Information Systems 52 (2015) 21–33

Contents lists available at ScienceDirect

Information Systems

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

Cost-based holistic twig joins[☆]

Radim Bača[*], Petr Lukáš, Michal Krátký

Department of Computer Science, Faculty of Electrical Engineering and Computer Science, VŠB – Technical University of Ostrava, Czech Republic

A R T I C L E I N F O

Article history:
Received 10 December 2014
Received in revised form
20 March 2015
Accepted 21 March 2015
Recommended by: D. Shasha
Available online 31 March 2015

Keywords:
XML
Query processing
Cost-based approaches
Holistic algorithms
Access path

A B S T R A C T

An evaluation of XML queries such as XQuery or XPath expressions represents a challenging task due to its complexity. Many algorithms have been introduced to cope with this problem. Some of them, called binary joins, evaluate separated parts of a query and subsequently merge intermediate results, while the others, called holistic twig joins, evaluate a query as a whole. Moreover, these algorithms also differ in what index data structure they use to handle XML data. There exist cost-based approaches utilizing binary joins and various index data structures; however, they share a limitation. The limitation is that they cannot perform a join between query nodes not having a direct XPath relationship. Such a join can be advantageous especially if their joint selectivity is high. Since holistic joins work with all query nodes they overcome this limitation. In this article, we introduce such a holistic twig join called CostTwigJoin. To the best of our knowledge, CostTwigJoin is the first holistic join capable of combining various index data structures during an evaluation of an XML query. Usage of the holistic join has yet another advantage for cost-based approaches: an optimizer does not have to resolve the order of binary joins; therefore, the search space is reduced. In this article, we perform thorough experiments on hundreds of queries to evaluate our approach and demonstrate its advantages.

1. Introduction

All existing cost-based approaches in XML query processing are based on a selection of an appropriate algebraic execution query plan according to a cost model [36,16,42,12,35]. Such a cost-based approach can significantly improve the performance of XML query processing, especially for highly selective queries (i.e. queries producing small results) that are very common in practice. The above-mentioned works pick one out of two join algorithms for each algebraic operator and resolve their ordering. There are two basic access paths used in join algorithms: (1) *merge access path* [1,41,39], where the basic idea is based on a merge of input sequences, and (2) *navigational access path* [14–16], where a number of DOM-like operations are processed using the previous intermediate results.

In spite of the effectiveness of existing cost-based approaches, they work only for *binary joins* [1,41,39,15]. The binary joins work only with a pair of query nodes. However, there exists a large family of algorithms called *holistic joins* [5–7,21,22] working with all query nodes simultaneously and having two major advantages when compared to binary joins: (1) the intermediate result is minimized even without sophisticated query optimizations [5], (2) their optimality has been proved for certain query types [5,2]. In [34], a thorough comparison of binary and holistic join algorithms can be found, where both use the merge access path. The conclusion of this work is that the holistic join outperforms the binary join if the selectivity of a query is high. Note that high selectivity is important for

☆ This work is partially supported by SGS, VŠB – Technical University of Ostrava, No. SP2014/211 and No. SP2015/170, Czech Republic.

* Corresponding author.
E-mail addresses: radim.baca@vsb.cz (R. Bača), petr.lukas@vsb.cz (P. Lukáš), michal.kratky@vsb.cz (M. Krátký).

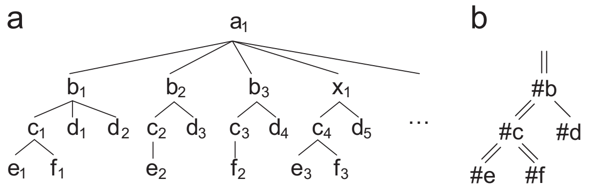

Fig. 1. (a) XML tree and (b) TPQ.

a significant improvement of all cost-based approaches. On the other hand, all the existing holistic joins are based only on the merge access path which significantly limits their usage in cost-based approaches.

To overcome this problem, we introduce a holistic join algorithm supporting both access paths. To demonstrate this idea, we extend the state-of-the-art holistic algorithm GTPStack [2] and call this extended version CostTwigJoin. Before the contribution of this paper is described in more detail, let us first mention some preliminaries.

XPath or even XQuery are robust languages supporting versatile semantics and we are focused mainly on searching XML nodes satisfying defined relationship constraints, and therefore, a simplified model of a query should be used. *Twig Pattern Query* (TPQ) is the simplest model used by many approaches [1,5,21,33,41]. It is naturally modeled as a tree, where each node corresponds to an XPath step. Single and double lined edges correspond to parent–child (PC) and ancestor–descendant (AD) relationships, respectively. A formal definition of TPQ is given in Section 3.

Example 1.1. Consider the following XPath expression:

//b[.//c [.//e and .//f]]/d.

The corresponding TPQ can be found in Fig. 1(b).

The problem of evaluating an XPath or an XQuery expression is then often transformed into a problem of searching for *TPQ matches*. A query match is a tuple of XML nodes satisfying the following conditions: (1) every XML node corresponds to exactly one TPQ query node, and (2) relationships between XML nodes correspond to relationships between query nodes.

Many existing algorithms [1,2,5,13,41] are designed so that an inverted list with XML nodes is used in order to support an efficient merge (i.e. they use the merge access path). In such algorithms, lists of XML nodes are accessed according to the tag name and a merge join is performed during a sequential scan of the lists. This technique has been shown to be very efficient and robust in many situations. Moreover, holistic twig joins can use the inverted list more efficiently and with less memory requirements than binary joins can [25]. However, in some cases both approaches can spend a lot of time reading and processing many irrelevant XML nodes from the inverted list, especially if a query contains a highly selective part.

Example 1.2. When searching all matches of the TPQ in Fig. 1(b) in the XML tree in Fig. 1(a), we retrieve two query matches

$[b_1, c_1, e_1, f_1, d_1]$ and $[b_1, c_1, e_1, f_1, d_2]$;

however, there are significantly more XML nodes read from the inverted list. That is caused by the fact that we read all XML nodes corresponding to each tag.

The problem of reading many irrelevant XML nodes is addressed by several existing cost-based approaches [36,16,42,12,35]. However, the high selectivity of a query is sometimes determined by a join of query nodes not having any direct XPath relationship (we call them *distinct query nodes*). Since the existing cost-based approaches do not consider merge joins between distinct query nodes, it can happen that the most selective join is not considered by any cost-based approach.

Example 1.3. In the previous example, we see that the high selectivity of the result is determined by a joint occurrence of #e, #f, and #b query nodes not having a direct XPath relationship. The existing approaches create a query plan, where merge joins of #e and #c, #f and #c, and #b and #c are considered. However, it means that many XML nodes corresponding to #c are unnecessarily accessed in all of these joins. Clearly, it is more efficient to merge lists corresponding to the distinct #e, #f, and #b query nodes first since we find just one triplet (b_1, e_1, f_1). The b_1 node in this triplet is then used to find the remaining nodes c_1, d_1, and d_2 by a simple scan of the b_1's subtree, i.e. the navigational access-path is used.

In this article, we show how to choose a merge of the highly selective query nodes regardless of whether there is a direct XPath relationship between the query nodes or not. This can be achieved since holistic joins process all query nodes during a single run. Therefore, in this article, we extend an existing holistic join algorithm to integrate both access paths.

Now let us summarize the contributions of this article: (1) an introduction of the CostTwigJoin algorithm that is capable of using merge and navigational access paths during query processing and that minimizes the intermediate result size using a merge of lists corresponding to distinct query nodes, (2) a cost-based optimization framework to select an appropriate index data structure for processing each query node, (3) thorough experiments showing the main advantages of our approach for a large number of queries and various data collections.

The paper is organized as follows. Section 2 analyzes related works. In Section 3, we depict a model of an XML document and the supported query constructs. Section 5 introduces the CostTwigJoin algorithm and its features. In Section 6, we thoroughly describe our cost-based optimization framework. Section 7 experimentally verifies the advantages of our approach.

2. Related work

Various TPQ processing approaches have been developed [1,5–7,21,22,33,41,13,2].

Let us start with techniques orthogonal to our approach which can significantly reduce the number of irrelevant nodes read from indices. The first type of these techniques uses a summary tree for more fine-grained access to the inverted list [7,3]. In other words, these techniques use different key types of inverted lists (called *partition labels*).