JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 26, 659-672 (2010)

Processing Strategy for Global XQuery Queries Based on XQuery Join Cost

JONG-HYUN PARK AND JI-HOON KANG*
Department of Computer Science and Engineering
Chungnam National University
Daejeon, 305-764 South Korea

XML is a standard for exchanging and formatting data over the Internet and XQuery is a standard query language for searching and integrating XML data. Therefore, it is a natural choice for interoperability to use XQuery over the Internet. Global XQuery queries search and integrate heterogeneous data, being distributed in the local systems.

In order to process efficiently global XQuery queries, their processing strategy is important because an improper processing strategy could produce an enormous number of intermediate results or execute redundant expressions. In distributed relational databases, there are some techniques for processing global SQL queries. Unfortunately, however, the structure of the data handled by the XQuery language is quite different from the one by the SQL. The XQuery language deals with semi-structural data, *i.e.* tree-structured data, while SQL deals with well-structured data, *i.e.*, the table-shaped data. These structural differences make it difficult to apply the techniques for global SQL queries into for global XQuery queries. Especially this paper considers the join cost for devising a query processing strategy. Therefore, we define some problems for estimating the join cost in XQuery queries and propose ECNJ algorithm for solving these problems. Also this paper proposes the query processing strategy and evaluates the strategy by implementing a prototype system.

Keywords: global XQuery processing, estimating join cost, XQuery, join cost, XQuery processing strategy

1. INTRODUCTION

XML has emerged as a leading language for representing and exchanging data over the Internet. In recent years, there have been a number of researches focusing on the integration of XML data and heterogeneous data including relational data [1, 2]. One of the methods for integration is to make non-XML data to be considered as XML data by using XML views [2-4]. The global XML view integrates the local XML views, and thus users can see and search the distributed heterogeneous data via the global XML view. At this time, one of the standard query languages for searching the integrated heterogeneous data is XQuery language [5].

The users can consider only the global XML view and can give queries in XQuery and then get results, but they do not need to know either where the data is or how the data is shaped. Such queries are just global XQuery queries over the integrated heterogeneous data.

In order to process efficiently global XQuery queries, it is important to decide a processing strategy of the queries. Global queries consist of some local queries for sending to

Received March 11, 2008; revised May 19, 2008; accepted August 14, 2008.
Communicated by Chih-Ping Chu.
* Corresponding author.

each local system. Therefore, to map out strategies for processing the local queries affects the total processing time of global queries. Especially, global XQuery queries naturally contain value-based join operations among local systems such as "=" and "eq". Since the join operations are expensive for processing a query, its processing strategy is very important for efficient processing of global XQuery queries. Therefore, there are some studies on the efficient processing of join operations and one of these studies is that select a processing strategy with minimum join cost by estimating the join cost. In case of SQL, there are already some researches for estimating the join cost of global SQL queries. However, we cannot apply the methods for estimating the cost of join operations in SQL queries into XQuery queries without modification because of the structural difference between relational data and XML data. Contributions of this paper are threefold. The first, we define some problems to apply the method for estimating the join cost of SQL queries into for XQuery queries. The second, we propose an algorithm for estimating the join cost described in global XQuery queries and processing strategy of the queries based on our algorithm. Three, we implement a prototype for our strategy and evaluate.

The rest of the paper is organized as follows. In section 2, we present related works. Section 3 the Join Cost-Based Strategy. In section 4, we evaluate our strategy. Finally, we summarize the contributions of this paper and indicate some directions for future research work in section 5.

2. RELATED WORK

The destination of global queries is distributed local systems and a lot of global queries contain operations for joining among local systems. [6-8] have proposed methods for efficient processing global SQL queries in distributed environments. They basically consider the structure of the global SQL queries. One of these researches is that estimates the join cost and selects the join order with minimum cost because join operations need high cost for its processing [8, 9]. The processing method for global XQuery queries also is similar to the case of the global SQL queries. [6-8] have proposed a cost model for estimating the join cost in SQL queries and a method for selecting the join order. Their approaches refer to statistics from relational view for estimating optimal join order. That idea can be referred for XQuery queries because XQuery query expressions borrow features from SQL [5]. For example, a primary key in table is similar to 'id' attribute in XML document. However, we can not fully accept their cost model for processing the XQuery queries because of the structural difference between SQL and XQuery. Therefore, this paper describes about what is the difference between SQL and XQuery. Also the paper proposes strategy for processing global XQuery queries by referring to the techniques for processing the global SQL queries. Of course, communication cost over distributed environment also affects the query processing time [6, 7, 9, 10]. However, this paper focus on the join cost.

In research field on XML query processing, there are some studies for efficient processing of join operations in XML queries [11, 12]. Especially, they focus on the processing cost of XPath expression, I/O cost, and the execution time of operations *etc*. However, their approaches are useful in only native XML systems and are not general methods. Thus, their methods depend on the storing method or indexing technique of a

specific application. Therefore, we cannot accept previous techniques for the goal of this paper. Our approach can be used in any XQuery engine and in any middleware for processing global XQuery queries because we only uses the given input query instances and query views for deciding the processing strategy.

3. JOIN COST-BASED STRATEGY

It is difficult to decide the join order amongst the local queries by using the global query instances only. Therefore, by using local views this paper proposes a method for estimating the join cost and deciding the join order based on the estimated cost.

3.1 Considerations

The XML view can be expressed in various languages like XML DTD, XML Schema, and XQuery. The XML view contains the information about the structure of stored data, occurrence information, cardinality, *etc*. In the case of SQL, the cardinality of the attributes described in the view is used to estimate the join cost [8, 14]. Therefore, this paper also assumes that we can obtain the cardinality information of each node stored from the local XML views. Of course, we can use the Count() Function which is defined in XQuery Functions and Operations [15], if the local view does not serve the cardinality information. However, this paper is not interested in estimating the cardinality even if there are already some researches about it in [16, 17].

Fig. 1 shows the three views which describe the structure of data stored in the three local systems. The labels beside each node express the pair of occurrence and cardinality. For example, the label (?, 4) for the '*b*1' node expresses that the '*b*1' node can occur zero or one time in the '*b*' node as a child node and the total number of the '*b*1' node stored in the local system is four. Table 1 is a sample XQuery query written toward the three local views in Fig. 1. This query contains two join operations between the 'A' system and 'B' system and between the 'B' system and 'C' system.

Fig. 2 shows the join relationships among the three local systems which are target systems for the query in Table 1. The first join operation joins the value of the '*a*2' node in the 'A' system with the value of the '*b*2' node in the 'B' system. The second join operation joins the value of the '*b*3' node in the 'B' system with the value of the '*c*1' node in the 'C' system. Fig. 3 shows the three kinds of possible strategies for processing the query in Table 1. Fist case is that processes the join operation between the A and B

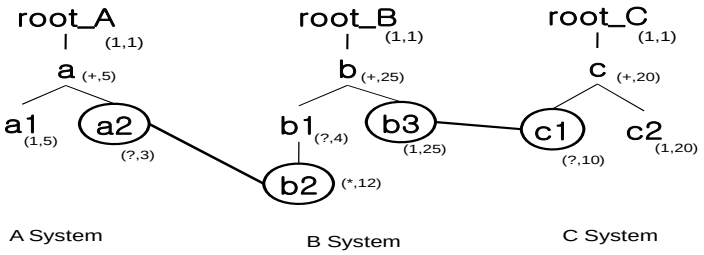

Fig. 1. Sample XML views.

Table 1. Sample global XQuery query written toward three views in Fig. 1.

```
for $A in doc("A.xml")/root_A/a
for $B in doc("B.xml")/root_B/b
for $C in doc("C.xml")/root_C/c
for $A2 in $A/a2
for $B2 in $B/b1/b2
for $B3 in $B/b3
for $C1 in $C/c1
where $A2/text() = $B2/text() and $B3/text() = $C1/text()
return
   <result>{$A/a1, $C/c2}</result>
```

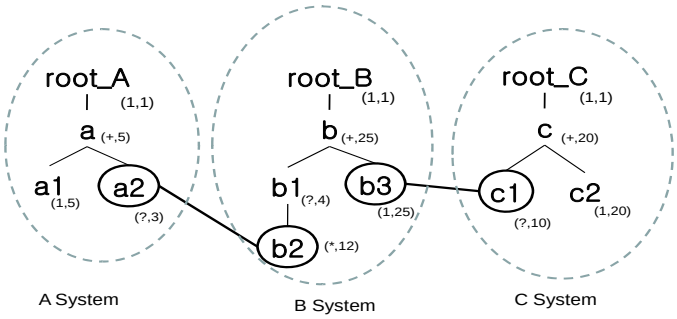

Fig. 2. The join relationships of the query in Table 1.

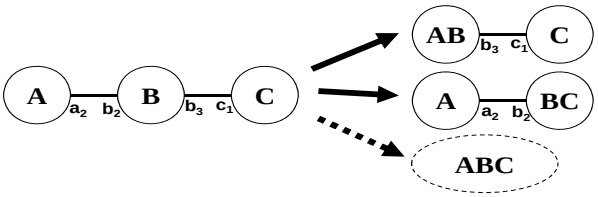

Fig. 3. The possible strategies for processing a query with two join operations.

systems and then joins the C system, second case is that joins between the B and C systems and then joins the A system, and last case processes all join operations in single system. However, third case is not in our consideration because our goal is that decides the join order. Consequently, the number of possible strategies can be calculated by the permutation formula in the statistics [13]. Thus the number of possible strategies depends on the number of the join operations. If the number of join operations is *n*, then the number of possible strategies is ${}_nP_n$ and finally is calculated by *n*! [13]. In the case of our example query, the number of possible strategies can be calculated by ${}_2P_2$ because the query contains two join operations, of course, when except the third case.

In order to select the optimal join order, one of the most popular techniques is to estimate the join cost of all possible processing methods and then select a method with the minimal join cost [14, 18]. This paper calculates the join cost by estimating the number

of comparison times. For example, in the case of SQL, if the Fig. 3 is a join graph for an SQL query, then the Table 'B' joins with the other two tables 'A' and 'C'. If the join operation with the Table 'A' is first executed, then the '*b*3' attribute for the next join operation is the filtered '*b*3' attribute by the first join operation. Thus, the former join operation filters out the operands for latter join operation. Therefore, we have to consider how much to be filtered by the formal join operations. The following is an expression for estimating the previous example query:

- Join Cost of $A \bowtie B$ = $|A.a2 \bowtie B.b2|$ = Join Selectivity($A.a2$, $B.b2$) × $|A.a2|$ × $|B.b2|$
- Join Cost of $B \bowtie C$ = $|B'.b3 \bowtie C.c1|$ = Join Selectivity($B'.b3$, $C.c1$) × $|B'.b3|$ × $|C.c1|$ (B' is the filtered B).

Now we can apply the above expression from the sample XQuery query in Table 2. To apply the $|A.a2 \bowtie B.b2|$, we must know the join selectivity of the two attributes. In the case of SQL queries, Join Selectivity can be calculated using three different kinds of expressions in Table 2 and these expressions can be similarly applied for the XQuery queries. For example, |$B/@id| expresses the total number of attribute "id" of node "B".

Table 2. Sample global XQuery query written toward three views in Fig. 1.

Conditions	SQL	XQuery
One operand is ID (key)	A.value = B.key → 1/|B.key|	$A/text() = $B/@id → 1/\|$B/@id|
Two operands are ID (key)	A.key = B.key → 1/Max(|A.key|, |B.key|)	$A/@id = $B/@id → 1/Max(\|$A/@id|, $B/@id|)
Two operands are not ID (key)	A.Value = B.Value → 1/Max(|A.key|, |B.key|)	$A/text() = $B/text() → 1/Max(\|$A|, |$B|)

For the views in Fig. 2, we can estimate the Join Selectivity of the $|A.a2 \bowtie B.b2|$ as following that:

- Join Selectivity($A.a2$, $B.b2$) = 1/Max($|A.a2|$, $|B.b2|$) = 1/$|B.b2|$ = 1/12

Finally, the join cost of $|A.a2 \bowtie B.b2|$ can be estimated as following that;

- $|A.a2 \bowtie B.b2|$ = Join Selectivity($A.a2$, $B.b2$) × $|A.a2|$ × $|B.b2|$ = 1/12 × 3 × 12 = 3

Thus, we can estimate that the $|B'.b2|$ related with $|B'.b3|$ is 3 among the total of 12. Therefore, the selectivity of $B.b2$ is 3/12 and we can express that by *Sel*($B.b2$) in this paper. Even if we can estimate the $|B'.b2|$ by the above method, we have to know the $|B'.b3|$ for estimating the cost for next join operation. However, in the case of XQuery queries, there are some problems for estimating the $|B'.b3|$ by using the $|B'.b2|$. The reason is that the mapping relationship between the XML nodes is not fixed by the one-to-one mapping.

Fig. 4 shows that the mapping relationship for relational data is only one-to-one and the mapping relationship for XML data is one of the sixteen cases; one-to-one (1:1, 1:?, ?:

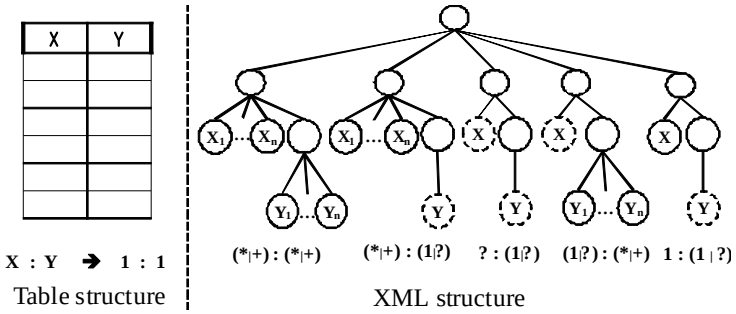

Fig. 4. The each relationship between attributes in relational table and nodes in XML document.

1, ?:?), one-to-many (1:*, 1:+, ?:*, ?:+), many-to-one (*:1, *:?, +:1, +:?), and many-to-many (*:*, +:*, *:+, +:+). The '*', '+', and '?' symbols represent that their child element occurrences are zero or more, one or more, and zero or one respectively. If the above example query is an SQL query, then we can estimate the $|B'.b3|$ by 3 because the $|B'.b2|$ is 3 and the mapping relationship between the '*b*2' and '*b*3' attributes is one-to-one [19]. However, in the case of XQuery, we have to select the mapping relationship between the '*b*2' and '*b*3' nodes. Therefore, this paper proposes the ECNJ (Estimating Cardinality for Next Join) algorithm for estimating the cost of the next join operation.

3.2 Estimating the Join Cost

The 'B' local system stores the '*b*2' node and the '*b*3' node for joining the 'A' local system and the 'C' local system, respectively. Fig. 5 shows the relationship between these two nodes and the direction of the arrow shows the procedure of the ECNJ algorithm:

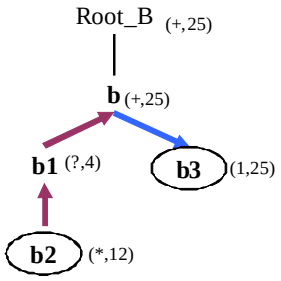

Fig. 5. Procedure of ECNJ algorithm.

In the previous section, we have estimated the selectivity for the '*b*2' node. However, it is impossible to estimate the selectivity of the '*b*3' node from the selectivity of the '*b*2' node because the '*b*2' node is not directly linked to the '*b*3' node. Thus, in order to estimate the selectivity of the '*b*3' node, we have to find the shortest path between the two nodes as shown by the direction of the arrow in Fig. 5, as well as esti-mate the selectivity between each pair nodes.

We classify the two kinds of relationship cases between the two nodes for applying the ECNJ algorithm. One is the parent-to-child which estimates the selectivity of the

child node by using the selectivity of a parent node and another is the child-to-parent which is opposite to the parent-to-child. In Fig. 5, it is one of the child-to-parent cases to estimate the selectivity of the '*b*1' node by using the selectivity of the '*b*2' node. Also the case which estimates the selectivity of the '*b*3' node by using the selectivity of the '*b*' node is one of the parent-to-child cases.

ECNJ (Estimating Cardinality for Next Join) Algorithm

Definition 1 is used to estimate the cardinality of child nodes by using the selectivity of a parent node (parent-to-child case). For Definition 1, this paper assumes that every parent node has the same number of child nodes when the occurrence character for child nodes is '+' or '*'.

Definition 1

PN: Parent Node
CN: Child Node
Sel(*N*): Selectivity of *N* Node
$|CN'| = \{|CN| \times \text{Sel}(PN)$ If Occurrence of $PN{:}CN = 1{:}(1 \mid ? \mid * \mid +)\}$

In an XML tree, the parent node for every node is only one except the root node. Therefore, in the parent-to-child case, the occurrence character of a parent node is always '1' and the occurrence character of a child node is '1', '?', '+', or '*'. Finally, the case for parent-child relationships is one of the four cases 1:(1 | ? | * | +). If the parent-to-child relationship is 1:1 or 1:?, then we can directly apply the selectivity of the parent node for the child node. However, although the parent-to-child relationship is 1:+ or 1:* we can apply the selectivity of the parent node into the selectivity of the child node because of the assumption of this paper.

Fig. 6 depicts a part of the view in Fig. 2 and the structure of "*b*1" and "*b*2" nodes stored in the 'B' system. As described in the view, the number of the stored '*b*2' node is 12 and the '*b*1' node is 4. Therefore, we can predict that every '*b*1' node has three '*b*2' nodes as its child node. If $b1_1$ and $b1_2$ among $b1_1$, $b1_2$, $b1_3$, and $b1_4$ nodes are filtered by the previous join operation, the selectivity of the parent node '*b*1' is 1/2. Thus, by Definition 1, we can estimate that the cardinality of the '*b*2' node required for the next join operations is $12 \times 1/2 = 6$.

Definition 2 is used to estimate the cardinality of the parent nodes for next join operation by using the selectivity of the child node. This paper classifies the two cases for estimating the cardinality of parent nodes. The first case is that the occurrence characters of child node vs. parent node are 1:1 or 1:?. In this case, a parent node has only one child node or no child node. Therefore, the cardinality of the parent node for the next join operation is the same as or less than the number of child nodes. The second case is that the occurrence characters of child nodes vs. parent nodes are 1:+ or 1:*. For example, if three '*b*2' nodes are selected among twelve '*b*2' nodes in Fig. 6, the possible number of parent nodes is one, two, or three nodes. For instance, if the '$b2_1$', '$b2_2$', and '$b2_3$' nodes are selected, then the '$b1_1$' node will be selected as its parent node. And if the '$b2_1$', '$b2_2$', and '$b2_4$' nodes are selected, then the '$b1_1$' and '$b1_2$' nodes will be selected. Also if the '$b2_1$', '$b2_4$', and '$b2_7$' nodes are selected, then the '$b1_1$', '$b1_2$' and '$b1_3$' nodes will be

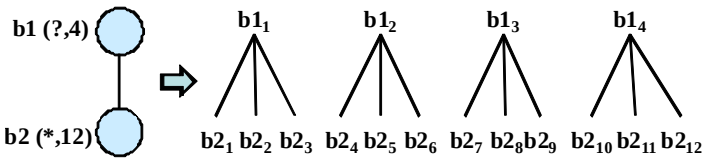

Fig. 6. The partial view and the structure of stored XML data.

Definition 2

$m = \{|CN|/|PN|\}$, The average number of child nodes

Min$P = \{\lceil |CN'|/m \rceil\}$, The minimum value of the number of selectable parent nodes

Max$P = \{\text{Min}(|PN|, |CN'|)$, The maximum value of the number of selectable parent nodes

$$|PN'| = \left\{ \begin{array}{ll} |CN'| & \text{If Occurrence of } CN{:}PN = (1 \mid ?){:}1 \\ \sum_{i=\text{Min}P}^{\text{Max}P} i \cdot P(i) & \text{Else Occurrence of } CN{:}PN = (+ \mid *){:}1 \end{array} \right\}$$

$$P(i) = \frac{\left({}_{i \cdot m}C_{|CN'|} - \sum_{j=\text{Min}P}^{i-1} E(j, i) \right) \times {}_{|PN|}C_i}{{}_{|CN|}C_{|CN'|}},$$ Probability when the number of selected parent nodes is *i*.

$$E(j, i) = \left({}_{j \cdot m}C_{|CN'|} - \sum_{l=\text{Min}P}^{j-1} E(l, j) \right) \times {}_{i}C_j$$

selected. Therefore, for estimating the cardinality of parent nodes, this paper estimates the number of parent nodes in all possible cases and then calculates their average.

In order to estimate the cardinality, the first step is to calculate the range of the number of selectable parent nodes and then estimate the probabilities for the selection of each number in the range. The Definition 2 shows the formula for this calculation. In the Definition 2, the Min*P* is the minimum number of selectable parent nodes and the Max*P* is the maximum number. For example, if three '*b*2' nodes are selected among twelve '*b*2' nodes in Fig. 6, the range of the number of selectable parent nodes '*b*1' is from one to three. Therefore, Min*P* is one and Max*P* is three. If four '*b*2' nodes are selected, its range is from two to four. Thus, the Min*P* is the number of child nodes selected by the former join operation divided by the average number of child nodes. Also, the Max*P* is a smaller one between the number of selected child nodes and the number of parent nodes. For example, if we select three '*b*2' nodes among twelve '*b*2' nodes, then the number of selectable parent nodes is one, two or three and the number of combinations of selectable child nodes can be calculated by ${}_{12}C_3$. The second step for estimating the cardinality of parent nodes is to calculate the probability of selected parent nodes. To estimate the cardinality of parent nodes in the previous example, we have to calculate the probability for possible three cases and average of the three probabilities because the number of selectable parent nodes is one, two or three. When the number of selected parent nodes is one, the selected child node set is one of these sets $\{b2_1, b2_2, b2_3\}$, $\{b2_4, b2_5, b2_6\}$, $\{b2_7, b2_8,$

$b2_9$}, or {$b2_{10}$, $b2_{11}$, $b2_{12}$}. Thus, the number of cases for combining three nodes among '$b2_1$', '$b2_2$', and '$b2_3$' can be calculated by ${}_3C_3$; and the number of cases for combining one parent node among the total parent nodes can be calculated by ${}_4C_1$. Therefore, in Fig. 6, the number of cases for selecting one parent node is calculated by ${}_3C_3 \times {}_4C_1$ when three child nodes are selected. Also, its probability $P(1)$ is calculated by $({}_3C_3 \times {}_4C_1)/{}_{12}C_3$. The second case to calculate the probability is when the number of selected parent nodes is two. If the selected three nodes are between $b2_1$ and $b2_6$, its parent nodes are the $b1_1$ and $b1_2$ nodes. Thus, we can use the ${}_6C_3$ to calculate the number of cases for combining the three nodes between the $b2_1$ node and $b2_6$ node. However, the ${}_6C_3$ includes the number of cases when the number of selected parent nodes is one. Therefore, we have to subtract the previous calculated number from the current number. Finally, the expression for calculating the number of cases for combining three nodes among six nodes is ${}_6C_3 - ({}_3C_3 \times {}_2C_1)$. Also, the number of cases for combining two parent nodes among the total parent nodes can be calculated by ${}_4C_2$. Thus, probability $P(2)$ is calculated by $(({}_6C_3 - ({}_3C_3 \times {}_2C_1)) \times {}_4C_2)/{}_{12}C_3$. The third case to calculate the probability is when the number of selected parent nodes is three. The calculation for third case is also similar to the previous two calculations and probability $P(3)$ is calculated by $(({}_9C_3 - ({}_3C_3 \times {}_3C_1 + ({}_6C_3 - {}_3C_3 \times {}_2C_1) \times {}_3C_2)) \times {}_4C_3)/{}_{12}C_3$.

In the case for the sample XML document in Fig. 6, if the three child nodes are selected, each probability for selecting parent nodes follows that:

$$P(1) = ({}_3C_3 \times {}_4C_1)/{}_{12}C_3 = 4/220 = 1/55$$
$$P(2) = (({}_6C_3 - {}_3C_3 \times {}_2C_1) \times {}_4C_2)/{}_{12}C_3 = 108/220 = 27/55$$
$$P(3) = (({}_9C_3 - ({}_3C_3 \times {}_3C_1 + ({}_6C_3 - {}_3C_3 \times {}_2C_1) \times {}_3C_2)) \times {}_4C_3)/{}_{12}C_3 = 108/220 = 27/55$$

Therefore, the cardinality $|b1'|$ required for next join operation is estimated by the following expression:

$$|PN'| = \sum_{i=\mathrm{Min}P}^{\mathrm{Max}P} i \cdot P(i) = 1 \times \frac{1}{55} + 2 \times \frac{27}{55} + 3 \times \frac{27}{55} \approx 2.5.$$

Finally, $Sel(b1)$ is 2.5/4 which is estimated from the selectivity of the '$b2$' node. The next step for estimating the selectivity of the '$b3$' node in Fig. 5 is to estimate the cardinality of the 'b' node by using the selectivity of the '$b1$' node. The occurrence relationship between the '$b1$' and 'b' node is '?:1'. Therefore, we can estimate that $|b|$ is 2.5 and $Sel(b)$ is 2.5/25 by the Definition 1. The last step is to estimate the number of the '$b3$' node by using the $Sel(b)$. The occurrence relationship between the 'b' and '$b3$' node is '1:1' and this case is in the Definition 2. Therefore, $|b3|$ is estimated by $25 \times (2.5/25)$.

4. PERFORMANCE EVALUATION

In order to evaluate whether our join cost-based strategy for processing global XQuery queries is relevant, we consider five local systems which use single hub in LAN. The hardware and software of all local systems are the same following that the CPU has an Intel(R) Pentium(R) M processor 1.6GHz, the memory size is 1.0GB, and the OS is

Windows XP. The CPU for the global XQuery query processor is Intel(R) Pentium(R) 4 processor 3.2GHz and the memory size is 2.0GB. The XQuery engine for the local systems and the global query processor is Saxon Beta Version [20]. The Saxon Beta Version is free and one of the most popular XQuery engines.

Fig. 7 shows five local XML views which describe the information of local data. Table 3 describes the join structure of three sample XQuery queries which are written toward local views in Fig. 7. The destinations of XQ1 are three local systems which store "Closed_Auctions", "People", and "Open Auctions" data in XML respectively. The XQ1 contains two join operations between the 'C' local system and the 'P' local system and between the 'P' local system and the 'O' local system. The destinations of XQ2 are four local systems including "ItemsOfAmerica" data and the XQ2 contains three join operations between the local systems. The XQ3 is written toward five local systems including "Categories" data and contains four join operations. For evaluation, this paper increases the number of join operations between local systems. The reason is that evaluates the dependency between our strategy and the number of join operations. Also, we increase the size of sample data from 5MB to 25MB by 5MB increments for evaluating the scalability property of our strategy.

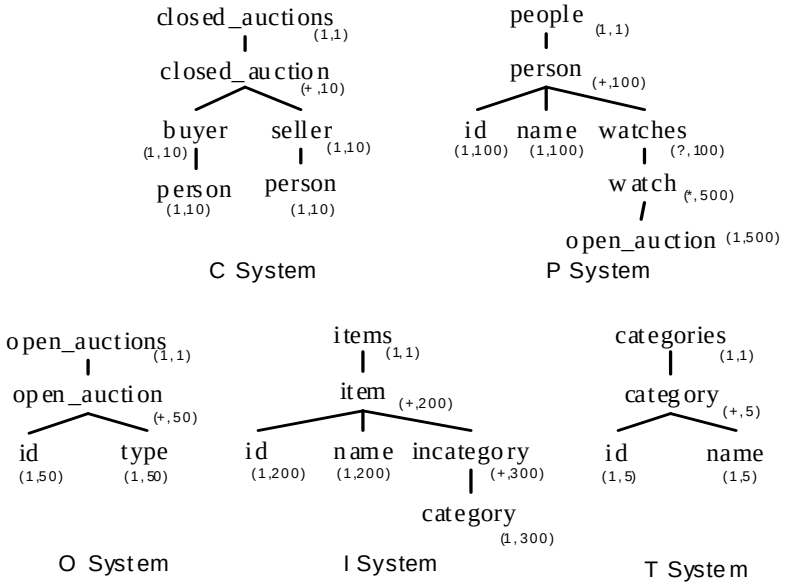

Fig. 7. Five local XML views.

Table 3. The structure of sample XQuery queries for evaluation.

XQ1	C — P — O
XQ2	C — P — O — I
XQ3	C — P — O — I — T

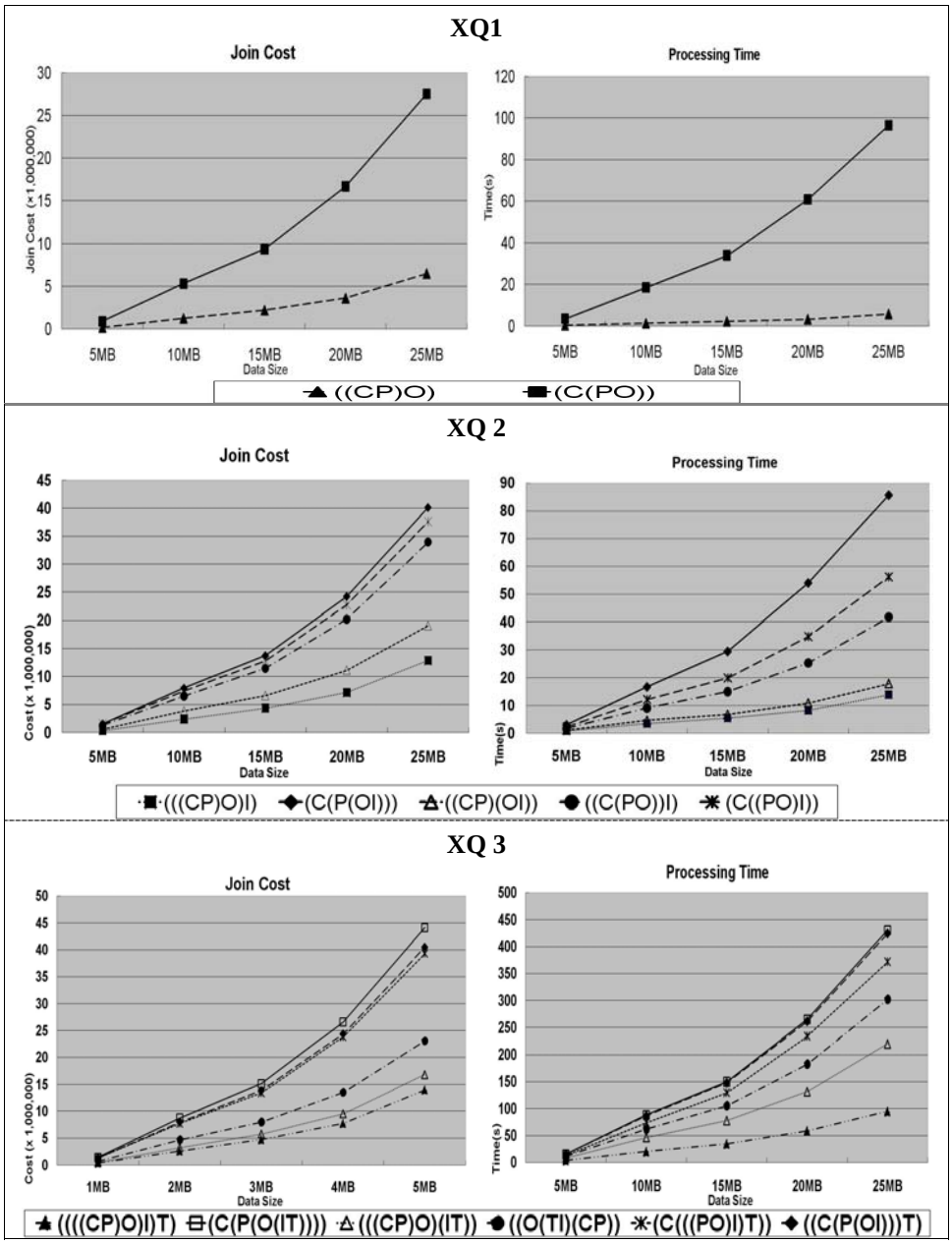

Fig. 8. Estimated join cost and processing time of XQ1, XQ2, XQ3.

Fig. 8 shows the estimated join cost by our algorithm and the processing time of XQ1, XQ2, and XQ3. Each graph shows the possible processing order of local queries. For example, the method ((CP)O) first joins the 'C' local system with the 'P' system, and then joins the 'O' system with the result of the first join. The three left graphs in Fig. 8

describe the join cost estimated by our algorithm and the three right graphs describe the real processing time. In the case of XQ2, the number of cases for the possible processing order is six by ${}_3P_3$. However, this paper shows five cases because the ((CP)(OI)) and ((OI)(CP)) are the same cases. In the case of XQ3, the number of cases for the possible processing order is twenty-four by ${}_4P_4$. Nevertheless, this paper describes just six of the best cases.

The cost graphs shown by our Join Cost-Based Strategy are similar to the performance graphs by real processing time. Also, we know that our method is not dependant on the number of join operations and the size of data. The result of the evaluation shows that our strategy is one of the most efficient approaches for deciding the optimal processing order of local queries including join operations.

5. CONCLUSION

In this paper, we have proposed the join cost-based strategy for processing global XQuery queries. For this strategy, we have addressed some considerations when process global XQuery queries and especially defined the difference point for estimating the join cost between SQL and XQuery queries. Also we have proposed the ECNJ algorithm which is used to estimate the cost for next join operation. In order to evaluate our strategy, we have implemented a prototype system and compared the estimated join cost with real processing time. Our result can obviously be applied to any XQuery processor that focuses on the integration and search of distributed data, since our processing strategy is a general method.

Currently, our strategy considers a join cost. In the future, we hope to deal with the communication cost and consider the features of functions and operations in XQuery queries.

REFERENCES

1. G. Gardarin, A. Mensch, T. Dang-Ngoc, and L. Smit "Integrating heterogeneous data sources with XML and XQuery" in *Proceedings of the 13th International Workshops on Database and Expert Systems Applications*, 2002, pp. 839-846.
2. D. H. Hwang and H. C. Kang, "XML view materialization with deferred incremental refresh: The case of a restricted class of views," *Journal of Information Science and Engineering*, Vol. 21, 2005, pp. 1083-1119.
3. I. Manolescu, D. Florescu, and D. Kossmann "Answering XML queries over heterogeneous data sources" in *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, pp. 241-250.
4. J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, and J. Funderburk, "Querying XML views of relational data," in *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, pp. 261-270.
5. XQuery 1.0, An XML Query Language, 2007.
6. I. Eldosouky, H. Arafat, and A. A. Eldin, "New heuristic approaches for improving distributed query processing based on the enhancement of semi-join strategies," in *Proceedings of the International Conference on Statistics, Computer Science and*

Operational Research, 2001, pp. 1-15.
7. L. Liu, C. Pu, and K. Richine, "Distributed query scheduling service: an architecture and its implementation," *International Journal of Cooperative Information Systems*, Vol. 7, 1998, pp. 123-166.
8. M. J. Yu and P. C. Y. Sheu, "Adaptive join algorithms in dynamic distributed data-bases," *Distributed and Parallel Databases*, Vol. 5, 1997, pp. 5-30.
9. X. Lin and M. E. Orlowska, "An efficient processing of a chain join with the minimum communication cost in distributed database systems," *Distributed and Parallel Databases*, Vol. 3, 1995, pp. 69-83.
10. C. Shahabi, L. Khan, and D. McLeod, "A probe-based technique to optimize join queries in distributed internet databases," *Knowledge and Information Systems*, Vol. 2, 2001, pp. 373-385.
11. N. May, S. Helmer, C. C. Kanne, and G. Moerkotte, "XQuery processing in natix with an emphasis on join ordering," in *Proceedings of the 1st International Workshop on XQuery Implementation, Experience and Perspectives*, 2004, pp. 49-54.
12. A. Halverson, J. Burger, L. Galanis, A. Kini, R. Krishnamurthy, A. N. Rao, F. Tian, S. D. Viglas, Y. Wang, J. F. Naughton, and D. J. DeWitt, "Mixed mode XML query processing," in *Proceedings of the 29th International Conference on Very Large Data Bases*, 2003, pp. 225-236.
13. J. T. McClave and T. Sincich, *Statistics*, Prentice Hall, New Jersey, 2006.
14. M. Steinbrunn, G. Moerkotte, and A. Kemper, "Optimizing join orders," Technical Report MIP9307, Faculty of Mathematic, University of Passau, Passau, Germany, 1993.
15. W3C, XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Recommendation 23 January 2007, http://www.w3.org/TR/2007/REC-xpath-functions-20070123/.
16. A. Aboulnaga, A. R. Alameldeen, and J. F. Naughton, "Estimating the selectivity of XML path expressions for internet scale applications," in *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, pp. 591-600.
17. J. Freire, J. Haritsa, M. Ramanath, P. Roy, and J. Simeon, "Statix: Making XML count," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2002, pp. 181-191.
18. M. Steinbrunn, G. Moerkotte, and A. Kemper, "Heuristic and randomized optimization for the join ordering problem," *The International Journal on Very Large Data Base*, Vol. 6, 1997, pp. 191-208.
19. C. Re, J. F. Brinkley, K. P. Hinshaw, and D. Suciu "Distributed XQuery," in *Proceedings of the 33rd International Conference on Very Large Data Bases*, 2004, pp. 116-121.
20. Saxonica's Saxon, http://saxonica.com/.

Jong-Hyun Park received his Ph.D. and M.S. degrees in Computer Science from Chungnam National University, South Korea, in 2002 and 2007, respectively, and his B.S. degree in Computer Science from Woosong University, South Korea, in 1999. He is now working toward researcher in software Research Center, Chungnam National University. His research interests

include XQuery Optimization, XML database, Distributed XQuery Processing, XML data mining, and web information system, ontology, information inference, semantic web, information inference, data base systems.

Ji-Hoon Kang received the B.S degree in Seoul National University, South Korea, in 1979 and the M.S. and the Ph.D. degree in KAIST, South Korea, in 1981 and 1996, respectively. From 1983 to 1985, He was with Samsung Electronics Co. and worked in Cheil Wool Textile Co. from 1981 to 1983. Since 1985, he has been a Professor of the Faculty of Computer, The Chungnam National University. His current research interests include Web-based Information, xquery processing, xml, digital library, semantic web, reasoning, hypermedia systems, database systems.