A Multidimensional Data Structure for Maintaining XML Data Partitions

Imam MACHDI†, Toshiyuki AMAGASA†,††, and Hiroyuki KITAGAWA†,††

† Graduate School of Systems and Information Engineering
†† Center for Computational Sciences
University of Tsukuba
Tennodai 1–1–1, Tsukuba, Ibaraki, 305–8573 Japan

Abstract To achieve good performance of processing queries on huge XML data in cluster machines, data partitioning and placement strategy is one of the key factors. In this paper we propose a multidimensional data structure for maintaining XML data partitions, specifically for holistic twig join processing. Initially, we construct the multidimensional data structure from statistical information on various XML documents and queries execution that have been recorded in the past, such that dimensions of the data structure such as XML tag, query, and XML document can be composed. Also, we outline series of multidimensional analysis operations for generating and maintaining XML data partitions in three basic steps: document clustering, query clustering and partition refinement. Each step yields partitions with their associated costs that are computed by a cost function. As a part of partition refinement, some partitions having considerably high costs resided in overloaded machines are analyzed to select an appropriate multidimensional analysis operation for refinement. Finally, we show the effectiveness of our proposed method with the cost distribution measurement in cluster machines.

Key words XML Database, Multidimensional data, XML data partition

1. Introduction

The rapid growth of XML data for information representation and exchange has attracted many researches how to process the XML data efficiently and timely. A number of methods for query processing have been introduced in [6], [12], [15]. These studies are based on an assumption that the platform used for query processing is a non-parallel system. The system performance will deteriorate when the size of XML data is increased and complex queries (including join operations and multiple path expressions) are involved [14].

An attractive solution is using parallel techniques to improve system performance since the use of parallelism has shown good scalability in traditional database applications. In fact, one of the most important issues in the design of parallel shared-nothing systems is data partitioning strategy. Many models have already been proposed for data partition such as XML sub-tree partition, range partition, and horizontal-vertical partition. The challenge of data partition model is to view the conceptual model for managing data partition and guiding the distribution of data partition to processing nodes.

Our system contains many XML documents and a set of queries drawn from logs of query execution in the past. XML documents have various contents of interests and vary in sizes ranging from several KB to hundreds of MB, but the total size of XML documents in the systems could achieve TB. Distributing those XML documents into cluster machines is a problematic issue that contributes to combinatorial optimization solution in order to achieve good workload balance and good performance of query processing in the parallel system.

In this research, we study a partitioning technique based on the multidimensional data model for holistic twig join processing [6] executed in parallel shared-nothing cluster machines. The statistics regarding XML documents such as XML tag occurrences, frequencies of queries and costs can be derived from extracting information on multiple XML documents and query patterns and can be organized to form a multidimensional data. To produce XML data partitions, we present some operations performed on the multidimensional model such as clustering, rolling up and splitting on dimensions. Every partition is associated with a cost, which is relied on the computation of the defined cost function taking the complexities of communication and computation of a query in the parallel system into account.

The rest of this paper is organized as follows. Section 2 provides an overview of related work and in Section 3 we

describe some prelimnary issues on XML data model, query patterns, and a multidimensional data model that underly our work. In Section 4, we present the proposed XML data partitioning using the multidimensional data model. Thereafter, the experimental results are explained in Section 5. We close this paper with a conclusion and future works in Section 6.

2. Related Works

Processing query patterns in parallel system has attracted a lot of attention recently, and most approaches focus mainly on structural joins processing [3], including our previous works in [4], [13]. 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 other approaches. WIN [7] adopts query rewriting and hash join operation with index structures for all query expressions. The traversing graphs in [5] for answering queries are used, 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 [9] .

For storing XML data current repository system uses two main approaches. The first approach maps XML data into relational database [4], [5], [13] or into object oriented database [7] . Lu et-al. [5] stores an XML document by parsing it into DOM tree and mapping the DOM tree into relation describing properties of its arc, its source vertex, its target vertex, its vertex label, and its value. Path-approach is used to map XML data to relational tables in our past studies [4], [13] where XML data is stored into two relational tables. Node table describes properties of node id, path id, node number, node type and a value. The other one is path table that stores path id and path expression. The second approach of the current repository system, native XML storage, views the XML document as an XML tree [18]. XML tree is partitioned into distinct records containing disjoint connected subtrees and records are stored in disk pages using some internal representation.

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 [7]. The work of Lu et-al. in [5] directly employs the notion of vertical and horizontal partitioning in relational databases, while in our previous works [4], [13] 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 [4], [7], [11], [13], but

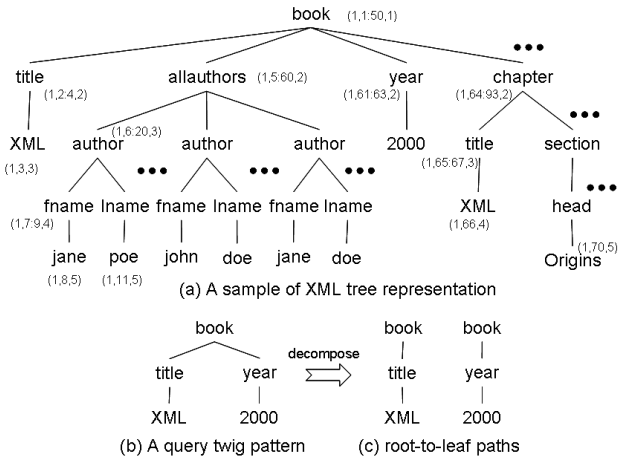

Figure 1 XML tree representation, a query twig pattern and root-to-leaf paths.

Lu et-al. do not discuss clearly its data placement strategy. Bremer et-al. [3] present fragmentation of XML data using an extended XPath that supports a vertical fragmentation approach and a horizontal fragmentation approach based on schema structures. In addition, the author proposes a Repository Guide, which is an extension of Data Guide [10], for indexing fragmented data for distribution purpose. In native XML databases, Bordawekar et-al. [9] introduce XML tree partitioning by clustering techniques where related XML nodes can be clustered and stored in the same disk page.

3. Preliminary

In this section we present a brief introduction related to XML data, query patterns and a multidimensional data model.

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 and its

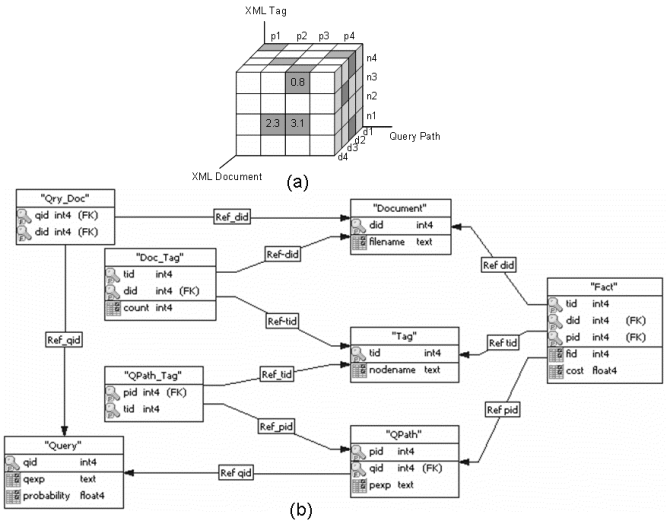

Figure 2 (a) Multidimensional data model, (b) A schema of a multidimensional data.

nodes. In this paper we use the term query to refer to query twig patterns and the term query path to refer to root-to-leaf path.

3.3 Multidimensional Data Model

A multidimensional data model is a model to view data as mapping from a point in dimensions to measures. It means that the model is constructed from dimensions, which are categorical attributes, and measures, which are the focus of analysis. In the relational implementation, a dimension may be associated with a table for storing its categorical attribute values and measures are attributes in a fact table that also contains keys related to dimension tables. A multidimensional data is also known as a data cube and Figure 2 (a) illustrates a multidimensional data representation whose dimensions are XML document, XML tag and query path. Conceptually, the multidimensional data model can be represented with Star schema, Snowflake schema, and Fact constellation [2]. We use Snowflake schema which is a refinement of star schema where the dimension hierarchy is normalized into a set of smaller tables. Figure 2 (b) shows a schema of a multidimensional data model. In this figure, *Document* table, *Tag* table and *QPath* table represent dimensions of document, tag and query path respectively. *Fact* table contains keys of these dimensions and a cost attribute as the measure.

3.4 Partitions

A partition is a subcube of XML data statistics. It is created as the result of slicing the multidimensional data 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 possibility of further refinement.

3.5 Multidimensional Analysis Operations

In analyzing multidimensional data, we use several operations as follows:

(1) 2-D projection is mapping two dimensions with its associated value. For example, *Doc_Tag* table represents a 2-D structure of document dimension and tag dimension with the mapping value of count, which is the number of tag occurrences in a document.

(2) roll-up is to increase the level of aggregation as for instance a query path dimension is rolled up for a query dimension with an aggregated cost value. The opposite operation is called drill-down.

(3) split (slice) is to split a data cube and create two or more partitions by slicing a specific value for one or more dimensions. For example, document split operation is to split the multidimensional data based on a given specific value for 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 as columns. This operation is utilized for outlining clustering data.

4. The Proposed Method

This section describes 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 cluster machines, one of the machines is selected as the coordinator that plays roles of accepting queries from users, delivering it to the appropriate machines, collecting solutions and sending the solutions back to the users. Other machines are responsible for processing queries as instructed by the coordinator and delivering the solutions to the coordinator.

4.2 System Configuration

The objective is to specify the system configuration that is capable of producing statistics of XML data represented in the form of a multidimensional data model for generating XML data partition. There are three main steps in gathering statistics of XML data: (i) traversing XML documents, (ii) fetching queries from logs, and (iii) constructing a fact table. Our system represents the multidimensional data using relational database system.

In the first step, every XML document is fed into the system to generate a document dimension and a tag dimension. Every XML document has an id and a file name that contribute to the document dimension whose values are stored in *Document* table. By parsing the content of each document, a tag dimension is constructed by considering distinct

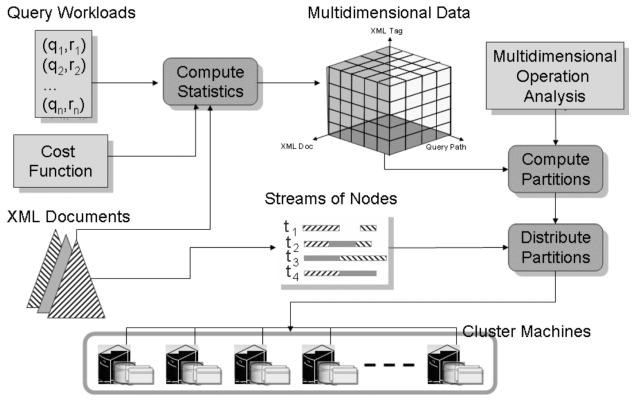

Figure 3 An overview of the proposed scheme.

tag name and is stored in a *Tag* table. In addition, a 2-dimensional structure that maps the document dimension denoted by document id *did* and the tag dimension denoted by tag id *tid* along with the number of tag occurrences as the mapping value is composed in *Doc_Tag* table. Meanwhile, 3-tuple representation of every tag occurrence as described in [6] is enumerated and is stored in a related stream of nodes. In this case, a table in XML database represents a stream of nodes.

Subsequently, for every query twig pattern in the log history, its query id and its frequency of occurrences contributes to a query dimension represented in a *Query* table. To cope with our partition scheme that due to cost imbalance a query might be decomposed into a set of root-to-leaf paths to derive finer partitions, the query dimension is refined in terms of a query path dimension. Thus, a map between a query and its query paths is created in *QPath* table. Additionally, a query path consisting of nodes associated with the tag dimension is extracted to form a 2-dimensional structure between the query path dimension and the tag dimension where it is maintained in a *QPath_Tag* table.

In addition, a mapping between query and document dimensions is required to ensure that the entire node names of a query are fully matched with the available tag names of an XML document. This mapping results are stored in *Qry_Doc* table.

Finally, the multidimensional data is accomplished by creating a *Fact* table that contains measures of costs and keys related to dimensional tables. The *Fact* table is built thorough joining *Query* table and the three tables of 2-D structures, which are *Doc_Tag*, *Path_Tag*, *Qry_Doc* tables under the join key attributes of *did*, *tid*, and *qid*. The cost attribute in *Fact* table will be computed by a cost function that will be explained in the next subsection. Also, the partition id *fid* attribute is initialized to the value of one; it will be updated as partitioning process is progressing. An instance in *Fact* table means that a node of a query path that occurs in an XML document contributes a cost of query processing. The overall relational schema of the multidimensional data is illustrated in Figure 2 (b) and the system configuration is described in Figure 3.

4.3 Cost Model

In the cost based allocation of partitions, cost estimation is a crucial part since the outcome of cost estimation will directly affect the data placement result, thereby influence the performance of a parallel system. In this section, we propose a cost model to estimate the cost of XML data partitions in a shared-nothing parallel environment and it is measured in terms of the number of node occurrences as the unit. Two parameters that are considered mainly influencing query performance in our parallel system are computation cost and communication cost. As in the worst case, a query twig pattern may be decomposed into root-to-leaf paths (query paths), we consider the cost of a query path as the foundation of computing the cost of a query twig pattern.

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

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

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

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

Formula (2) expresses the cost of individual query path, where f_q is the probability of query occurrence in the system and α ($\alpha < 1$) is the coefficient for weighting the computation cost C_{comp} and the communication cost C_{comm}. Meanwhile, the cost of query twig pattern is expressed in formula (1), which is the sum of costs of its query path patterns.

In the holistic twig joins [12] , basically the computation time includes I/O and CPU time and its computation complexity is linear in the sum of sizes of the input lists and output lists. The size of output lists is estimated to a fraction γ of the size of input lists, where fraction γ value ($0 \leq \gamma \leq 1$) can be derived from statistics of query execution in the past. If we take the size of output lists equal to the size of input lists ($\gamma = 1$), it means that all inputs satisfy a given query twig pattern to produce outputs in the best case. This computation complexity, which is an addition of $|inputs|$ and $|outputs|$, applies to the first phase of the holistic twig joins that generates solution extensions. The second phase that merges all root-to-leaf path solutions has the complexity of $|outputs|$. Thus, the computation cost of a root-to-leaf path pattern is $\alpha(2\gamma+1)|inputs|$.

On the other hand, the communication cost of a root-to-leaf path occurs in the second phase of the holistic twig joins in which root-to-leaf path solutions are merged from processing machines to a coordinator machine in cluster computers as described in the previous section. The complexity of communication relies on the size of output lists. Since the size of output lists is estimated from the size of input lists, the

communication cost of a root-to-leaf path is $(1-\alpha)\gamma|inputs|$.

In summary, the cost $C(q_i)$ is expressed in formula (3) and it is simplified in formula (4).

4.4 XML Data Partitioning over Multidimensional Analysis

Huge XML data containing many different contents of interests, different structures and numerous tags introduces such complexity in defining and maintaining XML data partitions. To perform XML data partition, we propose several multidimensional analysis operations described in three main steps: (i) document clustering, (ii) query clustering, and (iii) partition refinement. Every step results partitions and their associated cost values; each cost is the sum of all cost instances belonged to the partition.

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 tags similarity and the resulted groups of XML documents are regarded as the initial partitions. In this case, we select a hierarchical clustering technique from other standards such as K-means clustering and density-based clustering [2], [8] because we assume that ideally there is no clue about the expected number of clusters of varied XML documents and we let users have flexibility to analyze and decide the best cluster results. To outline the clustering data, we perform

- 2-D projection on document dimension and tag dimension with the number of tag occurrences as the mapping value; this can be derived from Doc_Tag table,
- cross tab operation on the 2-D projection data such that distinct document dimension values become the cluster objects in row wise, distinct tag dimension values become the feature vector in column wise and the aggregated values of tag occurrences become the values of the proximity measure,
- normalization of proximity measure for each tag in a document by taking the ratio of the number of tag occurrences against the entire number of tag occurrences in the document so that similar XML documents with different sizes will be grouped in the same cluster.

The proximity measure defining the similarity of objects in clusters adopts the Euclidean (L_2) distance for either single link, complete link or group average distance measure. The height of dendogram is selected to produce clusters according to user requirements. To evaluate which proximity measure gives the best clustering results, we use the Silhouette coefficient technique that measures both cluster cohesion and cluster separation [8]; we expect a positive coefficient average value for good clusters.

Finally, a partition is created by slicing the multidimensional data according to XML document ids grouped in a cluster. In addition, we need to compute the partition cost by summing up cost attribute values in $Fact$ table for documents belonged to the partition. 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, the resulted partitions in the document clustering step contains many queries. Our objective of this step is to refine the resulted partitions by grouping queries that share nodes in the same XML document aiming at storage efficiency. To outline the clustering data, for each partition we perform

- roll up operation on a query path dimension for a query dimension in the $Fact$ table and perform 2-D projection on query dimension and tag dimension by ensuring query dimension values belonged to the partition,
- cross tab operation on the resulted projection for each partition such that distinct query dimension values become the cluster objects in row wise and tag dimension values become the feature vector in column wise. 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.

Unlike the previous clustering, the Manhattan distance is more appropriate for proximity measure to select because a query has no tendency in terms of weighting the feature vector values. The rest of the procedure are the same with the previous clustering for determining the selected height of the dendogram, slicing a partition for refined partitions and assigning a cost for each refined partition.

4.4.3 Partition Refinement

Before conducting the actual XML data distribution to cluster machines, we simulate the distribution of partitions using Round Robin approach combined with greedy approach with the objective of minimizing the entire cost variance among all cluster machines. In this case, we need to define the cost of a cluster machine as the aggregation of all partition costs resided in the machine and a threshold τ value for determining overloaded machines. During this distribution computation, partition refinement is also continuously conducted for overloaded machines and redistribute the refined partitions to underloaded machines 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 machine consists of

(1) one or more queries and many XML documents, then perform document split operation. In this case, the partition is split into several new partitions in document wise.

(2) many queries and a single XML document, then perform query split operation. Here, the partition is decom-

posed into several new partitions in query wise.

(3) a single query and a single XML document, then perform query path split operation. A query is decomposed into its root-to-leaf paths where each path is considered as one partition.

5. Experimental Evaluation

In this section we conducted a preliminary experiment using small XML data set. The main objective is to show the distribution XML data and queries resulted from executing our partitioning method.

5.1 XML Data Set

As shown in Table 1 we list XML data that consists of 194 XML documents with the total size of 171 MB from 17 DTDs and 11 different contents of interests. The size of XML documents vary 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 [16]. The XML data set about movies was derived from the Standford InfoLab [17]. Synthetic XML data generated by XMark [18] and XOO7 [19] were also used. Based on given DTDs, we generate 64 query twig patterns randomly.

Table 1. XML data set and query twig pattern set.

No.	XML Data	Number of DTD	Number of Queries	Number of Files	Total Size (bytes)	Average Size (bytes)
1	Bibliography	1	4	16	78,992	4,937
2	Sport Club	1	3	12	81,415	6,785
3	Cars	1	6	48	678,568	14,137
4	Department	1	5	19	1,361,279	71,646
5	Purchase	1	2	10	2,436,555	243,656
6	Quotes	1	2	10	2,206,134	220,613
7	Drama	1	3	18	3,731,603	207,311
8	Sigmod 2002	3	10	43	1,486,807	34,112
9	Movies	5	21	5	14,231,952	2,846,390
10	Auction (Xmark)	1	4	8	59,950,150	7,493,769
11	Component Assembly (XOO7)	1	4	5	85,654,473	17,130,895
	Total	**17**	**64**	**194**	**171,877,928**	**171,877,928**
	Average of File Size					**885,969**
	Variance of File Size					**1.55014E+13**

After constructing the multidimensional data, the sizes of document dimension, tag dimension and path dimension are 194, 484, and 190 respectively. The fact table contains 6,248 records; it indicates the data is sparsely stored and requires small storage.

5.2 XML Data Partitioning

In the XML data partitioning step, first we defined the threshold τ as the cost average of storing partitions in 10 cluster machines; from *Fact* table we derived the total cost of 2,144,889 units and the average cost stored in cluster machines is 214,489 units. We conducted the document clustering step with three different proximity methods: single link, complete link and group average, and with the dendogram cutting height of 0.04 as described in Figure 4.

The dendogram with the single link proximity indicated undesirable clustering results as 96% of XML documents are

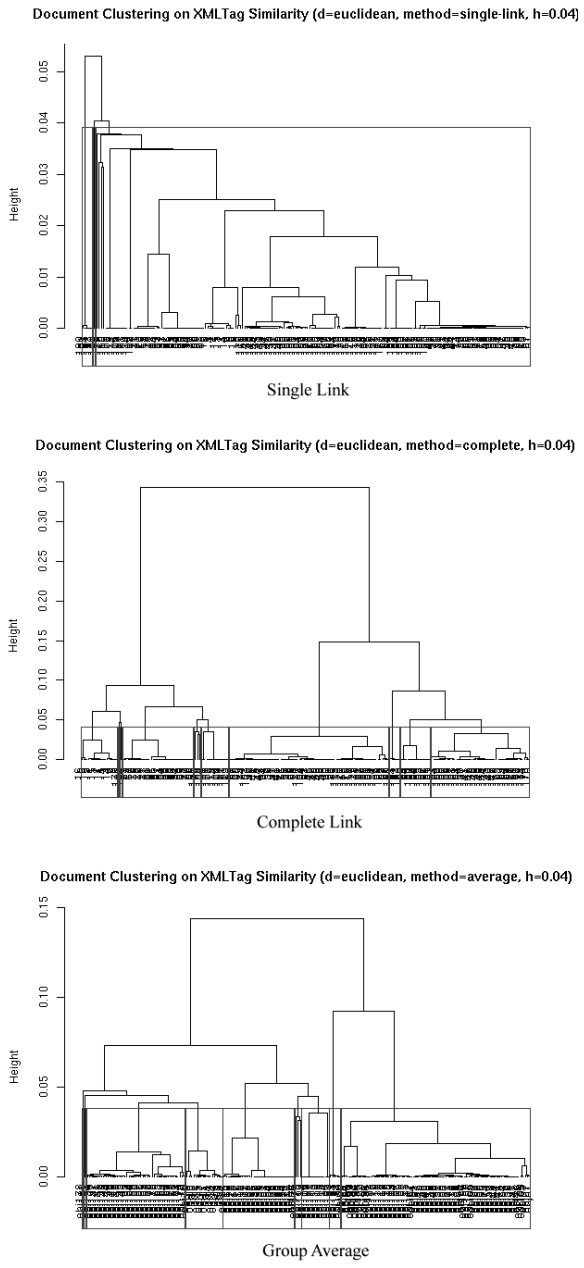

Figure 4 Agglomerative hierarchical clustering with proximity methods of single link, complete link and group average.

resided in one cluster. Meanwhile, complete link and group average proximities produced comparatively good clustering results. 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 -0.15, 0.73 and 0.76 respectively. Therefore, the group average proximity was selected to produce the clustering results.

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 as the measurement of cluster validity because its Silhouette coefficient showed better value

than other methods. Finally, it yielded 24 refined partitions, which were initially derived from 9 partitions.

In the partition refinement step, initially those 24 partitions were distributed to 10 cluster machines as shown in Table 2. We can see that aggregated costs of partitions in some machines exceeded the specified threshold τ and they were refined more further. The final refinement showed better distribution of costs as indicated by the variance value, which was reduced by 98.8% from the initial variance value.

As for the query distribution in Table 2, although some cluster machines have more queries than others, in terms of costs their workloads are about optimally equal balance. Also, because some queries access several XML documents, we can see that the queries i.e. Q33, Q34, Q35, Q36 are resided in more than one cluster machines as the result of performing document split and query split operations in the partition refinement step. In addition, if we take a look closely at Q33 in cluster machine number 4, solely its cost exceeds the specified threshold τ, eventhough the query was already decomposed into root-to-leaf paths. However, to measure the effectiveness of this query distribution, we need to measure the overall parallel performance, which is part of our future works to accomplish.

Table 2. Cost distribution in the cluster machines.

Machines	Initial Cost Distribution	Final Distribution	
		Final Cost	Query
1	136,890	205,604	Q15, Q16, Q17, Q22, Q23, Q24, Q25, Q26, Q55, Q56, Q57, Q61
2	317,728	209,220	Q19, Q28, Q42
3	346,228	209,968	Q5, Q18, Q37, Q38, Q39, Q40, Q41, Q51, Q52, Q53, Q54
4	774,504	269,955	Q33
5	99,393	199,625	Q13, Q21, Q34, Q35, Q36, Q49
6	98,004	197,232	Q20, Q43, Q59, Q60
7	98,044	194,550	Q8, Q11, Q12, Q14, Q33, Q34, Q35, Q36, Q44, Q45
8	86,514	204,326	Q33, Q34, Q36, Q63, Q64
9	100,590	244,874	Q46, Q47, Q48, Q50, Q62
10	86,994	209,535	Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q33, Q34, Q35, Q36
Total	2,144,889	2,144,889	
Average (τ)	214,489	214,489	
Variance	4.818E+10	5.746E+08	

6. Conclusions and Future Works

In this paper, we propose a novel scheme of XML data partitioning for parallel holistic twig join processing. We focus on building a multidimensional data structure for generating and maintaining the statistics of XML data partition. The multidimensional data model provides a view of partitioning XML data from its dimension perspectives. We describe several multidimensional analysis operations including clustering techniques that are utilized to define and refine partitions. As a result, we show the effectiveness of this approach in the preliminary experiment that the overall cost variance is reduced significantly to achieve load balance when partitions are physically distributed to the parallel cluster machines.

For the future research, we plan to study parallel holistic twig join processing and performance measurement 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 new XML document addition.

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. Han, M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, 2001.

[3] J.M. Bremer, M. Gertz. On Distributing XML Repositories. International Workshop on the Web and Databases (WebDB), 2003.

[4] 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.

[5] 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.

[6] N. Bruno, N. Koudas, D. Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching. ACM SIGMOD, 2002.

[7] 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.

[8] P. Tan, M. Steinbach, V. Kumar. Introduction to Data Mining. Pearson Addison Wesley, 2006.

[9] R. Bordawekar, O. Shmueli. Flexible Workload-Aware Clustering of XML Documents. Database and XML Technologies, Second International XML Database Symposium, XSym, 2004.

[10] R. Goldman, J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. Proceedings of the International Conference on Data Engineering (ICDE), 2002.

[11] S. Abiteboul, A. Bonifati, et-al. Dynamic XML Documents with Distribution and Replication. ACM SIGMOD, 2003.

[12] 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.

[13] T. Amagasa, K. Kido, H. Kitagawa. Querying Data Using PC Cluster System. Database and Expert Systems Applications (DEXA'07), 2007.

[14] W. Sun, K. Lu. Parallel Query Processing Algorithms for Semi-structured Data. The 14th International Conference on Advanced Information Systems Engineering (CAiSE 02), 2002.

[15] Zhang, J. Naughton, D. DeWitti, Q. Luo, G. Lohman. On Supporting Containment Queries in Relational Databases Management Systems. ACM SIGMOD, 2001.

[16] Niagara Query Engine. http://www.cs.wisc.edu/niagara/.

[17] Stanford University InfoLab. http://infolab.stanford.edu/pub/movies/dtd.html.

[18] XMark - An XML Benchmark Project. http://www.xml-benchmark.org/.

[19] The XOO7 benchmark. http://www.comp.nus.edu.sg/~ebh/x007.html.