VLDB Journal manuscript No.
(will be inserted by the editor)

Efficient Support for Complex Queries on Multiversion XML Documents

Shu-Yao Chien[1], **Vassilis J. Tsotras**[2] ⋆, **Carlo Zaniolo**[3], **Donghui Zhang**[4]

[1] NCR/Teradata Division

[2] CSE Dept., UC Riverside

[3] CS Dept. UCLA

[4] *communicating author*, CS College, Northeastern Univ., donghui@ccs.neu.edu, phone: 1-617-373-2177, fax: 1-617-373-5121.

Received: date / Revised version: date

Abstract Managing multiple versions of XML documents represents a critical requirement for many applications. Also, there has been much recent work on supporting complex queries on XML data (e.g., regular path expressions, structural projections, DIFF queries). In this paper, we examine the problem of implementing efficiently complex queries on multiversioned XML documents. Our approach relies on a scheme based on durable node numbers (DNNs) that preserve the order among the XML tree nodes and

⋆ This work was partially supported by NSF grants IIS-9907477, EIA-9983445, and the Department of Defense.

are invariant with respect to updates. Using the document's DNNs complex queries are reduced to combinations of *partial version retrieval* queries. We examine three indexing schemes to efficiently evaluate partial version retrieval queries in this environment. A thorough performance analysis is then presented to reveal the advantages of each scheme.

1 Introduction

The management of multiple versions of XML documents finds important applications [33] and poses interesting technical challenges. Indeed, the problem is important for application domains, such as software configuration and cooperative work, that have traditionally relied on version management. As these applications migrate to a web-based environment, they are increasingly using XML for representing and exchanging information—often seeking standard vendor-supported tools and environments for processing and exchanging their XML documents.

Many new applications of versioning are also emerging because of the web; a particularly important and pervasive one is assuring link permanence for web documents. Any URL becoming invalid causes serious problems for all documents referring to it—a problem that is particularly severe for search engines that risk directing millions of users to pages that no longer exist. Replacing the old version with a new one, at the same location, does not cure the problem completely, since the new version might no longer contain the keywords used in the search. The ideal solution is a version

management system supporting multiple versions of the same document, while avoiding duplicate storage of their shared segments. For this reason, professionally managed sites and content providers will have to use document versioning systems; frequently, web service providers will also support searches and queries on their repositories of multiversion documents. Specialty warehouses and archives that monitor and collect content from web sites of interest will also rely on versioning to preserve information, track the history of downloaded documents, and support queries on these documents and their history [22].

The problem of version management has not originated with the web or XML; in fact it has been studied extensively in the context of document management systems and software configuration management, for which techniques such as RCS [30] and SCCS [25] have been introduced. Various versioning techniques have also been proposed by database researchers who have focused on problems such as transaction-time management of temporal databases [24], support for versions of CAD artifacts in O-O databases [18] and, more recently, change management for semistructured information [7].

However in the past, database systems and document management systems were faced with different versioning problems since:

– Database systems are designed to support complex queries, while document management systems are not, and

– Databases assume that the order of the objects is not significant—but the lexicographical order of the objects in a document is essential to its reconstruction.

This past state of affairs has now been changed by XML that merges applications, requirements and enabling technology from the two areas. Indeed the differences mentioned above are fast disappearing since support for complex queries on XML documents is critical. This is demonstrated by the amount of current research on this topic [27,29] and the emergence of powerful XML query languages [1,15,5,13,6,14]. A particularly challenging problem is that of supporting efficiently path expression queries, which are discussed next.

2 Problem Definition

The following path expression query specifies figures that are immediate elements of chapters or their transitive sub-elements (e.g., figures in subsections)"

$$\texttt{doc/chapter/} * \texttt{/figure}.$$

Durable numbering schemes have been proposed to support these queries efficiently [16,23,20]. Durable node numbers are used to represent the document structure using numbers which do not change when the document is updated.

For multiversion documents, we have to support such complex queries on any user-selected version. Furthermore, we need to support difference queries between two versions, and queries on the evolution of documents or selected parts of it, such as for lineage queries.

In [8] and [9] we proposed schemes for the efficient storage and retrieval of multiversion documents and showed that these provide significant improvements with respect to traditional schemes such as RCS [30] and SCCS [25]. To enhance the version retrieval efficiency, [8] places document elements in disk pages using a clustering mechanism called UBCC (for Usefulness Based Copy Control). The UBCC mechanism achieves better version clustering by copying elements that live through many versions. A variation of UBCC was used in [9], where a reference-based versioning scheme was presented.

While the versioning schemes proposed in [8,9] are effective at supporting simple queries, they cannot handle complex queries such as the path-expression queries. For complex queries, we have recently outlined [10] the $SPaR$ scheme that adapts the *durable node numbers* [20] to a multiversion environment. Furthermore, SPaR uses *timestamping* to preserve the logical structure of the document and represent the history of its evolution. In this paper, we expand the properties of the $SPaR$ scheme and investigate efficient physical realizations for it. Different storage and indexing strategies are examined so as to optimize SPaR's implementation. Our study builds on the observation that evaluating complex version queries mainly depends on the efficiency of evaluating one basic type of query: the *partial version*

retrieval query. Such query retrieves a specific segment of an individual version instead of the whole version. Retrieving a segment for a single-versioned XML document is efficient since the target elements are clustered on secondary store by their logical order, but this might not be the case for a multiversion document. For a multiversion document, a segment of a later version may have its elements physically scattered in different pages due to version updates. Therefore, retrieving a small segment could require reading a lot of unnecessary data.

While UBCC is very effective at supporting full version retrieval queries, complex queries on content and history combined call for indexing techniques such are the Multiversion B-Tree [21,3,32] and the Multiversion R-tree [19]. We investigate the following three approaches:

Scheme 1: single Multiversion B-Tree,

Scheme 2: UBCC with a Multiversion B-Tree, and

Scheme 3: UBCC with a Multiversion R-tree.

The last two approaches still use the UBCC mechanism as the main storage scheme for the document elements. The additional indices are used as secondary indices so that partial version retrievals are efficiently supported. The first approach lets the Multiversion B-Tree organize the document elements in disk pages and at the same time uses the index for partial retrievals. The Multiversion B-Tree also uses a clustering technique. However, this technique is more elaborate and uses more disk space, since it clusters

by versions and (durable) element numbers. A performance evaluation is presented to compare the different schemes.

The main contributions presented in this paper are as follows:

- We propose techniques for reducing complex queries over a multi-versioned XML documents to partial version retrieval queries, and
- We present an in-depth study of the problem of implementing efficiently partial version retrieval queries, and investigate the pros and cons of alternative approaches though extensive experiments.

These results extend and improve those presented in [11] in several ways discussed in the body of the paper. In particular we solve the range deletion issue, and incremental update issue, thus providing efficient and practical support for deletion and incremental updates. The new experiments presented include performance comparison for full-version retrieval, single-element retrieval, and the version-interval query.

The rest of this paper is organized as follows. Section 3 provides background, while section 4 presents the SPaR scheme. In section 6, the three storage and indexing combinations are described. Their performance is presented in section 7 while conclusions appear in section 8.

3 Version Management

A new document version (V_{j+1}) is established by applying a number of changes (object insertions, deletions or updates) to the current version (V_j). In a typical RCS scheme, these changes are stored in a (forward) edit script.