INFORMATION TECHNOLOGY JOURNAL

Information Technology Journal 6 (5): 697-703, 2007
ISSN 1812-5638
© 2007 Asian Network for Scientific Information

Flexible and Effective Aggregation Operator for XML Data

Hongzhi Wang, Jianzhong Li and Hong Gao
Department of Computer Science and Technology,
Harbin Institute of Technology, Harbin, China

Abstract: XML is an important format of information exchange and representation. One of its features is that it has tag representing semantics and hierarchy structure. Based on these features, an extensive aggregation of operation of XML data, XAggregation, is represented in this research. XAggregation permits XPath expression decorating dimension property and measure property. With XAggregation, statistics of XML data becomes more flexible with function of aggregating heterogeneous data and hierarchy data along some path of XML. XAggregation could be embedded in query language of XML such as XQuery. In this study the definition of XAggregation is presented, as well as the semantics and application of it. Implementation of XAggregation based on native XML database with XML as tree structure is also designed.

Key words: XML, aggregation, query processing

INTRODUCTION

XML is an important information representation format. On one hand, because of its extendibility, it is used widely as information exchange format. On the other hand, model of XML could be considered as semi-structured (Abiteboul *et al*., 2000), information representation ability of which is stronger than that of tradition relational database. XML warehouse is often used as cache of information integration system based on XML is an. XML database is also used as web database.

Today, one usage of database especial massive database is for decision support. Aggregation is a basic operation of decision support.

There are many aggregations definition and implementation methods of relational database. But in XML database, the work of aggregation is quite little. A series of work of Pedersen *et al*. (2002a-c) presents a kind of aggregation related to XML. It uses XML data as decoration of dimension properties. Tufte and Mater (2001) presented an operation to implementation aggregation of XML stream. This operation is to aggregate a series of XML documents into one based on their same parts, and do not touch the real statistics of the information contained in XML document.

Aggregation of data in XML format is different from that of relational database in semantics. With stronger representation ability of tag representing semantics, aggregation of XML can be more flexible. Possible extensions of aggregation of XML data are:

- The aggregation of the same property should permit a special set of paths, such as a path set defined with Xpath (World Wide Web Consortium, 2006). That is to say, the dimension property and measure could be a set of object in XML document. The object in various positions of XML document with various paths could be aggregated together.
- The structure of XML document should be considered, representing the complex way of aggregation. This process is quite like roll-up operation in common aggregation with the difference of multiple roll-up paths that make up of a complex structure.

With above extensions, the aggregation of information in XML format becomes flexible.

In aggregation on relational database, the property to be aggregated up is measure property and the property to be grouped is dimension property. This study continues to use these definitions to describe new aggregation. In this study, path of a node n in XML tree refers to the path from root of the tree to n. Node a nearer to root than node b is defined as a is higher than b.

In this study, a special aggregation operation based on XML, XAggregation, is presented. This operation permits Xpath decorating dimension properties and measure properties. We believe this study to be the first to consider this instance of aggregation in XML data.

Corresponding Author: Jianzhong Li, P.O. Box 750, Harbin Institute of Technology, Harbin, 150001, China
Tel: 086-0451-86415827 Fax: 086-0451-86415827

However, the implementation of this kind of aggregation is more complex, because of not only complex process of aggregation, but also various logic and storage structures of XML data.

XML database has several storage structures. Mainly, four kinds of storage for XML databases are used, including common file system, OO database, relational database and native XML database (Tian *et al.*, 2002). Native XML database is used more and more because its implementation is optimized specially for XML data. Relational database is also widely used to store and process query of XML data (Deutsch *et al.*, 1999; Shanmugasundaram *et al.*, 1999). But because of the core idea of storing XML data in relational database depends on decomposing of schema of XML data into tables. XAggregation needs travel of path, as will bring out many join operations. The efficiency is affected. Therefore, the implementation of XAggregation in this study is based on native XML database with tree structure.

This study focuses on the definition of aggregation on XML data as well as implementation of XAggregation on native XML database. The contribution of this study includes:

- Operation XAggregation are defined. The usage of XAggregation is presented by example.
- Implementation algorithm of XAggregation is presented. The implementations are based on native XML database.

MOTIVATION

Here, we consider motivation for the present study. As mentioned in introduction, the XAggregation satisfies requirements of aggregation on XML data. XAggregation serves many purposes:

Complex document structure: The object to be aggregated in the same XML document may in various positions of the XML document. For example, statistics of the salary level of a company, the organization of the company may be complex. Some parts may be vertical structure, and some another parts may be flat structure. The properties to statistics may in different path. With XAggregation, this kind of statistic could be represented and implemented.

Data distribution: Data in information integration system distributes in various autonomy data sources. All data from these data sources could be considered as one XML document distributed among multiple databases (Abiteboul *et al.*, 2000). Representation and implementation of aggregation on this kind of data need XAggregation, because properties to be aggregated may be in various paths in autonomy data sources with respective schema.

Hierarchy information aggregation: Aggregation of XML data could be along different paths. E.g. statistics of the salary level of the company with complex structure on various organizations with different granularity needs aggregation along various paths because properties to aggregate may be in different structures.

DEFINITION OF XAGGREGATION

Aggregation on XML data is distinguished to that of relational database mainly because of the tree structure of XML data. Therefore, aggregation of XML data, XAggregation, focus on data with XPath (World Wide Web Consortium, 2006), a flexible path description. Here, the definition XAggregation operation and its properties are introduced.

The definition of XAggregation: XAggregation is aggregation on measure property b, decorating by an XPath path b, grouping by dimension property k also decorating by an XPath path k. The element the aggregation based on, just like a tuple of aggregation in relational database, is defined as aggregation object with the node identifying it called common root. The result of XAggregation is the aggregation result of measure properties in aggregation object grouping on dimension property satisfying XPath description in query.

In XAggregation, both measure property and dimension property do not have a single property description as relational database but a set of descriptions. If an aggregation object has more than one measure property, the aggregation of these objects could be considered as its measure value. In the aggregation of relational database, the measure in tuples with the same dimension value could be aggregated together. While in XAggregation on XML document, under the same common root there may be more than one dimension properties with the same tag but different value or XPath. Semantics of one aggregation expression may have different meanings. Therefore restriction is to be added to XAggregation. We have chosen the following because we believe they can all be useful in different instance.

ANY_VALUE: If one aggregation object contains more than one dimension properties, this aggregation object is considered as objects each with a single dimension property value and the aggregation result of measure property values. If there are multiple dimension properties in the same aggregation object with only one value, they are considered as one dimension property during aggregation.

ANY_PATH: If one aggregation object contains more than one dimension properties, this aggregation object in a single aggregation is considered as objects each with a single dimension property value and the aggregation result of measure property value. Only when the path and value of dimension properties are both same, they are considered same during aggregation.

COMPOUND_VALUE: If one aggregation object contains more than one dimension properties, this aggregation object is considered as an object with the combination of dimension property value and the aggregation result of measure property value. Judging rule of the aggregation objects to aggregate is that they have same dimension property, as means the combinations of dimension of them are same.

COMPOUND_PATH: If one aggregation object contains more than one dimension properties, this aggregation object is considered as object with the combination of dimension property value with path and the aggregation result of measure property values. Judging criterion of the aggregation objects to aggregate together is they have same dimension property, as means the combinations of dimension of them are same, both path and value of each element of the combination should be same.

Expression of XAggregation: A simple expression of XAggregation could be described as:

$$\text{fun}([\text{path}_a]/a/[\text{path}_b]/b)\text{group by } [\text{feature}][\text{path}_a]/a/\ [\text{path}_k]\ /k$$

In the description, path$_a$, path$_b$ and path $_k$represent the paths of element a, b and k, respectively. fun is aggregation function, which may be avg, sum, count, min, max and so on. The semantics of the expression is to aggregate all the value of bs in object a satisfying path$_b$ with a unique value of k under the same object with root tag a satisfying path$_k$ and assemble the aggregate value and k value in result. Feature is one of ANY_VALUE, ANY_PATH, COMPOUND_VALUE and COMPOUND_ PATH. Default value of feature is ANY_VALUE.

The representation could be embedded into XQuery[3] easily with format:

```
for $pn in distinct-values (document (document_name)
//[path_a]/a/[path_k]/k)
let $i := document(document_name)//[path_a]/a
group by $pn

return
    <result>
            <dimension>$pn</dimension>
            <aggvalue> {fun($i/[path_b]/b)} </aggvalue>
    </result>
```

Explanation of XAggregation: It is different from data in XML format and relation is that object under XML tag may be both single value and combine object while that of relational is just a value. Hence the condition of valid aggregation is:

With aggregation function sum, avg, measure property, b, should be a simple value. With aggregation function min and max, measure property should have predefined order relation. With aggregation function count, measure property could be both simple value and combine object.

Dimension property could be both simple value and combine object. When executing aggregation, aggregation objects with the dimension properties same in both value and structure are considered to be aggregated.

Because of recursion node of schema of XML document, there may be more than one common root in a path. E.g. when aggregation is sum (a/*/b) group by a/*/d, if there are branch as a/c/a/c/b in schema, there are several instances satisfying the aggregation schema. The aggregation of this structure is computed in the following rules:

If there is more than one common root in a path, aggregation is executed along the path from lower level to higher level. Only aggregation objects under a common root could be aggregated.

The semantics of the aggregation along with path is that aggregate all the values of a measure property decorated by a special dimension property.

An example of XAggregation: Figure 2 is a XML document fragment with the schema in Fig. 1. In this XML document fragment, the fragment in gashed bound is treated as a group of aggregation object. Nowhere in what model, has aggregation resulted of these elements in this level are (10, 16) and (5, 96). In the bracket, the former number represents the value of dimension property and the latter number represents the result of aggregation. For the tag a in the root of the fragment, the results of the aggregation in four models are shown as following:

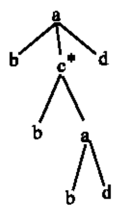

Fig. 1: Sample branch

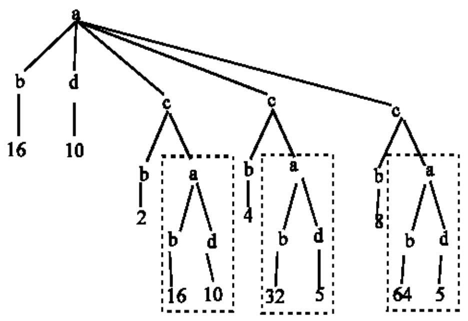

Fig. 2: An XML fragment

ANY_VALUE: (5, 142), (10,142)
ANY_PATH: (a/d, 10, 142), (a/c/a/d, 5, 142),(a/c/a/d, 10, 142)
COMPOUND_VALUE: ((10, 5), 142)
COMPOUND_PATH: (((a/d, 10), (a/c/a/d, 5), (a/c/a/d, 10)), 142)

Aggregation results of them are same because all of measure properties under root node a are considered as to be aggregated. In the four models, result representations are different. If there are upper aggregation objects of root a, in different model, different values of this level are used for aggregation.

Implementation of extended aggregation: Here, implementation algorithm of XAggregation is presented. All the implementation is based on native XML database, in which XML documents are stored in tree mode. An XML document is processed as an ordered label tree, defined as XML tree. But our algorithm does not depend on special storage structure.

The idea of XAggregation implementation is to compute aggregation result during traversing XML tree. Main data structure of the algorithm is stack_list representing aggregation nodes in the path, contains three parts: Aggregation result, dimension list, dimension-result list. Aggregation result is the aggregation result of this node, which is computed during traversing children of this aggregation node. Dimension list records values of dimension properties under this aggregation node. Dimension-result list records aggregation results and dimension values of children of this aggregation object.

XAggregation algorithm is a recursive process. When a common root is met, a new node n is pushed into a stack_list l. The value of aggregation result and dimension property is stored in n. If a measure property is met, the value of it is aggregated to all nodes in l the path to whom satisfies path condition. When a dimension property is met, the value of it is connected to the dimension-result list of all nodes in l satisfying path condition. When the computation of an aggregation is finished, the aggregation result of top in l is aggregated to the node under it and the dimension list and dimension-result list is merged into the node under it. Distinguish of the four kinds of aggregation model is that the dimension information stored in dimension list and dimension-result list is different.

Description of the algorithm is shown as follows. In order to simply the description of algorithm, the algorithm described here could only process queries with just one dimension property and one measure property. Access of the tree is considered as an interface of native XML database. Extension of the algorithm is easy.

Algorithm1 XAggregation algorithm: void Xagg (c, p_d, p_m, p_c, $path_d$, $path_m$, $path_c$, $path_r$, $path_t$, $stack_b$, func, l)

Input: r: the root of current XML sub-tree;
p_d: the name of dimension property;
p_m: the name of measure property;
p_c: the name of common root of p_m and p_c;
$path_d$: the path of dimension property in query;
$path_m$: the path of measure property;
$path_c$: the path of common root in query;
$path_r$: the path of *root* in entire XML document;
$path_t$: the path from current node to its nearest ancestor with label p_c;
$stack_b$: a lstack_list to store the value of measure properties and dimension properties along the aggregation path;
func: aggregation function

Method description

```
path'_r = path'_r+c; path_t = path_r+c; t = label_of(root);
if (p_d= =t && path_r= =path_d){// if a dimension property is met
    connect the value of this node to dimension list in the top of stack_b
    for(each node n in stack_b except top){
        if (path_r-n.path ==p_d){
```

```
        connect the value of this node to dimension list
in the top of stack_b
            }
    }
    return;
}
else if($p_m$= =t&&$path_t$= =$path_m$){// a measure property is
met
    aggregate the value of measure to result in the top of
stack_b
    for(each node n in $stack_b$ except top){
            if ($path_r$-n.path ==$p_m$){
                add the value of measure to sum in n
        }
    }
    return;
}
else if ($p_c$ = =t and $path_r$ = = $path_c$){//a common root is met
    $path_t$' = NULL;
    push a new list node n into l;
}
for (each of the child of root c){
    plain_agg(c, $p_d$, $p_m$, $p_c$, $path_d$, $path_m$, $path_c$, $path_r$',
$path_t$', $stack_b$, func);
}
if ($p_c$= =t and $path_t$= =$path_c$){//a common root is to be pop
up from $stack_b$
    stack_list_node $l_n$=the top of $stack_b$;
    output(the aggreagation value of $l_n$)
    pop $stack_b$;
    merge(top of $stack_b$, $l_n$);
}
return
```

Note of algorithm1:

- symbol '+' is overloaded for path as connect two path and symbol '-' is also overloaded. $Path_a$-$path_b$ means cutting $path_b$ as the prefix of $path_a$, if $path_b$ is not the prefix of $path_a$, NULL is returned.
- Function merge is to aggregate the aggregation results in two lists with same dimension property value. For different model, the judging condition of same at the rate of is different.
- Function output is determined by the requirement of result ' s schema. The result could be outputted as a series of tuples with position information. With the information, tuples could be assembled to result with special schema.
- If func is avg, both the result of aggregation sum and count should be recorded.

EXPERIMENTS

Here, we present an experimental evaluation of our XAggregation implementation algorithm using real and synthetic data sets. We execute our algorithm directly on original XML documents. We traverse XML document as a tree but only contain a single branch in memory.

Experimental setup:
Hardware and software environment: Our algorithm is implemented in Microsoft Visual C++ on a PIII running at 850MHZ with 128M memory RAM.

Data sets: In order to test the algorithm comprehensively, both real data set and synthetic data set are used.

Real data set we used is Shakes. The set contains 36 XML documents with size 7.31 M, 327461 nodes and 179871 elements. In order to execute the aggregation, we connect all the documents into one, shakes. xmL , with an additional root.

The query on Shakes is

```
for $pn in distinct-values (document (shakes. xmL)
//*/SPEECH/*/SPEAKER)
let $i := document(shakes.xmL)//*/SPEECH
group by $i//*/$pn
return
    <result>
            <dimension>$pn</dimension>
            <aggvalue> {count($i//*/LINE)} </aggvalue>
    </result>
```

This query has special semantics as Astatistics how many line each speaker has said in all Shakes plays@.

The schema of real data is simple and the number of measure properties and dimension properties is small. In order to further test our algorithm, synthetic data sets are designed with fixed schema using XMLgenerator (Diaz and Lovell, 1999). Two data sets, named Set B and Set C, are generated with different schema as are presented in Fig. 3. XML documents with various size and structure are generated.

The core task of Set B is to test the relationship between file size, number of nodes and process efficiency. The core task of Set C is to test the relation ship between the structure and process efficiency, especially when the recursive common root exists.

```
(a)
<!ELEMENT root (a*, f*)>
<!ELEMENT f (a*, b*, c*)>
<!ELEMENT d (a, f*, c)>
<!ELEMENT a (#PCDATA)>
<!ELEMENT b (#PCDATA)>
>!ELEMENT a (#PCDATA)>
```

```
(b)
<!ELEMENT root (a*, f*)>
<!ELEMENT f (a*, b, d, c*)>
<!ELEMENT d (a, f*, c)>
<!ELEMENT a (#PCDATA)>
<!ELEMENT b (#PCDATA)>
>!ELEMENT a (#PCDATA)>
```

Fig. 3: (a) DTD of set B, (b) DTD of set C

For compare, the query on the two synthetic is the same:

```
for $pn in distinct-values(document("b.xml")//*/f/*/c)
let $i := document("b.xml")//*/f
group by $i//*/$pn
return
    <result>
            <dimension>$pn</dimension>
            <aggvalue> {sum($i//*/a)} </aggvalue>
    </result>
```

RESULTS

From Table 1, it is noted that in Set B, the growth rate of speed of data process is faster along with the file size. Although the comparison of string is slower than that of number, the process speed of Shakes is faster than Set B, because there are more nodes with no relation with aggregation in Shakes than those of XML documents in Set B. Our algorithm is related to not only document size but also number of nodes of common root, measure properties and dimension properties.

In Table 2, the file size and number of elements of group 2 and group 3, group 4 and group 5 is similar but the query process efficiency is quite different. It is because in our algorithm, the deeper a measure property is, the more add operation should be executed. The query process speed gap between group 4 and group 5 is larger than that between group 2 and group 3, because the structure of these XML documents becomes different when adjusting parameters to obtain fit document size. It could be concluded that our algorithm is sensitivity to the structures of XML documents, even when they have same schema.

Comparing Table 1 with Table 2, query process efficiency of Set C is higher than that of Set B, when XML tree is not too high. The reason is that the number of measure properties and dimension properties in the documents of Set C in less than that of Set B, but the number of common roots is contrast. That is to say, the affect of number of measure properties and dimension properties is more than that of number of common root and level. But the height of XML tree has large affect on the efficiency of query process.

Table 1: Comparison of experiment result of shakes and set B

Data set	File size	No. of nodes	No. of elements	Process time (s)
Shakes	7.31M	327461	179871	9.713
Set B	64.1K	6862	3501	0.16
Set B	628K	66262	33201	15.321
Set B	1.63M	174442	87381	87.355
Set B	17.7M	1863252	933191	1700.354

Table 2: Comparison of experiment result of set C

Group id	File size	No. of nodes	No. of elements	Maximum No. of level	Process time (s)
1	11.3k	1181	748	8	0.110
2	646K	65549	41518	15	9.063
3	658K	67235	42277	8	6.178
4	2.39M	234785	203977	8	16.924
5	2.81M	284250	199277	11	104.74
6	17.4M	1771441	1121918	21	2087.501

DISCUSSION

As an important part of query, expression of group and aggregation are defined in some of query languages. In XML query languages, LORLE and XML-GL aggregate functions are fully implemented, XSL and XQL implementation aggregation partly, XML-QL does not support aggregation (Bonifati and Ceri, 2000). Xquery (World Wide Web Consortium, 2006) considers group and aggregation. In query language with aggregation, the properties are permitted to be decorated by path information. But none of them considers the complex instance when path of property is described using complex path expression such as XPath, although XQuery uses XPath as its path description standard.

Some algebra for XML also defines group and aggregation (Beech *et al*., 1999; Fankhauser *et al*., 2001; Galanis *et al*., 2001). But none of the aggregation definitions could represent the instance of recursion.

Process query with XPath is hot in research of XML (Shim *et al*., 2002; Grust, 2002; Gottlob *et al*., 2002; Chien *et al*., 2002). Most of them focus on selection and projection of XPath. These works could be used for effective implementation of XAggregation on special storage structure. However, none of them relates to aggregation directly.

CONCLUSIONS

In this study, XAggregation, an operation of aggregation on XML data is presented. It is flexible for aggregation of objects in XML document decorating with XPath. It could be used in statistics of XML documents with complex structure, such as recursion properties or distributed in various autonomy sites in internet. XAggregation could be embedded into Xquery. Implementation algorithm is presented in this study.

Our algorithm is based on native XML database stored as a tree but not on special storage. Efficiency of our algorithm is not high. It is necessary to design special algorithms of special storage. In distribution environment, information transmission time should be considered. These problems are left for further work.

ACKNOWLEDGMENT

This study is supported by the key Natural Science Foundation of Heilongjiang Province under Grant No. Zjg03-05, the National Natural Science Foundation of China under Grant No. 60473075, NCET under Grant No. NCEF-05-0333 and the National Grand Fundamental Research 973 Program of China under Grant No.2006CB303000.

REFERENCES

Abiteboul, S., P. Buneman and D. Suciu, 2000. Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, 2000.

Beech, D.A. Malhotra and M. Rys, 1999. A formal data model and algebra for XML. Communication to the W3C.

Bonifati A. and S. Ceri, 2000. Comparative Analysis of Five XML Query Languages, ACMSIGMOD Record 29 (1).

Chien, S.Y., Z. Vagena, D. Zhang and V.J. Tsotras, 2002. Efficient structural joins on indexed XML documents. Proceedings of 28th VLDB conference 2002.

Deutsch, A., M.F. Fernandez and D. Suciu, 1999, Storing Semi-structured Data with STORED, SIGMOD, 1999.

Diaz, A.L. and D. Lovell, 1999. XML Generator, http://www.alphaworks.ibm.com/tech/xmlgenerator.

Fankhauser, P., M. Fernandez, A. Malhotra, M. Rys, J. Simeon and P. Wadler, 2001. The XML query algebra. http://www.w3.org/TR/2001/WD-Query-algebra-20010215.

Galanis, L., E. Viglas, D.J. DeWitt, J.F. Naughton and D. Maier, 2001. Following the paths of XML data: An algebraic framework for XML query evaluation. 2001. Available at http://www.cs.wisc. edu/niagara/ papers/algebra.pdf.

Gottlob, G., C. Koch and R. Pichler, 2002. Efficient algorithms for processing XPath queries. Proceedings of 28th VLDB, 2002.

Grust, T., 2002. Accelerating XPath location steps. Proceedings of ACM SIGMOD2002.

Pedersen, D., K. Riis and T.B. Pedersen, 2002a. XML-extended OLAP querying. 14th International Conference on Scientific and Statistical Database Management (SSDBM'02).

Pedersen, D., K. Riis and T.B. Pedersen, 2002b. Cost modeling and estimation for OLAP-XML federations. In Proceedings of DaWaK2002, pp: 245--254.

Pedersen, D., K. Riis and T.B. Pedersen, 2002c. Query optimization for OLAP-XML federations. In Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP.

Shanmugasundaram, J., K. Tufte, C. Zhang, G. He, D.J. DeWitt and J.F. Naughton, 1999. Relational databases for querying XML documents: Limitations and Opportunities. VLDB 1999.

Shim, K., C. Chung and J. Min, 2002. APEX: An Adaptive Path Index for XML data. Procedings of ACM SIGMOD 2002.

Tian, F., J. David DeWitt, J. Chen and Chun Zhang, 2002 The design and performance evaluation of alternative XML storage strategies. SIGMOD record special issue on Data Management Issues in E-commerce, March 2002.

Tufte, K. and D. Mater, 2001. Aggregation and accumulation of XML data. IEEE Data Eng. Bull., 24: 34-39.

World Wide Web Consortium, 2006. XML Path Language (XPath) 2.0. http://www.w3.org/TR/ xpath20/.

World Wide Web Consortium, 2006. XQuery 1.0: An XML query language. http://www.w3.org/TR/ xquery/.