

XML-OLAPのための構造に基づくグルーピング処理

キットチャントラ[†] 天笠俊之^{†,††} 北川博之^{†,††}

[†] 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻

^{††} 筑波大学計算科学研究センター

〒 305-8573 茨城県つくば市天王台 1-1-1

E-mail: [†]kchantola@kde.cs.tsukuba.ac.jp, ^{††}{amagasa,kitagawa}@cs.tsukuba.ac.jp

あらまし Extensible Markup Language (XML) はデータ交換と表現のための重要なデータ形式となっている。XML データから隠された情報を引き出すために、従来の問合せ処理に加え、より複雑な分析処理が今後重要になると思われる。関係データベースを対象にした OLAP 手法は過去に多くの研究がなされ、実際に実社会で利用されているが、XML データに対する OLAP は検討が始まったばかりであり、研究者の注目を集め始めているところである。XML はその構造の柔軟性において、関係データベースと全く異なっている。XML データの対話的分析処理の効率化のため、我々は XML データのデータキューブ計算に、XML 問合せ処理手法の一つである構造結合演算を適用することを提案する。さらに、提案手法の妥当性を実験により示す。

キーワード XML, OLAP, XPath, XQuery, Structural Join

An Efficient Structure-based Grouping for XML-OLAP

Chantola KIT[†], Toshiyuki AMAGASA^{†,††}, and Hiroyuki KITAGAWA^{†,††}

[†] Department of Computer Science, Graduate School of Systems and Information Engineering

^{††} Center for Computational Sciences

University of Tsukuba

1-1-1 Tennodai, Tsukuba, Ibaraki 305-8573, Japan

E-mail: [†]kchantola@kde.cs.tsukuba.ac.jp, ^{††}{amagasa,kitagawa}@cs.tsukuba.ac.jp

Abstract *Extensible Markup Language (XML)* has become an important format for data exchange and representation on the web. In addition to conventional query processing, more complex analysis on XML data is considered to become important in order to discover valuable information. Seeing that OLAP system is well known for its mature performance with relational database system, how to apply OLAP for XML data has become a popular topic for many researchers. However, XML format is quite different from relational databases because of its semistructure and flexibility. To contribute to the need of XML data analysis, we propose *an efficient algorithm for XML cube computation* by applying a well known method of OLAP cube computation for XML data and optimizing with both value- and structure-based hierarchies. Moreover, we will show OLAP query extension with XML data, especially, *structure-based ROLLUP operation (TOPOLOGICAL ROLLUP) which has not been done before*.

Key words XML, OLAP, XPath, XQuery, Structural Join

1. Introduction

Since its emergence in 1998, the Extensible Markup Language (XML) [1] has become a de facto standard for data exchange and representation on the web. Now, XML has been used in a wide spectrum of application domains, such as web documents, business documents, and log data. For this reason, in addition to such simple query retrieval, more complex ways to make analysis of XML data

are considered to be more and more important in order to extract useful information from massive XML data.

For example, Figure 1 shows a piece of XML data from Shakespeare's play. The play contains two actions and each action contains one or many scene(s). There are some speeches in each scene and the speaker in the speech will say some lines. Suppose that we would like to count the occurrences of some words or phrases. The counts can be investigated in different levels, such as scene, speech,

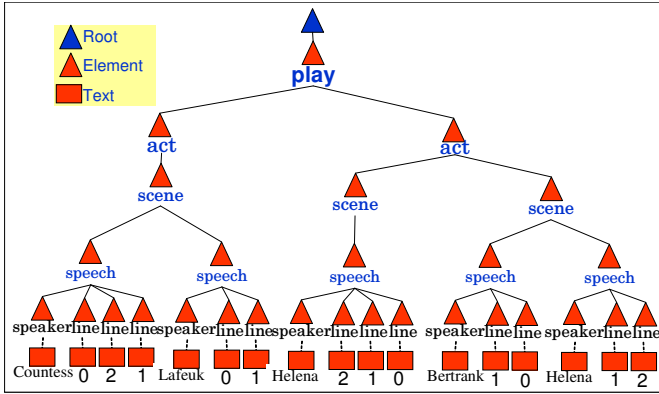


图 1 Shakespeare's play XML data.
Fig. 1 Shakespere's play XML data.

and line, depending on the analyzers' interests. Such an operation is useful in many applications. Statistical philology is one of the major applications. However, it is not straightforward to do this kind of analysis with existing OLAP systems, because we need to care about hierarchical nature of XML.

In our previous paper [2], we discussed an approach to XML-OLAP system based on relational database systems. We discussed a logical data model of XML datacube and an implementation using relational XML database technology. In this paper, we attempt to give a focus on the ROLLUP operation according to XML hierarchy as mentioned above. Such an operation is called "TOPOLOGICAL ROLLUP".

We make use of structural join algorithms as a mean to perform TOPOLOGICAL ROLLUP efficiently. Specifically, we employ *Stack Tree Join Ancestor / Descendant (STJA/D)* algorithms. STJA/D enable us to find matching pairs of ancestor and descendant nodes. We use STJA/D to compute different levels of grouping of XML nodes in top-down (bottom-up) direction, that is from the grand (detailed) level to detail (grand) level. Additionally, in the bottom-up process, we try to make use of the result from the previous step for upper level grouping without repeating the detailed level traverse. We compare the performances of the proposed algorithms, and show that TOPOLOGICAL ROLLUP can be computed efficiently.

The rest of this paper is organized as follows: in Section 2, we discuss related works. In Section 3, we describe our proposed algorithms for TOPOLOGICAL ROLLUP operation. We also show our implementation based on relational databases in Section 4 and our experimentation and evaluation in Section 5. Finally, in Section 6, we give conclusion and our future work.

2. Related Works

Wiwatwattana et al. [3] argued the straightforward way of extending relational cube of XML. They found that facts in relational warehouse are flat records and dimensions may have hierarchies, but in XML warehouse, both facts and dimensions may be hierarchical. XML is flexible: (a) an element may have missing or repeated subelements; (b) different instances of the same element type may

have different structure. They identified the challenges introduced by these features of XML for cube definition and computation. They proposed a definition for cube adapted for XML data warehouse, including a suitably generalized specification mechanism. They define a cube lattice over the aggregates so defined. They also identify properties of this cube lattice that can be leveraged to allow optimized computation of the cube. Finally, they present the results of an extensive performance evaluation experiment gauging the behavior of alternative algorithms for cube computation.

Bordawakar et al. [4] investigated various issues related to XML data analysis, and proposed a logical model for XML analysis based on the abstract tree-structured XML representation. In particular, they proposed a categorization of XML data analysis system: 1) XML is used simply for external representation for OLAP results, 2) Relational data is extracted from XML data, and then processed with existing OLAP systems, 3) XML is used for both data representation and analysis. In order to support complex analytical operations, they also proposed new syntactical extensions to XQuery, such as "GROUP BY", "ROLLUP", "TOPOLOGICAL ROLLUP", "CUBE", and "TOPOLOGICAL CUBE". In our research, we employ the syntax of "GROUP BY ROLLUP" and "GROUP BY TOPOLOGICAL ROLLUP" to allow users to specify OLAP operation in XQuery.

Jensen et al. [5] proposed a scheme for specifying OLAP cubes on XML data. They integrated XML and relational data at the conceptual level based on UML, which is easy to understand by system designers and users. In their scheme, a UML model is built from XML data and relational data, and the corresponding UML snowflake diagram is then created from the UML model. In particular, they considered how to handle dimensions with hierarchies and ensuring correct aggregation.

Pedersen et al. [6] proposed a federation of OLAP and XML, which allows external XML data to be presented along with dimensional data in OLAP query results. It enables the uses of external XML data for selection and grouping. It is the same to the third approach mentioned by Rajesh Bordawakar et al. [4]. They allow XML data to be used as "virtual" dimensions, and present a data model and multi-schema query language based on SQL and XPath.

Agarwal et al. [7] presented a *fast algorithms for computing a collection of group-bys*. They focused on a special case of the aggregation problem (computation of CUBE operator) which requires computing group-bys on all possible combinations of a list of attributes, and is equivalent to the union of a number of standard group-by operations. They showed how the structure of CUBE computation can be viewed in terms of a hierarchy of group-by operations. Their algorithms extend sort-based and hash-based grouping methods with several optimizations (*smallest-parents*, *cache-results*, *amortize-scans*, *share-sorts*, and *share-partitions*), like combining common operations across multiple group-bys, caching, and using pre-computed group-bys. Their experiment showed that the resulting algorithms give much better performance compared to straightforward methods.

Khalifa et al. [8] developed two families of structural join algo-

rithm for XML tree structure relationships: tree-merge and stack-tree. The Tree-merge and the recently proposed multi-predicate merge joins, while the stack-tree algorithms have no counterpart in traditional relational join processing. They presented experimental results on a range of data and queries using TIMBER native XML query engine built on top of SHORE. They showed that while, in some cases, tree-merge algorithms can have comparable performance to stack-tree algorithms, but in many cases, they are considerably worse. This behavior is explained by analytical result demonstrated on sorted inputs which showed that the stack-tree algorithms have worst case I/O and CPU complexities linear in the sum of the sizes of inputs and output while the tree-merge algorithms do not have the same guarantee.

3. Stack Tree Join for TOPOLOGICAL ROLLUP

One possibility to improve TOPOLOGICAL ROLLUP is to use a dedicated algorithms to compute structural relationships among sets of XML nodes, that is, structural joins.

3.1 Stack Tree Join Algorithm

Structural tree join [8] (STJ for short) is an algorithm for finding the relationship between sets of XML nodes. This algorithm can help us to find nodes in the same group. In the course of Depth-first traversing of a tree that can be performed in linear time using a stack of size as large as the height of the tree, every ancestor-descendant relationship in the tree is manifested by the descendant node appearing somewhere higher on the stack than the ancestor node. STJ traverses only the candidate nodes provided by two input lists: ancestor node list $AList = a_1, a_2, \dots, a_i$ and descendant node list $DList = d_1, d_2, \dots, d_j$. Each node contains did , $startPos$, $endPos$, and $value$ (document ID, start position number, end position number, and node value, respectively). STJ will compare $startPos$ and $endPos$ number of each $AList$ node with each $DList$ node to find the parent-child relationship. If STJ finds that d_j is not a_i children, STJA will output $(a_i, d_0)-(a_i, d_{j-1})$ to output table in the order of ancestor node and go to the next node, a_{i+1} . We suppose that all data are in the same document.

STJ family has two variants with respect to the way how the output is ordered by ancestor nodes (STJA) or descendant nodes (STJD). In our work, we proposed three algorithms for TOLOGICAL ROLLUP operation using either STJA or STJD.

3.2 STJ for TOPOLOGICAL ROLLUP

We can obtain our TOPOLOGICAL ROLLUP by making use of STJ as in the following order of process.

- 1 compute the pairs of parent-child nodes.
- 2 group the descendant nodes which have the same parents.
- 3 repeat the STJ process for each hierarchical level.

As ROLLUP can be done in two directions, top down and bottom-up, we propose three algorithms as follows:

- Top down algorithm using STJA (TOD-STJA)
- Bottom up algorithm using STJA (BUC-STJA)
- Single scan algorithm using STJD (SSC-STJD).

For more detail, let us look at each algorithm in the following subsections.

3.2.1 Top-down Algorithm using STJA (TOD-STJA)

TOD-STJA computes the grouping from the grand level to the detail level. The input for TOD-STJA will be $ALists$ of all hierarchical levels and a $DList$ which is the measure to be grouped. We firstly use STJA to join and group $DList$ with the top level $AList$. Then, the same $DList$ is repeatedly used to join and group with lower levels. Finally, we get all groupings of all hierarchical levels of XML data in top-down order.

For example, Figure 2 shows TOD-STJA joining in Shakespeare's play XML data. In this figure, the total number of word "can" can be computed by joining $AList$ of a *play* node with $DList$ of *line* nodes. Then the same $DList$ of *line* nodes is reused to join with other $ALists$ of lower levels of play (*act*, *scene*, and *speech*). Finally, the output will be the total number of word "can" grouped by the whole *play* followed by the lower level groupings.

Notice that, we can view our output as ROLLUP order by sorting our result in descending order.

3.2.2 Bottom-up Algorithm using STJA (BUC-STJA)

In reverse order of process, BUC-STJA computes groupings from the detail level to grand level. In this process, we use the grouping results from lower level as $DList$ for higher level groupings so that we also have many input $ALists$ of all hierarchical levels. This technique is effective in helping us to save our ROLLUP processing time. The output of the algorithm will be the ROLLUP groupings of all XML hierarchical levels in descending order.

Let us look at an example with Shakespeare's play XML data in Figure 3. BUC-STJA firstly joins and groups $AList1$ of lowest level, *speech*, nodes with $DList1$ of *line* nodes. The result of joining $AList1$ and $DList1$ is the total number of word "can" grouped by each *speech*. Then, this result can be used as $DList2$ for joining with $AList2$ of higher level, *scene* nodes, and further result is continuously used until we reach the top level, *play* node.

By using the result from lower level for current level grouping, we can finally get all groupings by all hierarchical levels without repeatedly using the large $DList1$ input. The reason is that the higher the level is, the fewer nodes it obtains.

3.2.3 Single Scan Algorithm using STJD (SSC-STJD)

SSC-STJD outputs the result as in the same order of BUC-STJA. By the way, SSC-STD is different from BUC-STJA in that it contains only one input $AList$ and one input $DList$. So it does not use the result from lower level for current level grouping and we can reduce the cost of producing $DList$ for each level. Second, SSC-STJD makes grouping in the way of traversing XML tree. The traversing process enables us to partition XML data in case that the data size is very large and there are a lot of nodes for grouping.

To see how SSC-STJD traverses XML nodes for ROLLUP grouping, we give an example as in Figure ???. We have an $AList$ of all nodes of Shakespeare's play including *play*, *act*, *scene*, *speech* (the nodes in blue circles) without separating $AList$ by each level. The same to $DList$ in the TOD-STJA, and $DList$ for lowest level joining in BUC-STJA, SSC-STJD's $DList$ (the nodes in red circle) contains all *line* nodes. By using STJD algorithm, the process starts from the top node, *play*, and go down for its children, and descen-

dants. When it find the last descendant node (the first *line* node) the measure 0 of the line will be added to the parent node, *speech* and then it goes to its sibling (the second *line* node) which its measure of 2 is added to the same *speech* parent node. The process will continue to other children and descendant nodes of *play* as the arrow lines' directions in the figure till we get back to the top node, *play*.

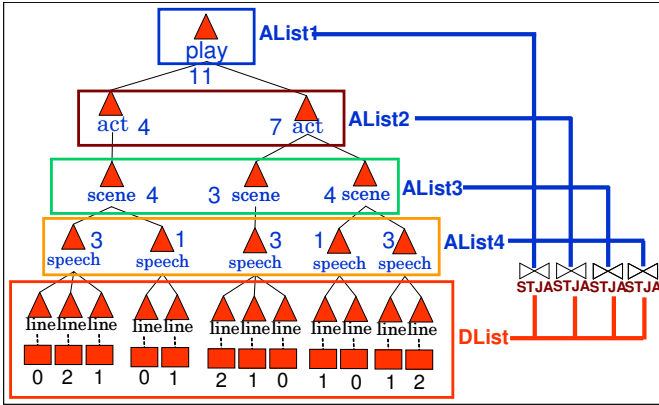


Fig. 2 TOD-STJA.

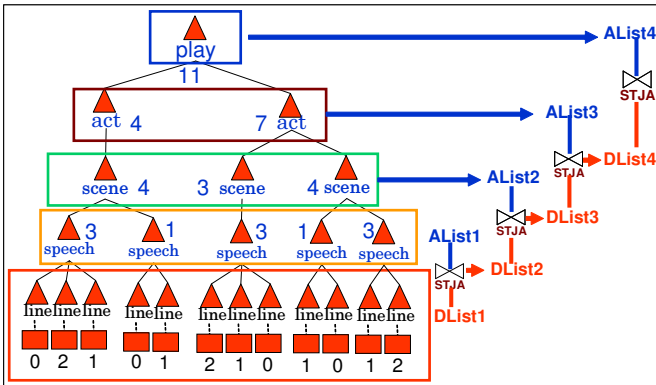


Fig. 3 BUC-STJA.

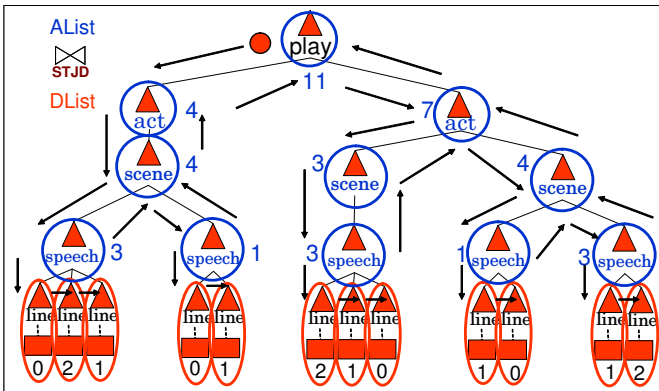


Fig. 4 SSC-STJD.

4. System Implementation

As we can see the system implementation overview in Figure 5, XML data are decomposed and stored in relational tables in advance. We employ the path-approach[9] for mapping XML data to relational tables, because we can manage any well-formed XML documents with fixed relational schema and realize practical subset of XPath solely by the use of SQL functionalities. In the path-approach, an XML node is basically mapped to a relational tuple of two tables, path table which contains all absolute path expression of all XML nodes, and node table which contains all XML node information. Table 1 (left) shows the path table extracted from Shakespeare's play XML data. The attributes *pid* and *pexp* in the path table denote path id to join path table with node table, and the absolute path of XML node respectively. In the node table (Table 1, right), there are *did*, *pid*, *startPos*, *endPos*, *name*, and *value*. Attribute *did* denotes the id of the XML document, *pid* is the path id referring to the path expression in the path table, *startPos* and *endPos* are start position and end position numbers used to identify the node, *name* denote the node type, which is either of element name, "#TEXT", "@attribute", or "CDATA" depending on the type of the node. The last column is the value of the text or attribute node.

Then, users issue requests for OLAP analysis in some means. A possible way is to utilize XQuery with OLAP extension [4] The system translates the given query into SQL queries so that it can be processed in the underlying database system. If the request contains "TOPOLOGICAL ROLLUP" statement, the system extracts *AList* and *DList* from path and node tables. These extracted *AList* and *DList* are the elements of our proposed algorithms input that finally, our system can provide the result to the user's query.

For instance, to ROLLUP count the number of word "can" as mentioned previously, a user gives an XQuery as in Appendix 1.. By using SQL translator, we get two SQL queries, Appendix 2. and 3., for extracting *AList* and *DList*. Then our proposed ROLLUP algorithm uses *AList* and *DList* as the input for grouping the number of word "can" in each level and provides the output as in Table 2 to the user.

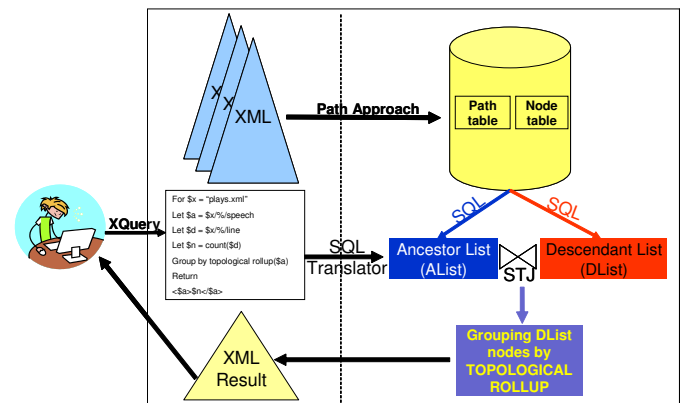


Fig. 5 System Overview.

表 1 Path table and node table.

Table 1 Path table and node table.

pid	pexp	did	pid	startpos	endPos	name	value
1	/play/act/scene/speech/speaker	0	6	1	73	play	null
2	/play/act/scene/speech/line	0	5	2	30	act	null
3	/play/act/scene/speech	0	4	3	29	scene	null
4	/play/act/scene	0	1	5	7	speaker	null
5	/play/act	0	1	6	6	#TEXT	Countess
6	/play	0	2	8	10	line	null
		0	2	9	9	#TEXT	Who go...
		0

表 2 Result of Shakespeare’s play TOPOLOGICAL ROLLUP.

Table 2 Result of Shakespeare’s play TOPOLOGICAL ROLLUP.

startPos	endPos	name	value
4	17	speech	3
18	28	speech	1
33	46	speech	3
49	59	speech	1
60	70	speech	3
3	29	scene	4
32	49	scene	3
48	71	scene	4
3	30	act	4
31	72	act	7
1	73	play	11

5. Experimental Evaluation

5.1 Experimental Setup

All experiments were performed in Sun Microsystems Sun Fire X4200 server whose CPU is a 2-way Dual Core AMD Opteron(tm) processor (2.4GHz). This machine has 16GB memory and runs Sun OS 5.10. We used Java version 1.5.0.09 to parse XML data to relational tables, and PostgreSQL 8.2.6 to perform query processing.

For the experimental data, we used XMark data which is a comprehensive distributed system benchmarking and optimization suite. Figure 6 depicts the structure of the XMark data. We chose “regions” element for our experiment. Each “region” node contains child nodes representing world continents, like Africa, Asia, Australia, Europe, North America, and South America and some other child nodes. Each Continent node contains several “item” elements, and others. Each “item” node contains “quantity”, “payment”, and some other child nodes. We tested the following sizes of XML data: 10MB, 100MB, and 200MB.

5.2 Benchmark Queries

We performed four processes of TOPOLOGICAL ROLLUP the quantity of XMark data by regions. The first process is the intuitive technique of using the trivial SQL query. We call the process TOD-SQL. The other processes are the process of the proposed algorithms, TOD-STJA, BUC-STJA, and SSC-STJD.

5.3 Experimental Results

The experimental result in Table 3 and Figure 7 showed that both TODs consumed much more time than BUCs. BUC-STJA performed well except for the lowest level grouping that STJA ROLLUP performed worse than the former ROLLUP. This worse

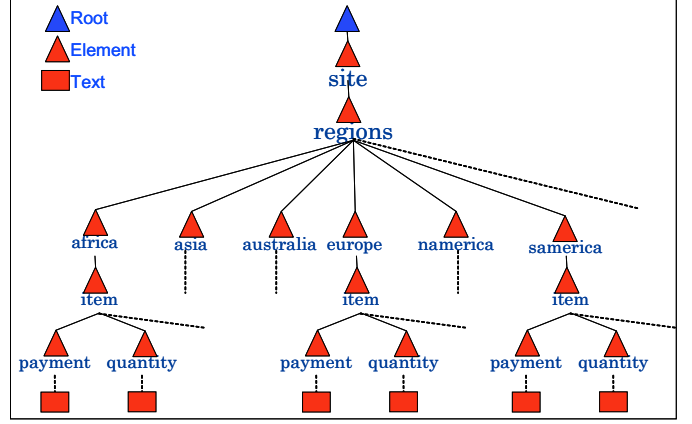


图 6 XMark data.

Fig. 6 XMark data.

表 3 Table of Time performance for XMark ROLLUP (ms).

Table 3 Table of Time performance for XMark ROLLUP (ms).

Algorithms	10MB	100MB	200MB	300MB	400MB	500MB
TOD-SQL	35	3798	7537	11524	15541	19803
TOD-STJA	261	2705	4756	6888	8969	11364
BUC-STJA	443	2551	4504	6611	8682	10975
SSC-STJD	229	2316	4315	6472	8465	10708

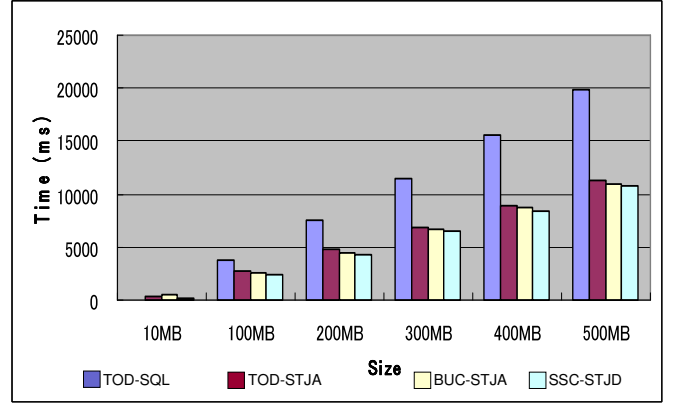


图 7 Time performance for XMark ROLLUP (ms).

Fig. 7 Time performance for XMark ROLLUP (ms).

case comes from the extra time of creating table after STJA joined AList and DList while TODs does not need to do so. For larger XML data, STJA ROLLUP took about two times less than TODs because the number of higher grouping always decreases. For SSC-STJD, the time performance is similar to BUC-STJA’s but it had the best performance for all data sizes that for 10MB data, it consumed less time than BUC-STJA and TOD-STJA.

6. Conclusions

In this paper, we discussed grouping and ROLLUP operations for XML data which we focus on structure-based grouping, TOPOLOGICAL ROLLUP operation. We proposed three algorithms for TOPOLOGICAL ROLLUP using Stack tree join. We also show our implementation based on relational databases. Our experiment with large collection of XMark data showed the effectiveness of our proposed algorithms.

For future works, we plan to make our experiment with XML data of high-level hierarchies as our experiment with XMark data contain only two levels. In addition to our current proposed algorithms, we are thinking of improving our SSC-STJD algorithm by eliminating the stack usage of STJD. We expect that this new algorithm will enable us to reduce our ROLLUP operation cost. Another benefit from SSC algorithm is that we can apply partitioning technique to our algorithm to cope with very large XML data.

Acknowledgments

This research is partly supported by the Grant-in-Aid for Scientific Research (19024006) from Japan Society for the Promotion of Science (JSPS), Japan, and the Grant-in-Aid for Scientific Research on Priority Areas (19700083) from the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan.

文 献

- [1] World Wide Web consortium: Extensible Markup Language (XML) 1.0 (Third Edition), <http://www.w3.org/TR/REC-xml>. W3C Recommendation 04 February 2004.
- [2] Chantola Kit, Toshiyuki Amagasa, and Hiroyuki Kitagawa. OLAP Query Processing for XML Data in RDBMS. *IEEE International Workshop on Databases for Next Generation Researchers(SWOD)*, pages 7–12, 2007.
- [3] Nuwee Wiwatwattana, H. V. Jagadish, Laks V. S. Lakshmanan, and Divesh Srivastava. X^3 : A Cube Operator for XML OLAP. *IEEE International Workshop on Databases for Next Generation Researchers(SWOD)*, pages 7–12, 2007.
- [4] Rajesh Bordawakar and Christian A. Lang. Analytical Processing of XML Documents: Opportunities and Challenges. *SIGMOD Record*, 34(2):27–32, 2005.
- [5] Mikael R. Jensen, Thomas H. Moller, and Torben Bach Pedersen. Specifying OLAP Cubes on XML Data. *SSDBM*, pages 101–112, 2001.
- [6] Dennis Pedersen, Karsten Riis, and Torben Bach Pedersen. XML-Extended OLAP Querying. *SSDBM*, pages 195–206, 2002.
- [7] Sameet Agarwal, Rakesh Agrawal, Prasad M. Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi. On the Computation of Multidimensional Aggregates. In *Proc. of VLDB*, 1996.
- [8] Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel, Divesh Srivastava, and Yuqing Wu. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *Proc. of ICDE 2002*, 2002.
- [9] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura. XRel: A path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology (TOIT)*, 1(1):110–141, 2001.
- [10] Kenneth A. Ross and Divesh Srivastava. Fast Computation of Sparse Datacubes. In *Proc. of VLDB*, 1997.
- [11] Kevin Beyer and Raghu Ramakrishnan. Bottom-Up Computation of Sparse and Iceberg CUBES. *Proc. ACM SIGMOD 1999*, pages 359–370, 1999.

付 録

1. XQuery: A User ROLLUP Query Example

```
For $x = "plays.xml"
Let $a = $x/~/speech
Let $d = $x/~/line
Let $n = count($d)
```

```
Group by topological rollup($a)
Return
<$a>$n</$a>
```

2. SQL: AList

```
SELECT n.did, n.startPos, n.endPos, n.value
FROM ntable n, ptable p
WHERE n.pid = p.pid
AND (p.pexp LIKE '/play'
OR p.pexp LIKE '/play/act'
OR p.pexp LIKE '/play/act/scene'
OR p.pexp LIKE '/play/act/scene/speech')
AND n.value = ''
ORDER BY n.did, n.startPos;
```

3. SQL: DList

```
SELECT n.did, n.startPos, n.endPos, n.value
FROM ntable n, ptable p
WHERE n.pid = p.pid
AND p.pexp LIKE '/%/line'
AND n.name = '#TEXT'
ORDER BY n.did, n.startPos;
```