Future Generation Computer Systems 37 (2014) 212–231

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Semantic-based Structural and Content indexing for the efficient retrieval of queries over large XML data repositories

Norah Saleh Alghamdi *, Wenny Rahayu, Eric Pardede

Department of Computer Science and Computer Engineering, La Trobe University, Australia

HIGHLIGHTS

- We exploit the semantics of XML Schema and data in building our index.
- We trim search space using an object-based intersection technique of large data.
- We eliminate irrelevant portions of data by discarding and irrelevant objects.
- We prove the efficiency based on measuring CPU Cost and the scalability.
- We show the high precision and recall of query results.

ARTICLE INFO

Article history:
Received 19 April 2013
Received in revised form
10 October 2013
Accepted 17 February 2014
Available online 11 March 2014

Keywords:
Query optimization
Index
Query processing
Objects
Semantic
XML
Twig query

ABSTRACT

The emergence of XML adoption as semi-structured data representation in multi-disciplinary domains has highlighted the need to support the optimization of complex data retrieval processing. In a Big Data environment, the need to speed up data retrieval processes has further grown significantly. In this paper, we have adopted an optimization approach that takes into consideration the semantics of the dataset in order to deal with the complexity of multi-disciplinary domains in Big Data, in particular when the data is represented as XML documents. Our method particularly addresses a twig XML query (or a branched path query), as it is one of the most costly query tasks due to the complexity of the join operation between multiple paths. Our work focuses on optimizing the structural and the content part of XML queries by presenting a method for indexing and processing XML data based on the concept of objects that is formed from the semantic connectivity between XML data nodes. Our method performs object-based data partitioning, which aims at leveraging the notion of frequently-accessed data subsets and putting these subsets together into adjacent partitions. Then, it evaluates branched queries through two essential components: (i) Structural and Content indexing, which use an object-based connection to construct indices i.e. Schema Index, Data Index and Value Index; and (ii) query processing to produce the final results in optimal time. At the end of this paper, a set of experimental results for the proposed approach on a range of real and synthetic XML data, as well as a comparative study with other related work in the area, are presented to demonstrate the effectiveness of our proposed method in terms of *CPU cost, matching and merging cost, scalability (size and number of branches) and total number of scanned elements*. Our evaluation demonstrates the benefit of the proposed index in terms of performance speed as well as scalability which is critical in a large data repository.

1. Introduction

The powerful ability of XML in describing and presenting data has been recognized as the standard for electronic data interchange in multi-disciplinary domains [1–3]. The metadata in XML documents provides a semantically rich structure which can be leveraged for various information system applications. The metadata also opens up opportunities to improve techniques to access and process XML data.

In this paper, we focus on processing XML queries efficiently by taking into consideration the semantic connectivity of the underlying XML documents. In particular, we focus on XML twig queries with or without value predicates. A twig query is a type of query which accesses XML trees with multiple branches and

* Corresponding author. Tel.: +61 421386433.
E-mail addresses: nalghamdi@students.latrobe.edu.au, ninasg11@hotmail.com (N.S. Alghamdi), w.rahayu@latrobe.edu.au (W. Rahayu), e.pardede@latrobe.edu.au (E. Pardede).

http://dx.doi.org/10.1016/j.future.2014.02.010
0167-739X/

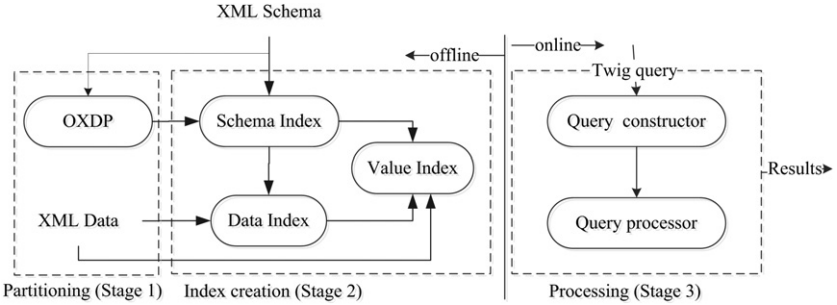

Fig. 1. The system configuration.

this query requires complex processing due to the joins between multiple paths.

A Naïve query processing by scanning the entire XML data to search for a particular path will cause significant performance degradation. Indexing schemes have been developed in recent years to overcome this issue. Different XML data indexing and query processing approaches have been proposed to support twig queries. The previous XML data indices were classified into three categories [4]. The first is path-based indices such as APEX [5] and MDFB [6], which group nodes in data trees based on local similarity and have an adjustable index structure depending on the query workload. These indices need to deal with a huge index size because its index keeps tracks of the forward and backward paths to establish a supportive layer that can help in answering twig queries effectively. The second is node-based indices, such as TwigX-Guide [7], which index the position of each node within the XML tree and then process the nodes by joining them when in some cases; aggressive joins deteriorate the query performance. The third is sequence-based indices such as ViST [8], PRIX [9] and LCS-Trim [10], which evaluate queries based on sequence matching after transforming both XML data and twig queries into sequences. However, the third approach has a drawback, which is the occurrence of false positive caused by sequence matching instead of tree matching. All the above work focus on the structural presentation of XML data (e.g. a sequence of nodes or a tree pattern) rather than the semantic relationship between groups of nodes (i.e. objects). Therefore, exploiting the semantic relationship between XML data nodes to build an index scheme for twig query processing is an ideal solution which has not been proposed in the existing literature as yet.

The work presented in this paper is the ongoing work of a research project on XML query optimization, which consists of three stages (see Fig. 1). The first two stages can be considered as off-line stages and the third is on-line stage. Stage one focuses on the object-based partitioning methodology of XML data. Stage two focuses on the XML data indexing methodology. Stage three focuses on the query processing method over indexed data. This paper presents the second and third stage in more detail and the methodology of the first stage of the Object-based XML Data Partitioning (OXDP) has been presented in [11,12]. Fig. 1 shows an overview of our system configuration.

Our proposed indices in the second stage consist of three main parts. The first part is the Schema Index; the second is the Data Index; and the third is the Value Index. The construction phases of the indices in the current stage are:

Phase 1: We build the Schema Index based on the object partitions identified by the OXDP process. We tokenize each distinct element tag to set up the Schema Index components.

Phase 2: We construct the Data Index components by grouping the XML data within object partitions and we establish keys from the Schema Index to the Data Index.

Phase 3: We then build the Value Index with knowledge of Schema and the Data Index.

Then, in the third stage of our system, we proposed a query processing method to handle indexed data. The query processor can evaluate simple XML paths as well as XML paths with branches and different value predicates.

This paper is organized as follows. Section 2 provides our motivation with a brief example that shows the utilization of the semantics of XML in indexing and processing XML data to improve the query performance. Also, it presents the importance of processing value predicates in XML queries. The related work is reviewed in Section 3. The preliminary knowledge is presented in Section 4. We introduce our proposed indices in Section 5 and discuss the processing method of XML queries in Section 6. Experimental results are provided in Section 7. Finally, we conclude the paper in Section 8.

2. Our motivation and contributions

While most of the existing work does not consider the semantics of XML data during the construction or processing of XML data, our work actually exploits the semantics connectivity between nodes to construct our indices. The incorporation of the semantic features of index construction into the XML query processing approach will lead to an efficient pruning technique. This is because the search space can be trimmed down to a group of data that follows certain semantics.

$Q1 =$ "*/purchaseOrder/ShipTo*[*city*][*state*]/*name*/*Fname*"

For instance, assume we have a purchase order schema as shown in Fig. 2 which describes a purchase order generated by home products ordering and a billing application [13] and we have Q1 as above. Let say that this schema has two object partitions: one includes a ShipTo element with its descendants and another includes a BillTo element with its descendants. Instead of accessing the two partitions to find the answer to Q1, it is ideal to retrieve the answer from a related portion of data. This action is possible if the index is constructed semantically. Therefore, we introduce the concept of objects in constructing and processing XML data. It will accelerate the processing of XML queries by localizing the XML data into a small and relevant portion of data.

In this paper, we tackle the issue of iteration on a large index size to find matched trees, or matched sequences by introducing the Schema Index. The Schema Index usually has a smaller index size compared to an index built only based on the data. This will help to find important keys within a small index size which is the Schema Index built from the schema to find the answer from the Data Index which is usually bigger since it is built from the data.

$Q1_a =$ "*/purchaseOrder/ShipTo*[*city*
$=$ '*Melbourne*'*and state* $=$ '*VIC*']/*name*/*Fname*".