A Development of an XML-OLAP System

Chantola KIT[†1]

Extensible Markup Language (XML) has become an important factor 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 systems, 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, in our previous work[5)], we proposed an XML-OLAP system which is a core operation in the interactive analysis of XML data. Moreover, in [6)], we proposed several efficient algorithms for aggregation computation according to XML tree hierarchy or **TOPOLOGICAL ROLLUP** which play a key role in XML structure-based grouping. Initially, we proposed two basic algorithms: **Top-Down** and **Bottom-Up**, which are based on well known Structural Join Algorithms[1)]. We then proposed a modified Bottom-Up algorithm to improve applicability, and the **Single-Scan** algorithms that make use of dedicated data structures. Our experiments with synthetic XML data show the effectiveness of the proposed algorithms.

1. Introduction

Since the Extensible Markup Language (XML) has become a de facto standard for data exchange and representation on the web, XML has been used in a wide spectrum of application domains, such as Web documents, business documents, and log data.

An example of XML data, which can be seen in Figure 1, is coding the content of a bookstore, "Bookstore.xml". The bookstore consists of `book`, `magazine`, and `DVD` as its items. All the items are stored in accordant to the classified categories and subcategories, such as `math`, `linear`, `cs` (computer science), `db` (databases), and `web`. Specifically, each item contents author's name (shortly reprensented as `a` in the figure), title (`t`), quantity (`q`), and text(`txt`).

In order to extract information from XML data, generally, we can use existing XML query languages, such as XPath and XQuery, or XML information retrieval (XML-IR). These kind of XML queryings are based on the basic functions provided by the query languages or the provided keywords. However, the more complex ways to make analysis of XML data are considered to be extremely important in order to extract useful information from massive XML data.

†1 Department of Computer Science, Graduated School of Systems and Information Engineering

Several researchers have been working in this area of research over the past few years; some of their work is discussed in Section 2. According to those preceding research efforts, several operations specific to XML-OLAP have been proposed. Among those operations, structure-based grouping or "TOPOLOGICAL ROLLUP" is considered to play an important role because XML structure is fundamentally modeled as a tree.

For example, in the above bookstore example, a user may want to compute total quantity of the books by the subsidiary categories. To do so, we group the quantity from the bottom level, and sum it up to the root node. Since the book nodes can be recursively grouped by their categories in upper levels, we need to repeat grouping operations many times; this is obviously time consuming.

Refferring to the underlying issues, in our previous work [5)], we proposed XML-OLAP system and recently[6)], we discussed various effective algorithms for TOPOLOGICAL ROLLUP operation.

For the rest of this report, we discuss some related work in Section 2, then give an overview of our XML-OLAP system in Section 3. In Section 4, we discuss several proposed algorithms starting from the trivial to efficient algorithms. In order to show the effectiveness of the proposed algorithms, we make experimental per-

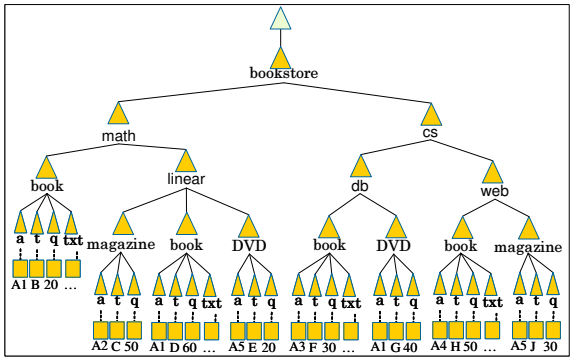

Fig. 1 Bookstore.xml

formance and evaluation with synthetic data in Section 5. Finally, we show our GUI for XML-OLAP in Section 6 and give the conclusion in Section 7.

2. Related Work

While diversification of data spreads rapidly, interactive analysis on XML data is becoming so curcial and popular for many researchers.

Bordawekar et al.[2] investigate various issues related to XML data analysis and propose a logical model for XML analysis based on abstract tree-structured XML representation. In particular, they propose a categorization for the XML data analysis system: 1) XML is used simply to represent, externally, 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. To support complex analytical operations, they also propose new syntactical extensions to XQuery, such as "GROUP BY," "TOPOLOGICAL ROLLUP," "CUBE," and "TOPOLOGICAL CUBE." Notice that they focus on the modeling and syntactic aspects of XML-OLAP, and do not develop discussions about implementation, which our work attempts to do.

Jensen et al.[4] propose a scheme to specify OLAP cubes on XML data. They integrate 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; the corresponding UML snowflake diagram is then created from the UML model. In particular, they consider how to handle dimensions with hierarchies and how to ensure correct aggregation. They do not, however, focus on structure-based grouping like the algorithms we propose for TOPOLOGOGICAL ROLLUP.

Pedersen et al.[7] propose a federation of OLAP and XML, which allows external XML data to be presented along with dimensional data in OLAP query results. It enables external XML data to be used for selection and grouping. It is the same as the third approach in[2]. They allow XML data to be used as so called "virtual" dimensions, and present a data model and multi-schema query language based on SQL and XPath. Related to this research, our work goes a step farther with efficient algorithms for structural-based grouping coming after trivial SQL and/or XPath.

Wiwatwattana et al.[8] propose a straightforward way to extend the relational cube to XML. They find that, in the XML warehouse, both facts and dimensions can be hierarchical, whereas the facts stored in a relational warehouse are flat. Beyond that, XML is flexible: (a) an element may have missing or repeated sub-elements; and (b) different instances of the same element type may have different structures. They identify the challenges introduced by these features of XML for cube definition and computation. They eventually propose the definitions of cube and cube lattice adapted for XML data warehouses. They also identify several properties of the cube lattice that can be leveraged for optimized computation. Their purpose is similar to[4], which we see our advanced work: OLAP extension for XML data analysis.

Gokhale et al.[3] investigate efficient algorithms for the group-by operator on XML _with the goal of supporting a full spectrum of aggregation operations. Their work includes holistic operations, such as `median()` and complex nested grouping and aggregations. Additionally, syntactical elements, such as `having clause` and `window` aggregation, are also considered. They propose a framework to express complex aggregation queries on XML data featuring nested group-by, having clause, and moving windows. They also develop a disk-based algorithm to efficiently evaluate queries involving subsets of the above features. Their research

focuses on some possibilities of structure-based aggregation while our research implements the TOPOLOGICAL ROLLUP for XML-OLAP.

In conclusion to related work, we develop concrete algorithms for "TOPOLOGICAL ROLLUP," a concept originally proposed by Bordawekar et al.[2]. To the best of our knowledge, there has been no preceding research addressing the problem of how to efficiently implement TOPOLOGICAL ROLLUP operations.

3. XML-OLAP System Overview

This section describes an overview of our proposed XML-OLAP system based on relational databases[5]. In Figure 2 (left side), according to the content of XML data, a user firstly gives a fact path and some dimensioin paths in XPath expression to denote his/her interest by which the system produces an XML cube. Then, the user can make analysis of XML data-cube using XQuery with OLAP extensions. We are based on relational databases because there are many off-the-shelf systems and they provide good performance.

In the right side of Figure 2, XML documents are translated into relational tables: node table and path table. Node table stores path ID, preorder, postorder, node type, and value of every XML nodes while path table stores path ID and path expression of the node. Then, the user given fact and dimension paths are translated to SQL query to extract all fact and dimensions from relational tables and form XML data-cube. Finally, the given XQuery with OLAP extension, specifically TOPOLOGICAL ROLLUP, will also be converted into SQL query in order to extract the user query result from XML data-cube.

Since we had noticed that our TOPOLOGICAL ROLLUP computation by SQL consumed a lot of time, we proposed several efficient algorithms for TOPOLOGICAL ROLLUP based on well known algorithms, Structural Join[1], which will be shown in the following sectioins.

4. TOPOLOGICAL ROLLUP Operation

Before we go deeply to the proposed algorithms, let us show the definition of TOPOLOGICAL ROLLUP. Originally, the idea is from the OLAP extension to SQL, so we start by briefly explaining ROLLUP operation in existing OLAP systems; we then explain TOPOLOGICAL ROLLUP for structure-based grouping in XML-OLAP, which is an OLAP extension to XQuery.

4.1 ROLLUP Operation in Existing OLAP Systems

Aggregation is a fundamental part of data warehousing; it comprises various operations, such as data cube operations, complex aggregations (median, variance, etc.), binary aggregations (correlation, regression curves, etc.), and ranking queries. To meet demand, SQL-99 OLAP extensions provide a variety of aggregation functions that allow users to specify their information needs. It includes GROUP BY and GROUPING SETS, ROLLUP and CUBE, RANKING, and WINDOWING.

Let us look closely at the ROLLUP operation. In existing OLAP systems, ROLLUP enables a SELECT statement to calculate multiple levels of aggregations across a specified group of dimensions. Specifically, it firstly calculates aggregations at the most detailed level; it then calculates coarser levels of aggregations up to the grand level according to the expression specified in the ROLLUP clause.

The ROLLUP clause is very helpful for calculating subtotals in such a dimension, which has a deep hierarchical structure, e.g., time dimension (year, month, and day) and geographical dimension (country, state, and city). In addition, for data warehouse administration tasks using summary tables, ROLLUP can simplify and speed up the maintenance of summary tables.

4.1.1 STJ for TOPOLOGICAL ROLLUP

Depending on Stack Tree Join, we attempt to implement TOPOLOGICALROLLUP in the following steps:

(1) As the preprocess, we extract a single DList, which consists of XML nodes containing values to be aggregated, and multiple ALists, each of which consists of XML nodes in upper levels and corresponds to a specific level of groupings.

(2) For each list in ALists, we perform STJ between `AList` and `DList` to obtain the grouping at the particular level, followed by computation of aggregation values.

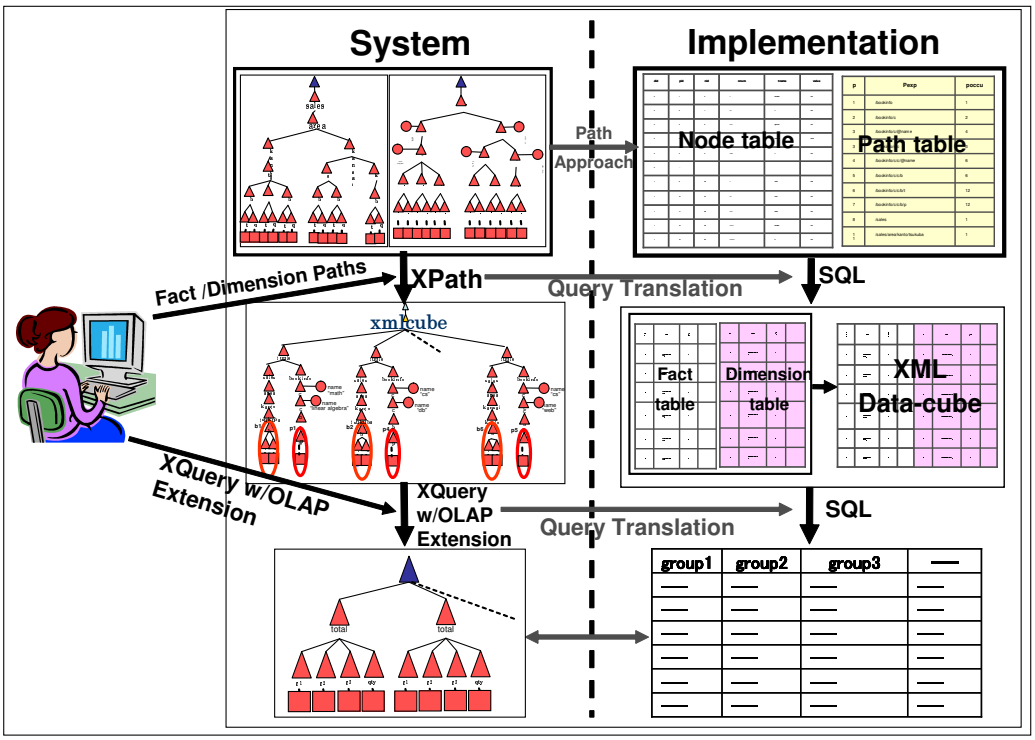

Fig. 2 XML-OLAP System Overview

Finally, we get the multiple levels of aggregations. The process can be implemented in two distinct directions: Top-Down and Bottom-Up. The following paragraphs describe the algorithms in detail.

4.1.1.1 Top-Down Algorithm (TDA)

The first algorithm is the Top-Down Algorithm (TDA). As the name implies, TDA uses STJ to compute the grouping from the topmost level to the more detailed levels.

Let us look back to the previous example in `Bookstore.xml` that a user would like to calculate the quantity of the books by categories. (Figure 3, left side) dipicts our algorithm process. We construct three ALists according to the levels of groupings, and also construct a single DList, which contains all `book` nodes. Notice that the topmost `bookstore` node is omitted from structural join process because there is no need for TDA to compute aggregations starting from the top level, AList1, by joining it with DList. Thus, we obtain the aggregations from the second level (AList2) and continue this process down to AList3. Finally, we get all levels of aggregations.

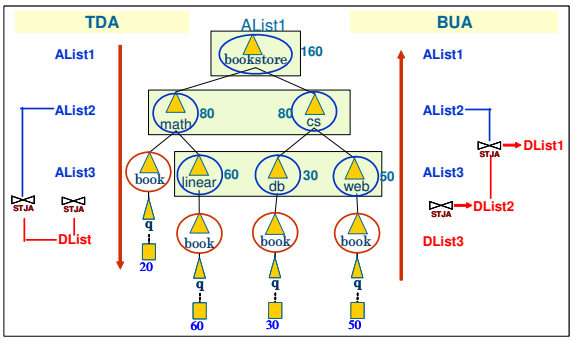

Fig. 3 Top-Down and Bottom-Up Algorithms.

4.1.1.2 Bottom-Up Algorithm (BUA)

You may notice that TDA is too straightforward and not at all efficient because we need multiple full-scans over DList. A possible workaround is to process in the opposite direction, starting with the lowermost grouping level and going up to the topmost level. The major advantage is that we can use the output of the previous grouping, whose size should be smaller because of grouping, as the input of the current grouping. Consequently, less processing time is gained than with TDA.

Look at the example on the right side of Fig-

ure 3, which illustrates the same query as the previous TDA example. This time, we start by joining AList3 with DList3, and obtain the aggregations as well as DList2. Notice that DList2 is the same size as AList3. We move ahead by performing STJ for joining AList2 with DList2, and so on. Using the result from the lower level instead of repeating DList1, we can obtain all groupings at all levels without performing a full scan over DList1 many times.

Having described TDA and BUA, we can observe their advantages and disadvantages:

- TDA is able to compute exact aggregation values, but it is relatively inefficient.
- BUA appears to be faster than TDA because it reduces full scans over DList; however, it may compute inaccurate aggregation values if grouping nodes and nodes to be aggregated are both residing at the same level. For instance, DList2, which is the result of STJ between DList1 and AList4, contains word counts of term "B" in `section(37,44)` and `section(45,52)`. After that, when we join DList2 with AList3, we eventually notice that `text(17)` and `text(25)` are not taken into account.

4.1.1.3 Bottom-Up Algorithm for Mixed Structure (BUA-Mix)

To cope with the above problem, we propose an improved version of BUA, which is called Bottom-Up Algorithm for Mixed Structure (BUA-Mix). Our idea is to keep descendant nodes, which are out of ancestors' scopes, in the output list. We do this because they may be joined with ancestors in upper levels. Consequently, even if BUA-Mix cascades the output of the current level to the upper level, it does not miss descendants that are at the same level as the aggregation nodes (ALists).

Figure 4 shows the algorithm for BUA-Mix, which has been modified from STJ. It contains three major conditions, and their intuitive explanations are as follows:

- The first condition `(a.pre < d.pre and d.post < a.post)` states that the current `a` node is an ancestor of descendant `d` node. If so, we update the aggregation value of `a`, and proceed to the next descendant.
- The second condition `(d.post < a.pre)` states that the descendant `d` precedes the current ancestor `a`. This means that `d` cannot be taken into account as a part of `a`, but may be used for future ancestors at upper levels. We therefore append it to the output for future usage instead of throwing it away.
- In other cases, we append the current ancestor `a` to the output, and move to the next ancestor.

```
Algorithm BUA-Mix (AList, DList)
/*Assume that all nodes in AList and DList have the same did*/
/*AList is the list of potential ancestors in sorted order of pre*/
/*DList is the list of potential descendants in sorted order of pre*/
a = AList → firstNode;
d = DList → firstNode;
output = Null;
while (AList and DList are not end of list) {
      if (a.pre < d.pre and d.post < a.post) {
          a.value = aggregate(a.value, d.value);
          d = d -> next;
      }
      else if (d.post < a.pre) {
          append d to output;
          d = d -> next;
      }
      Else {
          append a to output;
          a = a -> next;
      }
}
```

Fig. 4 Bottom-Up Algorithm for Mixed Structure (BUA-Mix).

Notice that `aggregate()` denotes a generalized aggregation function to compute aggregation, and should be replaced with specific aggregation functions, such as `sum`, `average`, `min`, and `max`.

4.2 Single-Scan Algorithms for TOPOLOGICAL ROLLUP Operation

In addition to the above algorithms, we propose more efficient algorithms for TOPOLOGICAL ROLLUP operation, *Single-Scan by Preorder Number (SSC-Pre)* and *Single-Scan by Postorder Number (SSC-Post)*, by which we can compute whole aggregations with a single scan over an AList and DList pair. Note that, in contrast to the above algorithms, a single AList, containing all ancestor nodes corresponding to aggregation levels, is being considered.

To make it possible to compute the whole aggregation within a single scan, we try to use a stack or a list as the data structure to maintain intermediate results. When we find a new matching pair of an ancestor and a descendant, we not only calculate current aggregation, but also propagate the result up to ancestors, thereby avoiding multiple scans over a number of ALists.

Note also that we propose two variations ac-

cording to the way AList and DList is ordered. Specifically, we care about two cases where the lists are sorted by preorder or postorder numbers. This is another difference between the above algorithms with which all lists are assumed to be ordered by a preorder number. One good reason to have algorithms for both orderings is that some XML database systems employ a postorder number as the order to organize their storage scheme while most systems use a preorder number. By choosing an appropriate algorithm depending on how underlying storage is organized, we can achieve better performance in implementing XML-OLAP.

The following subsections describe Single-Scan by Preorder Number, followed by Single-Scan by Postorder Number.

4.2.1 Single-Scan by Preorder Number (SSC-Pre)

SSC-Pre takes as its inputs a single AList, containing all ancestor nodes as grouping nodes sorted by preorder number, and a single DList, containing all descendant nodes to be aggregated and also sorted by their preorder numbers. It computes all aggregations while scanning AList and DList from heads to tails. To maintain intermediate aggregation results, we use a stack as temporary storage while scanning the inputs.

To see how SSC-Pre works, look at Figure 5. We have an AList consisting of `bookstore` and categories: `math`, `linear`, `cs`, `db`, and `web` nodes, and a DList of all `book` nodes. Each node of both lists contains document ID, preorder number, post order number and value. Both AList and DList are sorted by preorder number. The arrows in the figure show the process order among ancestor nodes. By referring to the ascending order of preorder number, SSC-Pre starts from the least preorder number ancestor node, `bookstore(1,113)`, and checks the following node in AList to determine whether or not it is subsumed. If it is, put the `bookstore(1,113)` node on the stack as in stack-output pair#1, and move to the second node `math(2,56)`. The same for `bookstore` node, `math(2,56)` node is also pushed onto the stack (see stack-output pair#1) and move to `linear(17,55)`. Here, we know that the next ancestor is beyond the region of `math`; we then check the DList. We try to find the first matching descendant by scanning DList. When we find a matching node, we compute the aggregation between the current ancestor and descendant. Also, we propagate the aggregation up to the ancestors in upper levels as in stack-output pair#2. They are actually stored before output, and thus we manipulate the stack to do so.

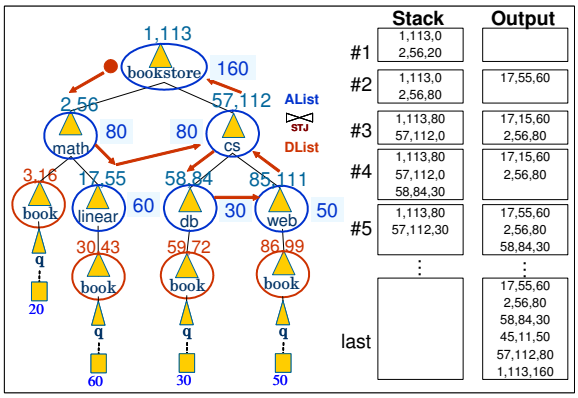

Fig. 5 Example of SSC-Pre in Book.xml Tree.

Figure 6 shows the algorithm of SSC-Pre. A rough sketch of this algorithm is as follows:

(1) The first condition `(stack.top.pos < a.pre and stack.top.pos < d.pre` states that both the current `a` and `d` nodes are not the descendant of the top node of the stack. This means that the top node of the stack reaches its last descendant node. We then move the top node of stack onto the output and also update the aggregation value of the new top node of the stack with the stack pop node value.

(2) The second condition `(a.pre < d.pre)` states that the current `a` node is an ancestor of `d` node. Here, we suspect that the next ancestor may also be the ancestor of the current `d`, so we keep the current `a` on the stack and proceed to the next ancestor.

(3) In other cases, we update the aggregation value of stack's top node with the current `d` value and move to the next descendant node.

4.3 Single-Scan by Postorder Number (SSC-Post)

SSC-Post outputs the result in the same order of SSC-Pre and BUA, and contains only one input *AList* and one input *DList* as in SSC-pre. By the way, SSC-Post differs from SSC-Pre in

```
Algorithm SSC-Pre (AList, DList)
/*Assume that all nodes in AList and DList have the same did*/
/*AList is the list of potential ancestors in sorted order of pre*/
/*DList is the list of potential descendants in sorted order of pre*/
a = AList → firstNode;
d = DList → firstNode;
output = Null;
while (AList and DList are not end of list) {
      if (stack.top.post < a.pre and stack.top.post < d.pre) {
          tuple = stack.pop();
          append tuple to output;
          stack.top.value = aggregate(stack.top.value, tuple.value);
      }
      else if (a.pre < d.pre) {
          stack.push(a);
          a = a -> next;
      }
      else{
          stack.top.value = aggregate (stack.top.value, d.value);
          d = d -> next;
      }
}
```

Fig. 6 SSC-Pre Algorithm.

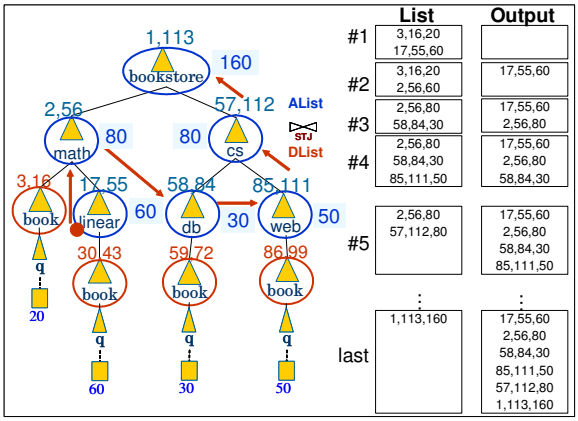

Fig. 7 Example of SSC-Post in Book.xml Tree.

that both the *AList* and *DList* nodes are in the order of postorder number whereas SSC-Pre's input lists are in the order of preorder number. SSC-Post makes the grouping by scanning the XML tree starting from the lowest level hierarchy until it reaches the top level. Doing so, SSC-Post can reduce the cost of traversing the XML tree, circled process of SSC-Pre (top-bottom-top), to a half circle, starting from the bottom and ending at the top (bottom-top). Moreover, SSC-Post also enables us to partition XML data when the data size is very large.

Let us look at the example in Figure 7. Illustrated by the arrows, SSC-Post starts from the first ancestor node, which is at the bottom level (`linear (17,55)`). There, the algorithm checks for the descendant node of `linear(17,55)` and updates its aggregate value, which we can see in the list-output pair#1 of the figure, and move the next ancestor node `math(2,56)`. Notice that since node `book(3,16)` is not the current ancesstor node `linear`, we push this node into list as in #1. Doing so, we will not miss this value when we go up to compute the aggregation of node `math(2,56)` which is the ancesstore of `book(3,16)`. At `math(2,56)` as well as the rest of ancestor nodes, SSC-Post repeats checking for all descendant nodes of this `math(2,56)` and store it in the list and output. The idea is that, for each ancestor node, SSC-Pre aggregates all of its descendant nodes' values and releases this ancestor node with its completed aggregate value to the output before it goes on to its sibling or its parent node. By repeating the process, which we can see in the arrow direction, SSC-Post finally provides an output list that contains all aggregate values grouped by category hierarchies.

4.3.1 SSC-Post Algorithm

More precisely, let us look at the algorithm of SSC-Post in Figure 8, which has four conditions and uses the list to keep intermediate aggregate values.

(1) The first condition `(a.pre < list.last.pre and list.last.post < a.post)` states that the current `a` is an ancestor of the last node of the list. If so, we update the aggregate value of `a` and remove the last node from the list.

(2) The second condition `(a.pre < d.pre and d.post < a.post)` states that the current `a` is an ancestor of `d`. We update the aggregate value of `a` and proceed to next descendant.

(3) The third condition `(d.post < a.pre)` states that `d` is the pre-sibling of `a`, in case of mixed structured XML data. Thus, we keep `d` into the list and move to next descendant node.

(4) In other cases, the current `d` is not the descendant of the current ancestor node (`a` is completed), but may be the descendant of the next ancestor node. Hence, we output ancestor node `a` and also keep it in the list before we move to the next ancestor node.

In the real world, some existing XML databases are in the order of preorder number and others are in the order of postorder number. That means our proposed SSC-Pre and

```
Algorithm SSC-Post (AList, DList)
/*Assume that all nodes in AList and DList have the same did*/
/*AList is the list of potential ancestors in sorted order of pre*/
/*DList is the list of potential descendants in sorted order of post*/
a = AList → firstNode;
d = DList → firstNode;
output = Null;
while (AList and DList are not end of list) {
      if (a.pre < list.last.pre and list.last.post < a.post) {
          a.value = aggregate(a.value, list.last.value);
          remove list.last;
      }
      else if (a.pre < d.pre and d.post < a.post) {
          a.value = aggregate(a.value, d.value);
          d = d -> next;
      }
      else if (d.post < a.pre){
          append d to list;
          d = d -> next;
      }
      else {
          append a to output;
          append a to list;
          a = a -> next;
      }
}
```

Fig. 8 SSC-Post Algorithm.

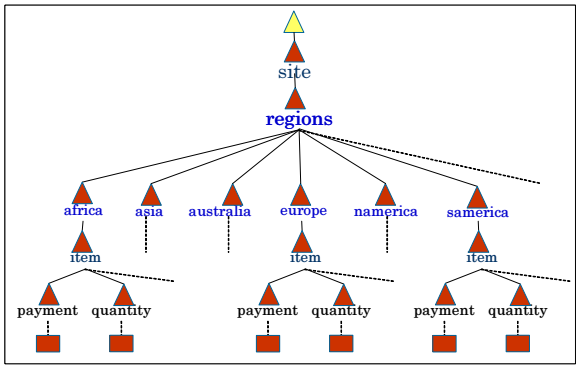

Fig. 9 XMark Tree.

SSC-Post is useful for all cases.

5. Performance Study

We have conducted a series of experiments to perform comparative analysis on the proposed algorithms. However, as mentioned in related work, our proposed algorithms are state of the art; there are no comparable algorithms from other research.

5.1 Experimental Setup

All experiments were performed on a Sun Microsystems Sun Fire X4200 server with a 2-way Dual Core AMD Opteron (tm) processor (2.4GHz). The machine has 16GBs of RAM and runs on Sun OS 5.10. We used Java version 1.5.0 to implement the algorithm. As underlying data storage, we used PostgreSQL 8.2.6 to store XML data. Specifically, we used the path-approach[9] to convert XML data to a relational table. When performing a TOPOLOGICAL ROLLUP operation, we first formed AList(s) and DList by issuing respective SQL queries, and applied the algorithm over the lists.

5.2 Experimental Data

To show the effectiveness of our proposed algorithms, we used two kinds of synthetic XML data. One is XMark and the other is synthesized XML data by our data generator.

XMark is a comprehensive distributed system benchmarking and optimization suite. Figure 9 depicts the structure of the XMark data. We chose the `regions` element for our target. Each `region` node contains child nodes representing world continents, like Africa, Asia, Australia, Europe, North America, and South America, as well as other child nodes. Each continent node contains several `item` elements and others, and each `item` node contains numerical measures, such as `quantity` and `payment`. We tried to compute aggregation of quantity according to the region hierarchy. We tested the following sizes of XML data:10MB, 100MB, 200MB, 300MB, 400MB, 500MB, and 1GB.

One major drawback of XMark is that it is not hierarchically deep enough as can be seen in the figure. Thus, we implemented a dedicated XML generator. It randomly generates XML data according to the predefined statistical values, such as the number of children, average depth, and number of text nodes. For our experiments, we generated the following sets of XML data.

(1) Without Mixed Structure: We generated XML data that did not have any text node being aggregated at the same level to aggregation nodes. This was done to evaluate BUA.

(2) Varying Hierarchical Levels: We generated XML data by changing the maximum depth.

(3) Varying Descendant Nodes: We generated XML data by changing the number of children for each node, thereby varying the width of XML data.

5.3 Experimental Results

Table 1 and Figure 10 show the experimental result of the TOPOLOGICAL ROLLUP operation with XMark data of depth 2. We compared TDA, BUA-Mix, SSC-Pre, and SSC-Post. Additionally, as the baseline, we computed the

Table 1 Rollup XMark With Two Hierarchical Levels

Size	10MB	100MB	200MB	300MB	400MB	500MB	1GB
AList	7	7	7	7	7	7	7
DList	2,175	21,750	43,500	65,250	87,000	108,750	222,720
SQL	62	4,128	8,065	12,289	16,735	23,421	49,142
TDA	223	2,613	5,089	7,386	9,924	12,314	32,120
BUA-Mix	229	2,685	5,087	7,410	10,038	12,655	40,690
SSC-Pre	216	2,334	4,468	6,642	8,887	10,730	27,473
SSC-Post	214	2,646	5,065	7,719	10,176	12,538	31,059

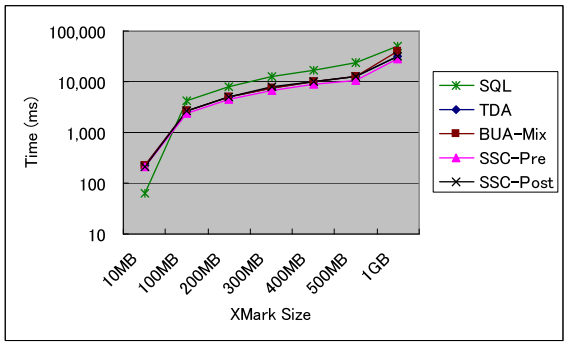

Fig. 10 Rollup XMark With Two Hierarchical Levels.

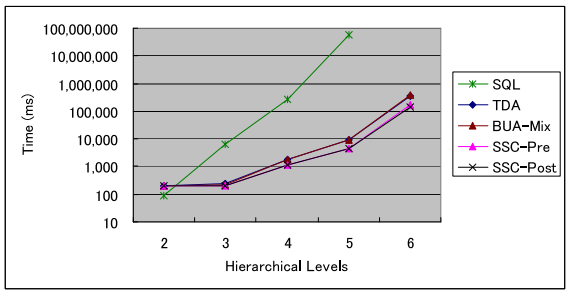

Fig. 11 Rollup Generated XML Data by Hierarchical Levels.

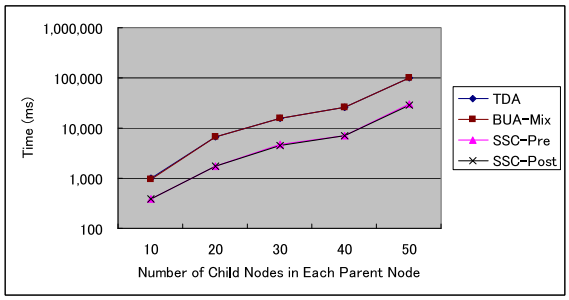

Fig. 12 Rollup Generated XML Data by Number of Child Nodes.

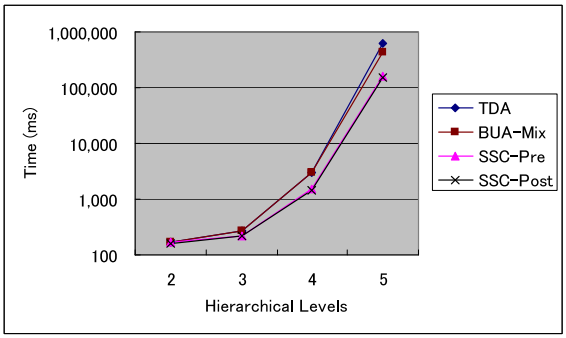

Fig. 13 Rollup Generated XML Data Without Mixed Structure.

same aggregations by SQL alone. The second and third rows of Table 1 show the length of AList and DList, respectively. For AList, it shows the total length of all ALists rather than the length of each level-AList in the cases of TDA and BUA-Mix. According to XMark data, the total number of ancestors is fixed (7 nodes), while the number of descendants increased. As a result, once the size of DList grows, SQL performs poorly compared with our proposed algorithms. Ultimately, among the algorithms, SSC-Pre and SSC-Post, because of their nature, showed similar performance but better trends compared to BUA-Mix and TDA. Note that we omit BUA because it cannot provide accurate aggregate values with a mixed structure.

Figure 11 is another experiment that shows the performance of our proposed algorithms with our generated XML data by varying the number of hierarchies. The result shows that TDA performance tendency is similar to BUA-Mix, and SSC-Pre and SSC-Post are about twice as fast as TDA and BUA. However, SSC-Post had the best performance.

Figure 12 shows the performance results from varying the number of child nodes from 10 to 50. Similar to the previous experiment, TDA and BUA showed similar trends whereas SSC-

Pre and SSC-Pos performed better than TDA and BUA. Still, SSC-Post slightly outperformed SSC-Pre.

Figure 13 shows the result for data without mixed structure, where BUA can be applied. Similar to the experiment with XMark data, we varied the number of hierarchies. Subsequently, as expected, BUA performs better than TDA. And again, SSC performed better than BUA and SSC-Post had the best performance.

In summary, the experimental results suggest that: 1) the proposed algorithms perform much better than the baseline implementation by SQL, 2) TDA and BUA have similar performance trends, but BUA performs slightly better than TDA because of its nature, and 3) the proposed algorithms SSC-Pre and especially SSC-Post perform better than TDA and BUA for all cases; specifically, SSC-Post has the best performance. As a consequence, we have the freedom to choose an appropriate algorithm from SSC-Pre and SSC-Post, taking into account how the underlying data storage of the XML database is organized.

6. GUI for XML-OLAP

This section demonstrates our XML-OLAP system with TOPOLOGICAL ROLLUP operation. The demonstrated program is produced by Java graphical user interface. As in Figure 14, the GUI shows the demonstration with three kinds of data: "Bookstore.xml", "XMark.xml", and our generated XML data. Each tab contents the demonstration with each XML data respectively. For the following detail, we will only explain our demonstration with "Bookstore.xml" since it is easier to understand.

6.1 XML Tree Viewer

XML Tree button enables the user to view "Bookstore.xml" as in Figure 15. The viewer shows the contain of "Bookstore.xml" as a tree hierarchy. The user can click on each node of the tree to see its subtree and node value. Here, the user can see what is inside XML data, such as hierarchy, property (books, magazine, author, etc.).

6.2 XQuery for "Bookstore.xml" TOPOLOGICAL ROLLUP

When the user looked at the content inside XML data, the user may think of making TOPOLOGICAL ROLLUP operation on "Bookstore.xml" by prviding XQuery with TOPOLOGICAL ROLLUP statement. Our application enables the user to input his/her XQuery by clicking on **XQuery** button. There, the user can write XQuery with TOPOLOGICAL ROLLUP extension as the input to our system.

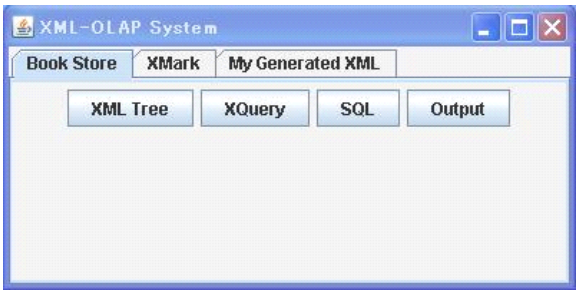

Fig. 14 XML-OLAP System Demonstration.

Figure 16 shows an example of user query which the user tries to compute the quantity of the books by categories. We can see that the user uses **"Group by TOPOLOGICAL ROLLUP("$x)"** statement to specifies to our system that he/she wants to rollup the aggregation by the proposional portion of **"$x"** path, and the portions are the subsidiary categories of the books.

6.3 TOPOLOGICAL ROLLUP Output

After the user compled his/her XQuery input, he/she can process the query by clicking on **Output**. As the result, our XML-OLAP system, by referring to the given XQuery, extracts *AList* and *DList* and runs SSC-Post join. Finally, the system produces the output which is viewed as XML tree in Figure **??**. The figure shows the quanitity of the books grouped by each category, and the last node is the quantity of the books in the whole bookstore.

7. Conclusions

This report discussed XML-LAP system followed by grouping and ROLLUP operations for XML Data, with the focus on structure-based grouping, TOPOLOGICAL ROLLUP operation. We first proposed two naive algorithms (TDA and BUA) for TOPOLOGICAL ROLLUP using the Stack Tree Join algorithm. We then showed the disadvantages in our proposed algorithms and proposed more

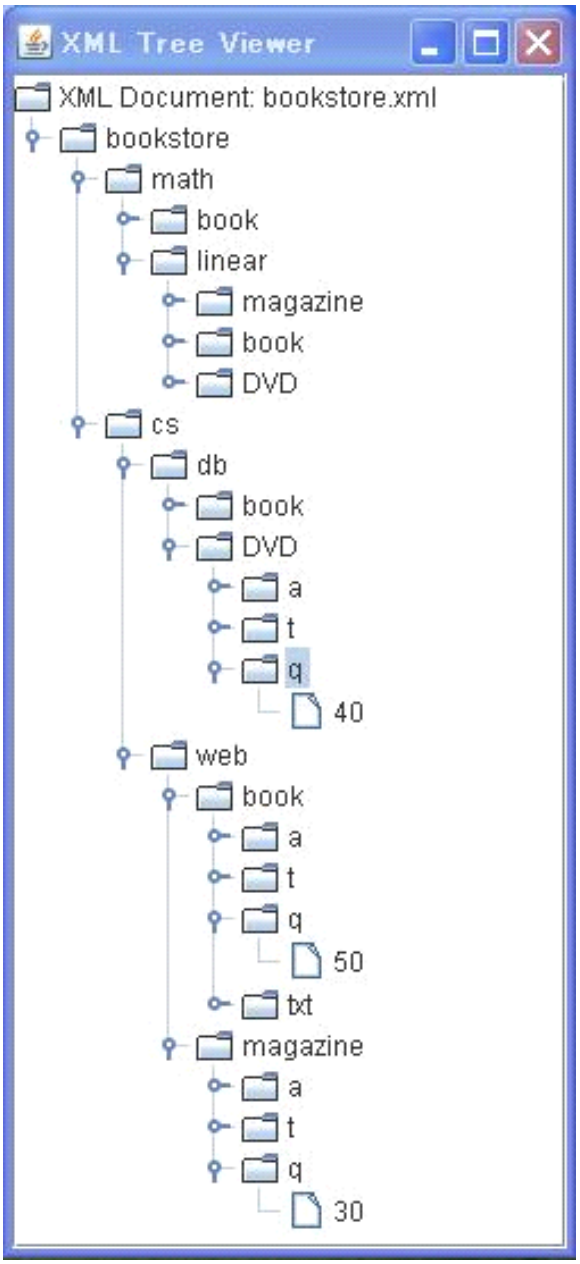

Fig. 15 "Bookstore.xml" Tree Viewer.

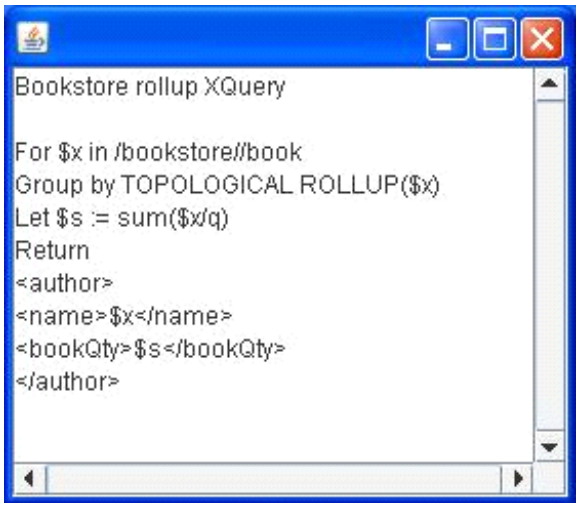

Fig. 16 XQuery of "Bookstore.xml" TOPOLOGICAL ROLLUP.

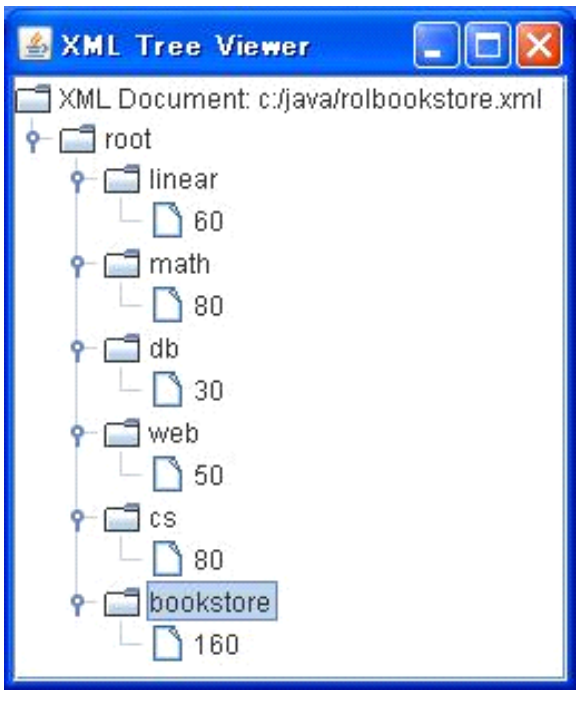

Fig. 17 Output of "Bookstore.xml" TOPOLOGICAL ROLLUP.

effective algorithms, BUC-Mix, SSC-Pre, and SSC-Post. Our experiment with a large collection of XMark data and our generated XML data showed the effectiveness of our proposed algorithms, especially SSC-Pre/Post, which enable us to effectively analyze any existing data storage of XML databases. Finally we show our GUI for XML-OLAP system where the user can process to Xquery with TOPOLOGICAL ROLLUP extension.

In the future, we plan to adopt our proposed algorithms for the remaining OLAP extension features (CUBE, WINDOWING, RANKING, ...). Furthermore, by using SSC algorithms, we can apply partitioning techniques to our algorithm to cope with very large XML data. We also plan to improve the algorithms performance by considering on multithreading with XML data partitioning. Another possibility is applying information retrieval on XML data based on our proposed algorithms.

References

1) S. Al-Khalifa, H. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu. Structural joins: A primitive for efficient xml query pattern matching. In *Proc. of ICDE 2002*, pages 141–152, March 2002.
2) R. Bordawekar and C. A. Lang. Analytical processing of xml documents: Opportunities and challenges. *SIGMOD Record*, 34(2):27–32, June 2005.
3) C. Gokhale, N. Gupta, P. Kumar, L. Lakshmanan, R.Ng, and B.Prakash. Complex group-by queries for xml. In *Proc. of ICDE 2007*, pages 646–655, April 2007.
4) M.R. Jensen, T.H. Møller, and T.B. Pedersen. Specifying olap cubes on xml data. In *Proceedings of SSDBM 2001*, pages 101–112, July 2001.
5) C. Kit, T. Amagasa, and H. Kitagawa. Olap query processing for xml data in rdbms. In *IEEE International Workshop on Database for Next Generation Researchers (SWOD)*, pages 7–12, April 2007.
6) C. Kit, T. Amagasa, and H. Kitagawa. Algorithms for efficient structure-based grouping in xml-olap. In *Proc. 10th International Conference on Information Integration and Web-based Applications and Services (iiWAS 2008)*, pages 170–177, November 2008.
7) D.Pedersen, K.Riis, and T.B. Pedersen. Xml-extended olap querying. In *Proceedings of SSDBM 2002*, pages 195–206, May 2002.
8) N. Wiwatwattana, H. Jagadish, L. V. Lakshmanan, and D.Srivastava. X^3: A cube operator for xlm olap. In *Proc. of ICDE 2007*, pages 916–925, April 2007.
9) M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. Xrel: A path-based approach to storage and retrieval of xml documents using relational databases. *ACM Trans. Internet Techn.*, 1(1):110–141, August 2001.