A Clustered Index Approach to Distributed XPath Processing

Georgia Koloniari and Evaggelia Pitoura

Computer Science Department, University of Ioannina, Greece
{kgeorgia,pitoura}@cs.uoi.gr

Abstract—Supporting top-k queries over distributed collections of schemaless XML data poses two challenges. While XML supports expressive query languages such as XPath and XQuery, these languages require schema knowledge so as to write an appropriate query which may not be available in distributed systems with autonomous and dynamic sources. Thus, there is a need for approximate query processing. Furthermore, retrieving the top-k results incurs large communication and processing cost, since partial result lists from numerous sites need to be combined and ranked to assemble the top-k answers. To address both of these issues, we present an approach for approximate XPath processing over distributed collections of XML data based on a clustered path index, where data is grouped based on structural information. Our method gradually generalizes a query by applying a set of structural transformations to it and the retrieved results are ranked based on the edit distance between two path expressions. A compact indexing data structure is used to reduce the index construction cost. Our experimental results show that our approach significantly reduces the communication cost for retrieving the top-k results, while maintaining a low construction cost for the clustered index.

I. INTRODUCTION

XML has evolved as a standard for data representation and exchange in the Internet. In general, due to the structural heterogeneity of XML data and the frequent lack of schema information, queries are often interpreted approximately. Most previous research work addressed the problem of processing approximate or top-k queries on XML data in a centralized setting. Our focus in this paper is on distributed approximate XML processing. We consider a collection of XML documents distributed among a number of nodes. Users issue queries over this collection and are interested in retrieving the k most relevant results. This is achieved by relaxing the original query and ranking the results based on their relevance to it. The problem with distributed processing is that a node cannot determine how much to relax a query without contacting the other nodes.

The straightforward solution is to forward the query to every node. Each node relaxes the query so as to produce a list with the locally best k results and sends it to the query origin which then produces the final list of top-k results. However, the number of nodes involved and thus the number of intermediate results may be very large. Furthermore, there may be nodes that do not support approximate query processing capabilities. To address the lack of processing capabilities and achieve scalability, we propose partitioning the documents into groups and assigning a coordinator (or superpeer) to each group responsible for computing the top-k results. Each superpeer maintains appropriate indexes of the documents assigned to its group. Thus, a distributed index is formed over all documents in the system where each superpeer maintains a part of it. Along the spirit of [1], instead of processing queries on actual data, the superpeers relax queries based on their corresponding indexes.

Round-based Evaluation. Retrieving the top-k results proceeds in phases. When a query is issued by a node, the node propagates it to a superpeer. In the first phase (Local Query Evaluation), the superpeer forwards the query to all other superpeers. In parallel, each superpeer evaluates the query against its part of the index to attain k results. Note that results refer to index entries and not to the actual data that are stored at remote nodes. The results are sorted locally at each superpeer.

Following a procedure used in a family of distributed top-k evaluation algorithms (Threshold-based Algorithms [2]) for reducing communication costs, in a second phase (Elimination Phase), superpeers exchange their maximum scores (the score of the k-th result). Upon receiving these scores, each superpeer discards the results with score higher than the lowest of all the scores it has received. In the third phase (Result Construction), each superpeer forwards the remaining results to the superpeer from which it has received the query. After the initial superpeer has gathered all lists, it merges, locally ranks them to construct the final result list, and sends it to the node that issued the query. Finally, the node contacts the nodes it finds in the ranked list to retrieve the actual data.

The elimination phase of the routing process may consist of more than one round so as to further prune the result list and thus further reduce the size of the data that is transmitted over the network. In the second round, the superpeers exchange the scores and the position in their ranked list of the last result. Taking the position of the lowest score among all the scores sent, each superpeer compares it to its own list and finds all results with greater score. If the sum of the position of such a result with the position of the lowest score is greater than k then that result can be safely discarded since it does not belong to the final result list. The elimination phase can then proceed with the next round taking into account the newly acquired pruned lists in a similar manner.

We define a measure, called pruning degree to evaluate the number of results that are eliminated in each round of the elimination phase of query evaluation. Let us consider N superpeers, each maintaining a unit of the distributed index
Motivation for a Clustered Index. Instead of relying on degree indicates that less processing is required for construct-
the distributed index produced this way 

Let us consider a very simple example. Assume a set \( D \) of XML documents that can be classified in \( N \) categories \( (C_i) \) of size \( x \), such that if a query \( q \) matches a document \( d \in C_i \) then it does not match any documents belonging to the rest of the categories. Assume also that there exist more than \( k \) exact matches. We partition the documents to \( N \) superpeers: (a) uniformly at random, where \( x/N \) documents of each category fall into each superpeer and (b) by assigning the documents of each category to a different superpeer. In (a), \( q \)'s results are expected to be distributed uniformly among the superpeers while in (b), the results reside at a single superpeer. For simplicity, we assume that all non exact matches have the same distance with \( q \). While in (a), all exchanged scores have the same value and no pruning is possible, in (b), the superpeer of \( C_i \) has a \( k \)-th score equal to 0 and all other superpeers can prune their entire lists. The above example shows that a clustered index can increase the pruning degree when there is similarity among the documents.

In a sense, our clustered index is the analog to clustered indexes used in centralized applications to minimize the I/O cost of query evaluation, by placing similar data close to each other in disk. Our clustered index aims at reducing communication costs by placing similar data at the same superpeer, thus achieving a higher pruning degree.

II. INDEX DEPLOYMENT

To relax a user query, we apply a set of structural transfor-
mations \([3]\) to it. We provide an appropriate distance measure for ranking the results in the top-\( k \) list based on a variation of the edit distance between the original and the relaxed query. The variation is such that, the distance depends only on the number and type of structural transformations applied to the original query, without requiring any knowledge about the global data distribution.

We propose an appropriate structure for the clustered index that is (a) compact, so that the cost of communicating its entries among the nodes of the distributed system is small, (b) representative of the document structure and able to provide selectivity estimations, and (c) incrementally updatable through a cost-effective procedure. It also leads to an effective grouping that reduces the pruning degree of the relaxation procedure. Our index, called Multi-Level Bloom Histogram, is an enhanced version of two previously proposed Bloom-based data structures, the Bloom Histograms \([4]\) and the Depth Bloom Filters \([5]\) that are hash-based structures that summarize data within a small space overhead, while introducing tolerable false positive errors. Bloom Histograms (BH) \([4]\) are designed for supporting selectivity estimations for XPath expressions. They use a histogram to summarize the frequency of the path expressions that belong to a document, and a Bloom filter to summarize the values, i.e. the path expressions that fall into each bucket of the histogram. The Depth Bloom Filter (DBF) for an XML tree \( T \) with \( j \) levels is a set of simple Bloom filters \( DBF_i \), \( i \leq j \), where all paths of length \( i \) are inserted into \( DBF_i \). Multi-Level Bloom Histograms exploit both the selectivity estimations provided by the Bloom Histograms, while improving their performance with respect to false positives by using the technique of splitting the paths to multiple levels that is used in Depth Bloom Filters.

We consider two variations of the new structure: the Depth Bloom Histogram (DBH) (Fig. 1(center)) and the Bloom Depth Histogram (BDH) (Fig. 1(right)). A DBH with \( j \) levels maintains a set of \( j \) Bloom Histograms. The Bloom Histogram for level \( i \) \( (i \leq j) \) is constructed using as input only the path expressions of length \( i \). For the BDH, we first construct the histogram based on the frequencies of all paths and then use a separate Depth Bloom filter instead of a simple Bloom filter for the paths in each of the buckets.

Based on the set of structural transformations and our index, our Relax Algorithm follows a dynamic programming approach for efficiently evaluating the top-\( k \) results at each superpeer. The algorithm takes a user query as input and gradually applies a combination of the available transformations to it until the user-specified number of results is attained. A central issue is the termination condition, that is, how much we need to generalize the query to get the required results. We derive the termination condition by exploiting the
distributed index. In particular, both the order and the number of transformations that the algorithm applies is driven by index selectivity estimations. At each step, the Relax Algorithm applies the transformation that results in a relaxed query with the smallest possible distance from the original query and with the most results as estimated by the index.

The distributed clustered index is incrementally constructed as new documents enter the system. Each document is assigned to a cluster based on its similarity with the cluster contents. We derive this similarity based on a similarity measure defined for the corresponding indexes, thus reducing both the communication and processing cost for the index construction.

III. EXPERIMENTAL EVALUATION

For evaluating performance, we use real data sets from the Niagara Project [6] that belong to 8 predefined thematic categories. There are two pairs of categories that share structural similarities, while there is little to no overlap among the rest. For query generation, we use the zipf distribution to select paths from the documents. To tune the number of buckets and their boundaries we apply the techniques presented in [4]. We use 4 buckets per level for both the DBH and the BDH to achieve a fair comparison. We configure the Depth Bloom filters parameters (BF size and number of hash functions) based on [5], limiting the levels to 3 resulting to a false positive ratio below 5%.

**Enhanced Bloom Histograms.** We compare the false positive ratio and the estimation error of the new structures (DBH/BDH) to the Bloom Histogram (BH) with respect to the space they occupy. We express this space as a percentage of the space a full path count table (PCT) requires to maintain the same data. The estimation error is defined following [4] as the average absolute error given by: \( \text{error} = \frac{1}{n} \sum_{i=1}^{n} |X_i - V_i| \), where \( n \) is the number of queries, \( X_i \) is the actual appearances of the query \( q_i \) in our data collection and \( V_i \) is the estimation returned by the index. Figure 2(left) shows that the enhancement of BH with multiple levels improves the error up to 50% for the same space overhead. This is because DBH/BDH query evaluation performs multiple checks for each query at different levels, whereas the BH performs just one. Since both DBH and BDH perform similarly for the same space overhead in the rest of our experiments we use the DBH as the indexing structure.

**Construction Cost.** To demonstrate the advantages of using our structures instead of a full path table, we measure the construction cost of the clustered index, as the total size of messages required, while varying the number of clusters (Fig. 2(center)). The deployment of the multi-level Bloom histograms reduces the construction cost of the clustered index from GBs to only a few MBs, while causing only a slight decrease to the pruning degree (Fig. 2(right)).

**Pruning Degree.** We compare the performance of a clustered and a non-clustered (random) index varying the number of clusters. Figure 2 (right) shows that for all number of clusters, the pruning degree is reduced in the case of clustering as expected. As we assign multiple categories to the same cluster (2 and 4 clusters), the pruning degree decreases for the clustered index, while it remains almost constant for the random index. This is because if documents from a category are distributed among different clusters then the relaxation algorithm at each superpeer produces results with similar quality. Using as many clusters as the document categories or less, alleviates this problem. On the other hand, having a small number of clusters increases the load at each cluster, especially for maintaining the index in the case of updates.

IV. CONCLUSIONS

In this paper, we address the problem of the efficient evaluation of XPath approximate queries over dynamic distributed collections of XML data. We focus on reducing the communication cost required for evaluating top-\(k\) queries in such settings, by introducing a distributed clustered path index which enables the nodes responsible for query evaluation to prune the number of candidate results they need to consider.

ACKNOWLEDGMENT

This paper is part of the 03ED591 research project, implemented within the framework of the “Reinforcement Programme of Human Research Manpower” (PENED) and co-financed by National and Community Funds (20% from the Greek Ministry of Development-General Secretariat of Research and Technology and 80% from E.U.-European Social Fund).
REFERENCES