Cube-Based Analysis for Maintaining XML Data Partition for Holistic Twig Joins

Imam MACHDI[♦] Toshiyuki AMAGASA[♠]
Hiroyuki KITAGAWA[♠]

In this paper, we propose an abstraction for maintaining XML data partition, especially for holistic twig joins processing in a cluster system through a multidimensional data model. As XML documents, XML schemas and queries are numerous and intricacy in our system, we extract their metadata to define such a relationship among them in a multidimensional data model. For the partitioning purpose, we propose a series of multidimensional analysis operations outlined in three basic steps: document clustering, query clustering and partition refinement. Each step yields partitions with their associated costs computed by a cost model that takes a query processing cost as the basis. During simulated distribution of partitions to cluster computers, we refine some partitions residing in an overloaded cluster node and redistribute them in order to achieve considerably well balanced costs among all cluster nodes. Finally, we show the effectiveness of our proposed method indicated by achieving minimized cost variance in the cluster system and good performance of query execution.

1. Introduction

As the performance of current query processing methods [5, 10, 12], which are based on non-parallel system, suffers from involving huge XML data and complex queries, parallel query processing techniques have gained more attention recently to improve the system performance. One of the most important issues in designing a parallel technique in shared-nothing cluster systems is data partitioning strategy. Many models have already been proposed for XML data partition, but they especially work well for homogeneous XML data. In fact, a system may contain various XML documents with different sizes and contents and many different XML schemas, and numerous complex queries are submitted against those XML documents. Distributing XML documents to a cluster system has raised a problematic issue that leads to combinatorial optimization solution in order to achieve good workload balance and good performance of query processing in the system. Therefore, the challenge of XML data partition model is to view the conceptual model for maintaining information about XML data partitions and distribution.

[♦] Student at Graduate School of Systems and Information Engineering, University of Tsukuba
machdi@kde.cs.tsukuba.ac.jp
[♠] Faculty at Graduate School of Systems and Information Engineering and Center for Computational Sciences, University of Tsukuba
{amagasa, kitagawa}@cs.tsukuba.ac.jp

In this research, we study a partitioning technique based on the multidimensional data model especially for holistic twig joins processing [5] executed in a shared-nothing cluster system. Our main objective is to provide an abstraction for partitioning XML data stored in the form of streams of nodes and for distributing partitioned streams of nodes and queries. XML metadata describing relationships among XML documents, XML schemas and queries are organized in such a way to construct multidimensional data, which is also known as data cube. For the purpose of XML data partition analysis, we present some OLAP-like operations performed on the data cube such as clustering, rolling up and splitting on dimensions. Results of an operation will partition the data cube; consequently, it will reflect the partition of streams of nodes. Inspired by our previous works [3, 11], we also adopt a cost-based approach for measuring a query processing cost. Every partition is associated with an accumulated cost of query processing. Eventually, distributing partitions to cluster computers will lead to good workload balance in terms of this cost.

The rest of this paper is organized as follows. Section 2 provides an overview of related work and in Section 3 we describe some preliminary issues related to XML data model and its representation that underlay our work. In Section 4, we present our proposed method. Thereafter, the experimental results are evaluated in Section 5. We close this paper with a conclusion and future works in Section 6.

2. Related Work

Processing tree pattern queries in parallel systems has attracted a lot of attention recently, and most approaches focus mainly on structural joins processing [3], including our previous works in [3, 11]. In addition, Mathis et al. in [1] proposes locking-aware structural joins operators for twig query evaluation and just outlines briefly an overview of twig join algorithm. On the other hand, some works have been proposed to process query patterns using some other approaches. WIN [6] adopts query rewriting and hash join operation with index structures for all query expressions. Graph traversal is used in [4] for answering queries as XML data is represented as a graph. The use of XPath as the query processor for processing queries is proposed for a native XML storage [8].

Many works aim to partition XML data for load balancing purpose and to introduce data allocation strategy in parallel systems. Partitioning XML tree into subtrees and maintaining them to different sites are utilized in WIN [6]. The work of Lu et al. in [4] directly employs the notion of vertical and horizontal partitioning in relational databases, while in our previous works [3, 11] the vertical and horizontal partitioning in relational tables is based on schema graph decomposition. To compute efficient XML data allocation, workload information and cost model have been proposed in [3, 6, 9, 11]. Bremer et al. [2] present fragmentation of XML data using an extended XPath that supports a vertical-horizontal fragmentation approach based on schema structures. In addition, the authors propose a Repository Guide, which is an extension of Data Guide, for indexing fragmented data for distribution purpose. In native XML databases, Bordawekar et-al. [8] introduce XML tree partitioning by clustering techniques where related XML nodes can be clustered and stored in the same disk page.

3. Preliminaries

In this section, we present a brief introduction related to XML data, query patterns and data representation. These concepts are basically derived from the work of holistic twig joins in [5].

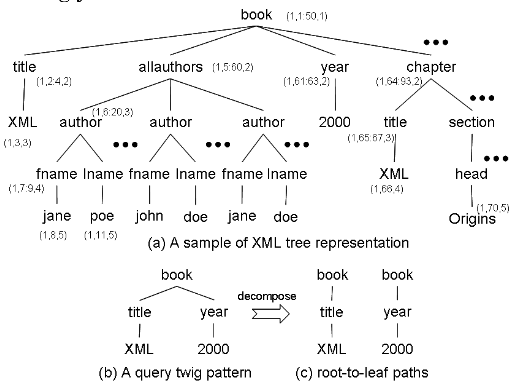

Fig. 1 (a) XML tree representation, (b) a query twig pattern and (c) root-to-leaf paths

3.1 XML Data Model

An XML document is a rooted, ordered, labeled tree, where each node corresponds to an element and the edges representing (direct) element-subelement relationships. Node labels consist of a set of (attribute, value) pairs, which suffices to model tags, PCDATA contents, etc. In addition, an XML document is identified by its document id and its file name including its full path, if necessary. In this paper we use the term document to refer to XML document. Figure 1 (a) shows the tree representation of a sample XML document.

3.2 Query Twig Patterns

A query twig pattern Q recognized by its query id and having a frequency of occurrences is a node-labeled tree pattern with elements and string values as node labels and its edges represent parent-child or ancestor-descendant relationships as shown in Figure 1 (b). It can be decomposed into a set of root-to-leaf path patterns as illustrated in Figure 1 (c) and each root-to-leaf path is identified by its path id. In this paper we use the term query to refer to a query twig pattern and the term query path to refer to a root-to-leaf path.

3.3 Representation in XML Database

The position of a string occurrence in an XML document is represented as a 3-tuple (DocId, LeftPos, Level) and analogously, the position of an element occurrence is as a 3-tuple (DocId, LeftPos : RightPos, Level), where (i) DocId is the identifier of the document; (ii) LeftPos and RightPos can be generated by counting word numbers from the beginning of the document DocId until the start and the end of the element, respectively; and (iii) Level is the nesting depth of the element or the string value in the document. By having this enumeration fashion, structural relationships of parent-child and ancestor-descendent can be determined easily.

XML database represents XML data in a set of streams of nodes. A stream of nodes contains elements of the same name in XML documents where each node in the stream is represented as a 3-tuple. Similarly, all string occurrences in XML documents are stored in one stream of nodes. Nodes in a stream are sorted by their (DocId, LeftPos).

4. The Proposed Method

This section describes our basic parallel processing system, an overview of the system configuration, the cost model and XML data partitioning over multidimensional data analysis operations.

4.1 Parallel Processing System

In our shared-nothing cluster system, one of the cluster nodes is selected as the coordinator that plays roles of accepting queries from users, delivering queries to the appropriate computers, collecting solutions and sending the solutions back to the users. Other cluster nodes are responsible for processing queries as instructed by the coordinator and delivering the solutions to the coordinator.

4.2 System Configuration

In this subsection, we present an overview of the system configuration for constructing cube and streams of nodes and distributing them into a cluster system as illustrated in Figure 2.

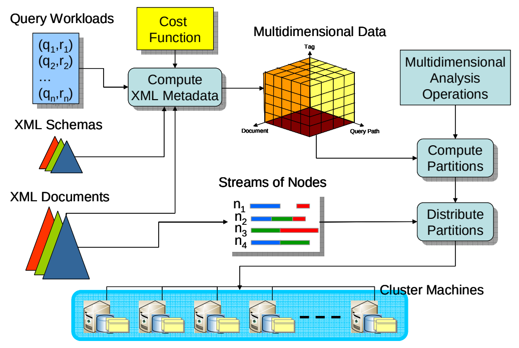

Fig. 2 An overview of the proposed scheme

XML metadata to be maintained in data cube is summarized from traversing XML documents, XML schemas and queries. Here, document identifiers indicating XML documents build a document dimension in the data cube. Meanwhile, distinct tags from XML schemas and distinct queries build a tag dimension and a query dimension, respectively. Since a query may be decomposed into its query paths, a query dimension may also be refined into a query-path dimension; in terms of the query dimension, the data cube may be drilled down into the query-path dimension. Once all dimensions are completely determined, we need to define a relationship among all dimensions to functionally determine a cost measure. In this case, a tag that is associated with an element of a query path and associated with element occurrences in an XML document contributes a cost partially to the query path processing for the XML document. Every occurrence of relationships among all dimensions is stored as an instance in the data cube. This relationship is illustrated in Figure 3; we can notice that tag *e* does not satisfy the relationship because it has no association with any query path and neither with any query. In addition, the number of node occurrences for each element in every XML document is counted for cost calculation, which will be explained in the next subsection.

In the meantime, streams of nodes are generated by traversing and enumerating positions of elements and string values in each XML document as explained in the preliminary section.

As the data cube is analyzed for partitioning, multidimensional analysis operations are applied. In relation with the streams of nodes, partitions in the data cube describe partitions in the streams of nodes. Eventually, partitions of the streams of nodes along with their associated partitions of the data cube are distributed to cluster nodes.

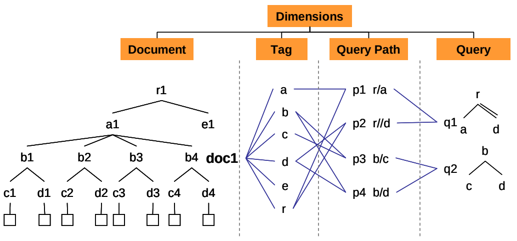

Fig. 3 Relationships among dimensions

4.3 Cost Model

In this section, we propose a cost model to estimate a cost of processing a query in our parallel system. A cost is measured in terms of the number of element occurrences as the unit. We consider computation and communication costs as parameters that primarily influence query performance in our parallel system. As a query may be decomposed into its query paths in the worst case, we also consider the cost of a query path as the foundation of computing the query processing cost.

$$C(q) = \sum_{q_i \in q} q_i \qquad (1)$$

$$C(q_i) = f_q(\alpha C_{comp} + (1-\alpha) C_{comm}) \qquad (2)$$

$$C(q_i) = f_q(\alpha(2\gamma+1) \sum_{n \in inputs} |n| + (1-\alpha)\gamma \sum_{n \in inputs} |n|) = f_q(\alpha + \alpha\gamma + \gamma) \sum_{n \in inputs} |n| \qquad (3)$$

$$C(q_i) = \sum_{n \in inputs} C(q_{in}) \quad \text{where} \quad C(q_{in}) = f_q(\alpha + \alpha\gamma + \gamma)|n| \qquad (4)$$

In this cost model, let us consider a query that accesses a document. The processing cost of a query is the sum of the processing costs of its query-paths as written in expression (1). Expression (2) states the cost of an individual query path, where f_q is a probability of query occurrence in the system and α is a coefficient for weighting computation cost C_{comp} and communication cost C_{comm}.

In the holistic twig joins [5], principally the computation time includes I/O and CPU time. Its computation complexity is linear in the sum of input lists ($|inputs|$) and output lists ($|outputs|$). The $|outputs|$ is estimated by a fraction γ of $|inputs|$; fraction γ value can be derived from statistics of past query execution. In the first phase of the algorithm, generating query solution extensions, which are node candidates for solutions, and root-to-leaf path solutions, which can be obtained from solution extensions, takes $|inputs|$ and $|outputs|$, respectively. The second phase to merge all partial solutions has the complexity of $|outputs|$. Thus, the computation cost of a query-path processing is $\alpha(2\gamma+1)|inputs|$. In addition to the computation cost, the communication cost of a query path occurs when sending solutions from a cluster node to the coordinator. In this case, the communication complexity is linear to output lists ($\gamma|inputs|$). Therefore, the overall cost of processing a query path can be simply stated in expression (3).

Finally, to cope with a cost measure for instances in the data cube, the query path cost is broken down to an element cost that takes the number of the element occurrences in an XML document into consideration. This element cost is expressed in (4).

4.4 XML Data Partitioning over Multi-dimensional Analysis

In analyzing the data cube for partitioning purpose, we use several operations as follows:

(1) 2-D projection is to map a dimension on another dimension with its associated value. For example, by mapping a document on the tag dimension, we get the number of occurrences for each distinct tag value in the document.

(2) Roll-up is to increase the level of aggregation. For example, a query-path dimension is rolled up for a query dimension with an aggregated value of the cost measure. The opposite operation is called drill-down.

(3) Split (slice) is to split a data cube into several partitions by slicing specific values for one or more dimensions. For example, splitting a data cube based on document wise is to split it according to a given specific value for a document dimension.

(4) Cross tab is to reorient a 2-D projection with the aggregation of mapping values where distinct values of a dimension is represented in rows and distinct values of another dimension is represented in columns. This operation is utilized for outlining cluster data.

(5) Clustering is to group instances of a dimension according to their similarity values. Details are discussed in subsection 4.4.1 and 4.4.2.

We aim to partition XML data stored in streams of nodes by means of multidimensional analysis. Here, a partition is defined as a subcube of the data cube. It is created as the result of slicing the data cube on one or more dimensions for specific values. A refined partition is a subcube of a partition. For generality, a refined partition is also called a partition due to possibly further refinement. In relation with query processing costs, a partition is associated with a cost that is computed as the sum of all contributed element costs of query paths that are grouped in this partition. Furthermore, when some partitions are distributed to a cluster node, their costs are accumulated as the cost of the cluster node.

Relationships between partitions of a data cube and streams of nodes can be described as follows. When a data cube is partitioned by document dimension values, it actually reflects vertical partitioning on streams of nodes. On the other hand, when a data cube is partitioned by tag dimension values, query dimension values or query-path dimension values, it reflects horizontal partitioning on streams of nodes. Figure 4 illustrates the association of partitions between a data cube and streams of nodes.

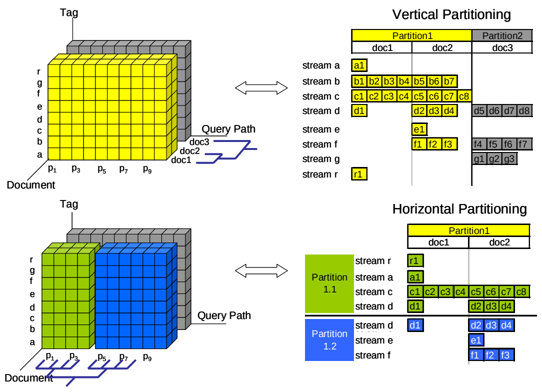

Fig. 4　Partitions on data cube reflect partitions of streams of nodes

4.4.1　Document Clustering

Since we have numerous XML documents in the system, in the first step those XML documents need to be grouped according to their tag similarity and the resulted groups of XML documents are regarded as the initial partitions. We select a hierarchical clustering technique from other standards such as K-means clustering and density-based clustering [7] because we assume that ideally there is no clue about the expected number of clusters and we let users have flexibility to analyze and decide the best cluster results. To outline the clustering data, we perform the following steps:

- 2-D projection on document and tag dimensions with the number of tag occurrences as the mapping value,
- Cross tab operation on the 2-D projection data such that distinct values of a document dimension are outlined in row wise as the cluster objects, distinct values of a tag dimension are outlined in column wise as the feature vector, and the aggregated values of tag occurrences become the values of the proximity measure, and
- Normalization of proximity measure by taking the ratio of the number of tag occurrences in a document against the entire number of tag occurrences in that document so that similar XML documents with different sizes will be grouped in the same cluster.

In clustering computation, the proximity measure that defines the similarity of objects in clusters takes the Euclidean (L_2) distance for computing single link, complete link or group average distance measure. Cutting height of dendogram and measuring cluster quality are up to user's choice. As a result, the data cube is sliced on the document dimension according to document clustering results where a cluster corresponds to one partition in the data cube. Generally, costs of partitions are still considerably high so that they need to be refined in the next step.

4.4.2　Query Clustering

In general, a resulted partition containing documents in the document clustering step is associated with many queries. Our objective of this step is to refine previously resulted partitions by grouping queries that have similar query elements. To outline the clustering data, for each previously resulted partition we perform the following steps:

- Roll up operation on a query-path dimension for a query dimension in the data cube and perform 2-D projection on query and tag dimensions by ensuring only queries and tags belonged to this partition, and
- Cross tab operation on the resulted projection such that distinct query dimension values are outlined in row wise as the cluster objects and tag dimension values are outlined in column wise as the feature vector. In this case, the mapping value is one for every matching value of a query as the cluster object and a tag in the feature vector.

In clustering computation, unlike the previous clustering, the Manhattan distance is more appropriate for proximity measure because a query has no tendency of weighting the feature vector values. Clustering results in this step reflect further slicing on the query dimension of the data cube to produce refined partitions.

4.4.3　Partition Refinement

The objective of partition refinement is to refine partitions that have considerably high costs into several refined partitions with lower costs and redistribute them to achieve good load balance. Initially, we simulate the distribution of partitions using Round Robin approach combined with greedy approach to minimize the entire cost variance among all cluster nodes. Partition refinement is conducted on certain partitions in a cluster node whose loads exceed a specified threshold δ value that is defined as the average value of costs held by all cluster nodes. Resulted refined partitions are, then, redistributed to underloaded cluster nodes. We keep doing partition refinement and redistribution until the objective of distribution is achieved.

There are three cases in which partition refinement is necessary to perform. If a partition in an overloaded cluster node is constructed by the followings:

(1) one or more queries that are associated with many documents, then the partition is split according to documents, or
(2) many queries that are associated with a single document, then the partition is split according to queries, or
(3) a single query that is associated with a single document, then the partition is split according to query-paths.

5.　Experimental Evaluation

In this section we conducted a preliminary experiment using heterogeneous XML data sets. The main objective is to show the effectiveness of partitions costs and queries distribution by our proposed partitioning method.

5.1　XML Data Sets

As shown in Table 1 we list XML data sets that consist of 194 XML documents with the total size of 328 MB, 17 DTDs and 11 different contents of interests. The size of XML documents varies greatly as indicated by the variance value in the table. We derived the XML data sets from several sources. The first data set up to the seventh data set were derived from experimental data sets used by Niagara Query Engine [13]. The XML data set about movies was derived from the Standford InfoLab [14]. Synthetic XML data sets generated by XMark [15] and XOO7[16] were also used. Based on given DTDs, we generated 64 query twig patterns randomly.

After constructing the data cube, the sizes of document dimension, tag dimension and path dimension are 194, 485, and 177, respectively. The data cube contains 6,276 instances; it indicates the data is sparse and requires small storage.

Table 1 XML data sets and query twig pattern sets

No	XML Data	Number of DTDs	Number of Queries	Number of XML Docs	Total Size (bytes)
1	Bibliography	1	4	16	158,096
2	Sport Club	1	3	12	162,986
3	Cars	1	6	48	1,357,856
4	Departments	1	5	19	2,722,723
5	Purchases	1	2	10	4,873,260
6	Quotes	1	2	10	4,412,418
7	Dramas	1	3	18	7,428,278
8	Sigmod 2002	3	10	43	2,934,193
9	Movies	5	21	5	28,463,633
10	Auction (XMark)	1	4	8	119,900,420
11	Comp. Assembly (XOO7)	1	4	5	171,309,031
	Total	**17**	**64**	**194**	**343,722,894**
	Average of File Size				**1,771,767**
	Variance of File Size				**6.20E+13**

5.2 XML Data Partitioning

Before performing XML data partitioning, we conducted a benchmark to determine the parameters used in the cost model. The α value is 0.97 for weighting the computation cost toward the communication cost. The average γ value to estimate the size of solutions is 0.1.

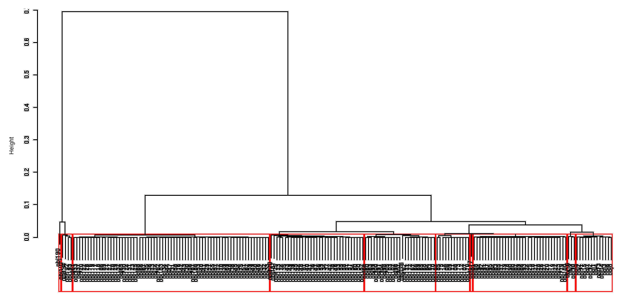

Fig. 5 Agglomerative hierarchical clustering with group average proximity method

In the XML data partitioning step, we performed the document clustering step with three different proximity methods: single link, complete link and group average, and with the dendogram cutting height of 0.01. To measure the validity of clusters we measured the Silhouette coefficient for each type of proximities. The averages of Silhouette coefficient for single link, complete link and group average proximities were 0.13, 0.80 and 0.81, respectively. Therefore, the group average proximity is selected and it yielded 10 clustering results regarded as initial partitions. Figure 5 illustrates the dendogram of clustering results for group average proximity.

The next step, query clustering, was conducted in the same procedure as the document clustering to refine partitions. Instead of selecting average group proximity, the single link proximity was selected because its Silhouette coefficient showed better value than other methods. Finally, it yielded 27 refined partitions, which were initially derived from 10 partitions.

Table 2 Distribution of costs of partitions and queries

Cluster Node Id	Initial	Final	
	Cost of Partitions	Cost of Partitions	Query Distribution
1	5,213.76	7,936.92	Q27, Q28, Q29, Q30, Q31, Q32, Q45, Q58, Q59, Q60, Q61, Q62, Q63
2	17,568.58	8,258.14	Q33, Q36, Q44
3	5,290.53	8,073.13	Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q34, Q35, Q36, Q64
4	10,839.99	8,531.30	Q33, Q36, Q43
5	5,832.10	7,959.90	Q15, Q16, Q17, Q20, Q21, Q22, Q23, Q24, Q25, Q27, Q28, Q29, Q30, Q31, Q32, Q33, Q36, Q55, Q56, Q57
6	6,687.01	7,916.58	Q11, Q12, Q14, Q33, Q36, Q37, Q38, Q39, Q40, Q41, Q52
7	6,793.38	7,881.50	Q26, Q42, Q47, Q48, Q49, Q51, Q52, Q53, Q54, Q64
8	6,427.97	8,095.85	Q18, Q19, Q46, Q50
Total	**64,653.32**	**64,653.32**	
Threshold δ	**8,081.66**	**8,081.66**	
Variance	**1.79E+07**	**4.81E+04**	

The initial distribution of 27 partitions into 8 cluster nodes is shown in Table 2. The result of distribution displays unbalanced loads among cluster nodes as indicated by a large variance value of cost distribution.

In the partition refinement step, partitions in some cluster nodes that exceeded the specified threshold δ were refined further to yield 35 refined partitions. For example, a partition in cluster node id 2 containing Q33 and Q36 underwent all three cases of partition refinement so that Q33 and Q36 were redistributed to other nodes. As the result, the final partition distribution shows more balanced costs as indicated by the variance value, which is reduced by 99.7% from the initial variance value.

In addition, we measured parallel speed up performance by implementing the basic holistic twig joins processing and executing in parallel those 64 queries for the XML data sets, which are also partitioned by other different techniques: (i) we partitioned XML data based on document-wise; (ii) we partitioned XML data based on document clusters. We considered the size of a document as a cost for the two methods and used the same distribution approach, which is Round Robin approach combined with greedy approach with the objective of minimizing cost variance among all cluster nodes, for all methods. As shown in Figure 6, our proposed method of partitioning heterogeneous XML data contributes to higher parallel speed up performance than the other two methods, mainly because the other two methods have no means of refining partitions when some cluster nodes are overloaded by partitions containing large document sizes.

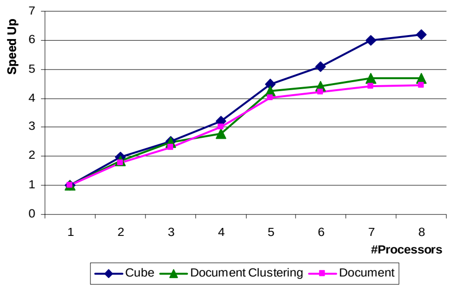

Fig. 6 Parallel speed up performance

6. Conclusions and Future Works

In this paper, we propose a new abstraction through a multidimensional data model for maintaining XML data partitions, especially for holistic twig joins processing executed in a shared-nothing cluster system. The proposed model provides a view of partitioning XML data from its dimension perspectives and outlines several multidimensional analysis operations including clustering techniques that are utilized to define and refine partitions. In the experiments, we show the effectiveness of this approach in the preliminary experiment that the overall cost variance is reduced significantly to achieve load balance and the parallel speed up is gained well.

For the future research, we plan to study parallel holistic twig joins processing in relation with our XML data partitioning approach. In addition, to cope with the growing XML documents in the system, we plan to study an adaptation of load balance changes due to addition of XML documents to the system.

Acknowledgments

This study has been partially supported by MEXT (#19024006), Grant-in-Aid for Young Scientists (B) (#19700083) and CREST of JST (Japan Science and Technology Agency).

References

[1] C. Mathis, T. Harder, M. Huastein: "Locking-Aware Structural Join Operators for XML Query Processing", ACM SIGMOD, 2006.
[2] J.M. Bremer, M. Gertz: "On Distributing XML Repositories", International Workshop on the Web and Databases (WebDB), 2003.
[3] K. Kido, T. Amagasa, H. Kitagawa: "Processing XPath Queries in PC-Clusters Using XML Data Partitioning", Proceedings of the International Conference on Data Engineering Workshops (ICDEW), 2006.
[4] K. Lu, Y. Zhu, W. Sun, S. Lin, J. Fan: "Parallel Processing XML Documents", Proceedings of the International Database Engineering and Applications Symposium (IDEAS), IEEE, 2002.
[5] N. Bruno, N. Koudas, D. Srivastava: "Holistic Twig Joins: Optimal XML Pattern Matching", ACM SIGMOD, 2002.
[6] N. Tang, G. Wang, J. Xu Yu, et-al.: "WIN: An Efficient Data Placement Strategy for Parallel XML Databases", Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS), 2005.
[7] P. Tan, M. Steinbach, V. Kumar: "Introduction to Data Mining", Pearson Addison Wesley, 2006.
[8] R. Bordawekar, O. Shmueli: "Flexible Workload-Aware Clustering of XML Documents", Database and XML Technologies, Second International XML Database Symposium, XSym, 2004.
[9] S. Abiteboul, A. Bonifati, et-al.: "Dynamic XML Documents with Distribution and Replication", ACM SIGMOD, 2003.
[10] S. Al-Khalifa, H.V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, Y. Wu: "Structural Joins: A Primitive for Efficient XML Query Pattern Matching", Proceedings of the International Conference on Data Engineering (ICDE), 2002.
[11] T. Amagasa, K. Kido, H. Kitagawa: "Querying Data Using PC Cluster System", The International Workshops on XML Data Management Tools and Techniques (XANTEC'07), 2007.
[12] Zhang, J. Naughton, D. DeWitti, Q. Luo, G. Lohman: "On Supporting Containment Queries in Relational Databases Management Systems", ACM SIGMOD, 2001.
[13] Niagara Query Engine. http://www.cs.wisc.edu/niagara/.
[14] Stanford University InfoLab. http://infolab.stanford.edu/pub/movies/dtd.html.
[15] XMark - An XML Benchmark Project. http://www.xmlbenchmark.org/.
[16] The XOO7 benchmark. http://www.comp.nus.edu.sg/~ebh/x007.html.

Imam MACHDI
He received B.Sc. in Computer Science from Louisiana State University, USA in 1995 and M.Sc. in Computer Science from the 10 November Institute of Technology, Indonesia and the University of Newcastle, UK in 2000. He is currently a Ph.D student at Graduate School of Systems and Information Engineering, University of Tsukuba.

Toshiyuki AMAGASA
He received B.E., M.E., and Ph.D degrees from Department of Computer Science, Gunma University in 1994, 1996, and 1999, respectively. He had been an assistant professor at Graduate School of Information Science, Nara Institute of Science and Technology (NAIST) from April 1999 to March 2005. Since April 2005, he has been an assistant professor at Center for Computational Sciences, and Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba. His main research areas cover data engineering and database systems including database systems for large scale XML data, P2P systems for XML data processing, and data management for scientific applications.

Hiroyuki KITAGAWA
He received B.Sc. degree in Physics and M.Sc. and Dr.Sc. degrees in Computer Science, from the University of Tokyo, in 1978, 1980, and 1987, respectively. He joined Institute of Information Sciences and Electronics, University of Tsukuba in 1988. He is currently a full professor at Graduate School of Systems and Information Engineering and at Center for Computational Sciences, University of Tsukuba. His research interests include integration of heterogeneous information sources, WWW and databases, structured documents, XML and semi-structured data, multimedia databases, and data mining. He is a member of ACM, IEEE Computer Society, the Database Society of Japan, IEICE, IPSJ, and JSSST. Also, he is an IEICE Fellow and an IPSJ Fellow.