Efficient Evaluation of XML Path Queries with Automata

Bing Sun, Jianhua Lv, Guoren Wang, Ge Yu, and Bo Zhou

Northeastern University, Shenyang, China
{sunb,dbgroup,wanggr,yuge}@mail.neu.edu.cn

Abstract. Path query is one of the most frequently used components by the various XML query languages. Most of the proposed methods compute path queries in instance space, i.e. directly facing the XML instances, such as XML tree traversal and containment join ways. As a query method based on automata technique, automata match (AM) can evaluate path expression queries in schema space so that it allows efficient computation of complex queries on vast amount of data. This paper introduces how to construct query automata in order to compute all regular expression queries including those with wildcards. Furthermore, a data structure named schema automata is proposed to evaluate containment queries that are very difficult from the conventional automata point of view. To improve the efficiency of schema automata, methods to reduce and persistent them are proposed. Finally, performance study of the proposed methods are given.

1 Introduction

Most XML query languages \cite{2,3,5,6,12} use path queries as their core components. The input of path queries are path expressions (PEs). PEs can be divided into steps connected by path operators. Different operators have different properties, and the main target of path query systems is to compute these operators correctly and efficiently.

XML tree traversal is a basic and simple way \cite{11}. It tests all XML tree nodes one by one in certain order to see if they are instances of the given PE. This way is effective but not efficient. There are some improved methods upon tree traversal. They can reduce the searching space of traversal and improve performance. However, since all nodes on pathes leading to candidate results must be accessed at least once, it is not expected to be an efficient way. Another popular method is containment join (CJ) \cite{4,8}. It gives each XML node a code as its id. Codes of instances corresponding to PE steps are gotten from indices, and they are joined up to compute the path operators between them. CJ is well designed to evaluate containment queries, which are queries containing ancestor-descendant operator ("//"). CJ is an efficient method and it can be easily implemented in relational database systems. However, it does not support closure operators ("*", "+").

Methods discussed above are facing XML instances. Tree traversal ways access instances to acquire their children, siblings or attributes; CJ ways join up
two instance sets and form a new one. These methods are regarded as executing in instance space. Speed of these methods is highly relevant to the amount of instances they process. Therefore, if they run on large scale XML documents, the efficiency will extremely decrease. The structure of an XML document is usually defined by DTD or XML schema, and their structures are much more stable than that of XML documents. Thus, if there is a method that runs in schema space, it will performs well much better on large documents. Automata match (AM) is such a method. It builds a path schema tree (PST) as index, and converts PEs to query automata. Then, the nodes on PST are matched with the status of query automata. Finally, the results are gotten on the nodes that matches the accepting status. AM does not deal with XML instances directly except results retrieval. Experimental results prove that it is efficient on large documents. AM approach has been proposed first in [7], but it does not describe how to evaluate ancestor-descendant operator. This question are fully resolved in this paper.

Most path query system allows PEs being arbitrary complex by nested structure. Moreover, some special path operators are very time-consuming in common ways, so the complexity of the PEs can extraordinarily affect the efficiency of query processing. These operators need to be studied carefully. Closure operators defined in regular expressions are useful and important for querying nested structured XML data. They are very difficult to evaluate by ordinary methods for most systems including RDBMS do not support them. Rewriting them into repeated joins is a feasible but time-consuming way. Since there always exists an equivalence automata for each regular expression, closure operators can be convert to automata components naturally. Therefore, AM can easily deal with closure operators.

Altneu and Franklin proposed an algorithm to perform efficient filtering of XML document in the selective information dissemination environments [1]. Automata were used to denotes user profiles and a SAX interfaced parser was used to activate the automata when parsing documents in there approach. They were facing XML documents directly while AM is facing schemas of XML documents for better performance.

Ancestor-descendant operator ”//” is another operator widely used in path expressions, which is a non-regular operator. This operator is used to find all descendant XML nodes of the context nodes. It can also be considered as a “wildcard” of any path expressions that may occur there. Several ways has been proposed to compute it. Rewriting this operator into path expressions is a usual and universal way. However, if the document structure is very complex, the expressions generated will be extremely complicated or even it is impossible to generate equivalent expressions. CJ stands for a series of algorithms designed to compute ”//”. Ancestor nodes and descendant nodes are retrieved into two lists respectively, and the two lists are joined using certain algorithms. CJ is effective, and it is suitable to compute queries like ”a//b” with the support of indices. However, its speed depends on the amount of data being joined. Therefore, if there are wildcards in PEs, like ”a//*”, the speed will be extremely lower. AM