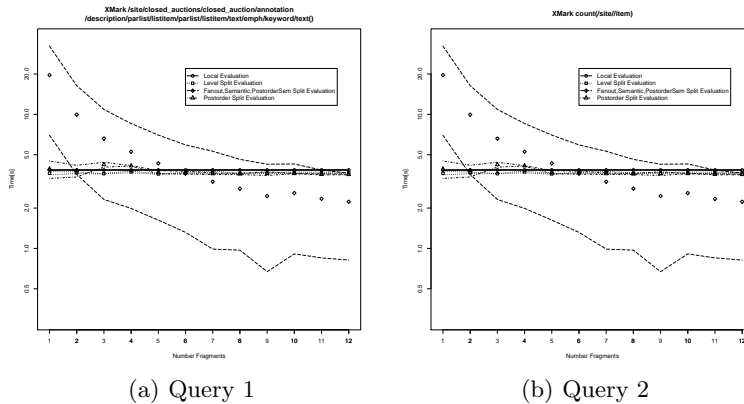


Fig. 4.

Figure 4(a) shows the result on a the first query. The Level Split is not benefitting from the distribution as well as the Postorder Split because the data which has to be evaluated is located on one children fragment. The other split operators are able to query the data distributed. Therefore the evaluation takes place in parallel.

Figure 4(b) shows another kind of query. The Level Split is scaling quite irregular. This behaviour is based on the structure of XMark. This leads to an unequal distribution of the *item*-elements because the father-nodes are containing a different load of *item*-nodes. The Postorder Split is scaling much better corresponding to the last query. Fanout, Semantic and PostorderSem Split are given the best evaluation time because their fragmentation is very consistent.

Other famous XML-databases as Treebank, DBLP, Wikipedia and Swissprot are benched with similar results as well.

7 Summary and Outlook

In this paper, we present five different algorithms to split a large XML document. We show how to distribute the resulting XML fragments to a fixed number of peers, and how to query these peers in parallel. To verify our approach, we evaluated two different XPath 2.0 expression classes on the well-known XMark

dataset. Our findings affirm the assumption that a trivial split algorithm does not guarantee an optimal distribution, which leads to a scalable parallel query evaluation. More sophisticated split algorithms such as the semantic split, the postorder split, or the postorderSem split invariably allow for much better scalability. At the same time, these algorithms split a large XML document with a single traversal as used with the trivial split algorithm and do not incur a significant split overhead.

We see many open questions for future work in the area of distributing large-scale XML data for parallel query evaluation:

- Depending on the query, a simple parallel evaluation may lead in the need to reorder the result retrieved from multiple peers. We want to investigate which queries are affected and whether this reordering can be prevented or efficiently done during either the split operation or the parallel evaluation.
- While XPath 2.0 provides a fundamental idea of how the evaluation time will be, it is only a sub-set of current query languages such as XQuery 1.0. We want to look at XQuery 1.0 and how it can be executed in parallel by rewriting the query itself or by optimising and splitting the logical operator tree.
- Currently, we split an existing XML document, which does not change. We are interested in updates and how they lead to re-assignments of XML fragments to keep the whole distribution in balance. This may be achieved through moving the fragments themselves or by dynamically further splitting the XML fragments.
- The availability of indices may lead to faster evaluations for certain queries, i.e., an index can answer the query in almost fixed time without the need to do a full XML fragment traversal. However, an index will incur more update overhead and may itself grow so large it needs to be distributed. E.g., a full text index which has to store a term occurring in a large percentage of the nodes but is not blocked in a stop list is no longer useful. Through splitting the XML document in smaller parts, we also make sure to split the domain of each index and potentially reduce the over-all update and search time.
- The reliability and availability of large XML documents will also become an issue. E.g., it is no longer possible to export Wikipedia to an XML file within a single day. Losing the whole file due to a peer failure is catastrophic and would interrupt a service relying on it for too long. As soon as Wikipedia is distributed, the loss of a single peer will only erase a small part of the overall document. The question then becomes how to store a single XML fragments on multiple peers and how to exploit this knowledge to further speed up the query evaluation when each peer has its individual performance and variable network connection quality.

With our five split algorithms, we break new ground on how to distribute large-scale document-centric XML data and how to query it in parallel for scalability reasons. However, much work remains before a cluster of peers will automatically and collaboratively store and query such large-scale XML documents.

References

1. Abiteboul, S. and Bonifati, A. and Cobéna, G. and Manolescu, I. and Milo, T.: Dynamic XML documents with distribution and replication Proceedings of the 2003 ACM SIGMOD international conference on Management of data 2003
2. Berglund, A. and Boag, S. and Chamberlin, D. and Fernandez, M.F. and Kay, M. and Robie, J. and Simeon, J.: XML Path Language (XPath) 2.0 W3C Working Draft 15 2002
3. Bonifati, A. and Cuzzocrea, A.: Efficient Fragmentation of Large XML Documents. Lecture Notes in Computer Science 4653 2007
4. Bose, S. and Fegaras, L.: XFrag: A Query Processing Framework for Fragmented XML Data. In Proceeding of WebDB 2005
5. Bremer, J.M. and Gertz, M.: On Distributing XML Repositories Proceedings of the WebDB 2003
6. Buneman, P. and Cong, G. and Wenfei, F. and Kementsietsidis, A.: Using Partial Evaluation in Distributed Query Evaluation. Proceedings of the 32nd international conference on Very large data bases 32 2006
7. Ezeife, C and Barker, K: A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object based System. Distributed and Parallel Databases 3 1995
8. Grün, C. and Holupirek, A. and Scholl, M.H.: Visually Exploring and Querying XML with BaseX. 2007
9. Hui, M. and Schewe, K.-D.: Fragmentation of XML Documents. Brazilian Symposium on Databases 2003
10. Karypis, G. and Kumar, V.: Parallel multilevel k-way partitioning schema for irregular graphs. Proceedings of the 1996 ACM / IEEE conference on Supercomputing 1996
11. Lin, X. and Orlowska, M. and Zhang, Y.: A Graph-Based Cluster Approach for Vertical Partitioning in Databases Systems. Data & Knowledge Engineering 11 1993
12. Lu, K. and Zhu, Y. and Sun, W. and Lin, S. and Fan, J.: Parallel processing XML documents. Proceedings of the Database Engineering and Applications Symposium 2002
13. Lu, W. and Chiu, K. and Pan, Y.: A Parallel Approach to XML Parsing. The 7th IEEE/ACM International Conference on Grid Computing 2006
14. Ma, H. and Schewe, K.D.: Fragmentation of XML Documents. Proceedings XVIII Simposio Brasileiro de Bancos de Dados (SBBD 2003) 2003
15. Ma, H. and Schewe, K.D.: Heuristic Horizontal XML Fragmentation. Proceedings of CAiSE 2005
16. Olteanu, D. and Meuss, H. and Furche, T. and Bry, F.: XPath: Looking Forward. XML-based Data Management and Multimedia Engineering—EDBT 2002 Workshops: EDBT 2002 Workshops XMLDM, MDDE, and YRWS, Prague, Czech Republic, March 24-28, 2002: Revised Papers 2002
17. Schmidt, A. R. and Waas, F. and Kersten, M. L. and Florescu, D. and Carey, M.J. and Manolescu, I. and Busse, R.: Why and How to Benchmark XML Databases. ACM SIGMOD Record 35 2001 27–32
18. Suciu, D.: Distributed Query Evaluation on Semistructured Data. ACM Transaction on Database Systems 27 2002