Parameterized XPath Views

Timo Böhme and Erhard Rahm

Database Group
University of Leipzig
{boehme,rahm}@informatik.uni-leipzig.de

Abstract. We present a new approach for accelerating the execution of XPath expressions using parameterized materialized XPath views (PXV). While the approach is generic we show how it can be utilized in an XML extension for relational database systems. Furthermore we discuss an algorithm for automatically determining the best PXV candidates to materialize based on a given workload. We evaluate our approach and show the superiority of our cost based algorithm for determining PXV candidates over frequent pattern based algorithms.

1 Introduction

With XML as the lingua franca for data exchange and an increasingly popular storage format for structured data there is a growing demand for natively storing and querying of XML. Consequently native XML database systems evolve and relational database systems have been augmented with XML support. Query optimization is a main challenge for these systems due to the high flexibility and ordered structure of XML and the complexity of its query languages.

XPath is a crucial component of XML query languages such as XQuery or XSLT and thus has been an essential part for improving query performance. Work on this topic ranges from indexing techniques [15, 21, 22], structural join algorithms [3, 12], containment, equivalence and intersection of XPath expressions [10, 13] to cardinality estimation [20, 25]. Despite the large amount of work on XPath processing, running complex queries on large XML data sets is still a challenge. Moreover, several of the proposed algorithms are not applicable in certain environments like implementations using a relational database system (RDBS) back-end for storing the XML tree structure.

It was shown that caching techniques [4, 14] and materialized views [1, 18, 23] could be used to address these performance problems. However we found that the proposed solutions were not flexible enough to adapt to specific workloads. We therefore propose to enhance the materialized view approach in two directions. First, we parameterize the view definition in order to use materialized views for queries with different comparison values. Second, our views contain extra information to efficiently use them as a replacement for query fragments which do not start at the query root.

With the enhanced flexibility of our views a manual selection of the most profitable views to materialize for a given workload, database and space constraint is not feasible. We therefore developed a method to automate this important decision

R. Cooper and J. Kennedy (Eds.): BNCOD 2007, LNCS 4587, pp. 125–137, 2007.

process and show how this can be implemented in an XML extension for RDBS called XMLRDB.

The rest of the paper is organized as follows. Next we discuss related work. Section 0 details our enhancements to materialized views called PXV. The integration of PXVs in XMLRDB is described in Section 4. In Section 5 we present our method for automatically determining the most valuable view candidates to materialize. Section 6 evaluates experimentally the performance gains obtained by employing PXVs. Finally Section 7 concludes the paper.

2 Related Work

Grust et al. proposed efficient implementations for XPath [7] and XQuery [8] based on a RDBS with a generic storage of XML data. The most efficient variant utilized a specific numbering scheme as well as a special join operator. A general problem of RDBS usage is that they need many expensive join operations for complex XPath expressions (see Section 0). This also holds for the related work on XQuery-to-SQL translation. The optimizations proposed in the present paper can complement these previous approaches.

A general framework for materialized XPath views is described in [1]. The views may contain XML fragments, typed data values, references to nodes in the actual data and full paths. The paper covers the XPath query rewriting process. Our work differs from this paper in the following points. We enhanced the view concept by parameterizing comparison values and added information for simplified view application. Furthermore we propose an algorithm to automatically determine valuable views to materialize.

[14] creates materialized views on the fly for query caching. The sampled workload is parameterized on comparison values. Each view stores its data as an XML fragment. Only the information which parameter values were used to build the fragment are kept. Therefore if comparison predicates are used in a query which should be answered by a view, the view must be pruned by a compensation query. Since the view only contains the result of its defining query q_v it is only possible to restrict on predicates of the last step in q_v because the corresponding node is the root node of the stored XML fragment.

Materialized XML views are used to improve performance of an XML interface of a RDBS in [18]. Instead of translating each query to SQL and transforming the relational result to XML it caches frequently accessed data as materialized XML views. This approach differs greatly from ours, as it is based on relational data whereas we depend on XML node based storage.

Query rewriting using views has been extensively discussed for RDBS [9]. Later this problem was studied for semistructured data [6, 17] and recently it was examined for the XML domain with the specialities of the XML data model and XML query languages. [11, 23] focus on subsets of XPath for polynomial time algorithms. [16] covers query rewriting using XQuery based views. In our approach we focus on a query rewriting to find an identical match (cf. Fig. 1) of the view definition within the query which can be achieved in $O(\#_{steps}(q) \cdot \#_{steps}(q_v))$ time complexity.

Finding frequent XML query patterns as candidates for caching or materialization is targeted in [24]. The proposed algorithm FastXMiner finds frequent query patterns