Chapter 8
A Framework for Cost-Based Query Optimization in Native XML Database Management Systems

Andreas M. Weiner
University of Kaiserslautern, Germany

Theo Härder
University of Kaiserslautern, Germany

ABSTRACT

Since the very beginning of query processing in database systems, cost-based query optimization has been the essential strategy for effectively answering complex queries on large documents. XML documents can be efficiently stored and processed using native XML database management systems. Even though such systems can choose from a huge repertoire of join operators (e. g., Structural Joins and Holistic Twig Joins) and various index access operators to efficiently evaluate queries on XML documents, the development of full-fledged XML query optimizers is still in its infancy. Especially the evaluation of complex XQuery expressions using these operators is not well understood and needs further research. The extensible, rule-based, and cost-based XML query optimization framework proposed in this chapter, serves as a testbed for exploring how and whether well-known concepts from relational query optimization (e. g., join reordering) can be reused and which new techniques can make a significant contribution to speed-up query execution. Using the best practices and an appropriate cost model that will be developed using this framework, it can be turned into a robust cost-based XML query optimizer in the future.

INTRODUCTION

In the last few years, XML became the de-facto standard for exchanging structured and semi-structured data in business as well as in research. The database research community took this development into account by proposingamong others—native *XML database management systems* (XDBMSs) for efficient and transactional processing of XML documents.

DOI: 10.4018/978-1-61520-727-5.ch008

As in the relational world, the quality of query optimizers plays an important role for the acceptance of database systems by a wide range of users, especially in business scenarios where longer-than-necessary running queries can cause high costs. One of the main tasks in query evaluation is plan generation, where physical operators are arranged in such a way, that the given optimization goal (e. g., maximum throughput) is satisfied while the semantics of the query is still preserved.

In recent years, several join operators for the evaluation of structural relationships like *child* or *descendant* have been proposed. All of them belong to one of the major classes of XML join operators: *Structural Joins (SJs)* (Al-Khalifa et al., 2002) and *Holistic Twig Joins (HTJs)* (Bruno, Koudas, & Srivastava, 2002). Being binary join operators, SJ operators decompose tree-structured query patterns, which are also called twig query patterns, into binary relationships and evaluate each of them separately, before they merge intermediate results to get the final query result. On the other hand, HTJ operators are able to evaluate twigs holistically. A precondition for the efficient evaluation of SJ and HTJ operators is a node labeling scheme (O'Neil et al., 2004; Härder, Haustein, Mathis, & Wagner, 2007) that assigns to each node in an XML document a unique identifier that (1) allows to decide, without accessing the document, for two given nodes whether they are structurally related to each other and (2) that does not require re-labeling even after modifications to the document.

Besides SJ and HTJ operators, several approaches for indexing XML documents were proposed. These approaches can be classified into a hierarchy of access methods w. r. t. their availability in a native XDBMS. *Primary access paths (PAPs)* provide input for navigational primitives as well as for SJ and HTJ operators. The most important representative of this class is a *document index* that indexes a document using the unique node labels as keys. *Secondary access paths*[1] (SAPs) provide more efficient access to specific element nodes using *element indexes* (Bruno et al., 2002). They are absolutely necessary for efficient evaluation of structural predicates by SJ or HTJ operators. *Tertiary access paths (TAPs)* like *path indexes* (Milo & Suciu, 1999) employ structural summaries such as *DataGuides* (Goldman & Widom, 1997) for providing efficient access to nodes satisfying structural relationships like *child* or *descendant*. *Content indexes* (McHugh & Widom, 1999) support efficient access to text nodes or attribute-value nodes. Finally, *hybrid indexes* (Wang, Park, Fan, & Yu, 2003), which are also called *content-and-structure (CAS) indexes*, are a promising approach for indexing content and structure at a time. Compared to PAPs, which are available per default in a native XDBMS, SAPs and TAPs have to be manually created by the database administrator. Furthermore, TAPs like path indexes or CAS indexes can replace complete trees of SJ and HTJ operators and become—if available—first-class citizens for query evaluation. As maintenance and updates of TAPs can cause substantial overhead, they will only be created by the database administrator in rare cases, e.g., for frequently queried subtrees.

Motivation

If we have a look at the status quo of native XML query evaluation, several unsolved problems arise. In the presence of a rich variety of physical operators, most of all empirical evaluations were performed using self-generated or slightly modified well-known documents with simple queries. By doing so, the superiority of a particular operator was shown and probable weaknesses were suppressed. Today, it is unknown how these operators really behave in realistic native XDBMS scenarios. For example, real-world XQuery expressions are provided by the XMark benchmark queries (Schmidt et al., 2002) or by