Notes on "DeweyIDs—The Key to Fine-Grained Management of XML Documents"

Theo Härder and Christian Mathis

Referring to the implementation and optimization of *XTC* as our prototype XDBMS, we discuss the experiences we have made with the application of DeweyIDs since the publication our original paper. For this reason, we highlight the areas where DeweyIDs are the source of fine-grained management, processing flexibility, and performance drivers in XDBMSs. The detailed results and progress accomplished can be found in the referenced literature.

Categories and Subject Descriptors: Information Systems [**Miscellaneous**]: Databases

Additional Key Words and Phrases: Tree node labeling; Dewey order; XML document storage;

1. INTRODUCTION

There was a life before the invention of prefix-based node labeling for XML documents. But there is *definitely a better life* which can directly be accredited to the use of DeweyIDs (or OrdPaths [O'Neil et al. 2004]) for various XML processing tasks. In the following, we sketch the most important features where we made substantial progress in the optimization of *XTC* as a full-fledged XDBMS.

2. FUNCTIONALITY OF NODE LABELING

Initial research on XML processing focused on navigation and retrieval in static documents which did not raise requirements such as node labeling schemes immutable under arbitrary updates or those supporting path-matching operations. Therefore, simple and straightforward proposals such as sequential numbering schemes only guaranteed *uniqueness* and *order preservation* of node labels. In the sequel, various range-based node labeling schemes were considered the prime candidates for XDBMSs, because their labels directly enable *testing of all (important) XPath axes*. Until today, their missing support of dynamic XML documents is ignored by quite a number of researchers—based on range-based schemes, they still develop solutions which do not meet the state of the art of XML processing anymore.

As XML processing has entered the realm of full-fledged, widely-used database products, flexible handling of dynamic XML documents and their transaction-protected, fine-grained manipulation in multi-user environments are indispensable. Hence, a suitable labeling scheme has to satisfy further XDBMS-specific criteria —it has to guarantee in dynamic XML documents *immutable labels* (under heavy updates/insertions) and has, for each document node, to enable the reconstruction of *all ancestor labels* without accessing the document. This property greatly supports for a context node *cn* intention locking on the entire path to the root and path matching for query processing, e. g., in case of twig queries.

While range-based schemes would need for these services further index access or similar deviation, i. e., overly expensive look-ups in the disk-based document, only prefix-based node labeling can support all desired labeling properties *without the need of document access*. Each label based on *Dewey Decimal Classification* directly represents the path from the document's root to the related node and the local order w. r. t. the parent node. Some specific variations of the Dewey numbering scheme such as OrdPaths, DeweyIDs, or Dynamic Level Numbering (DLN) are equivalent for all XDBMS tasks and mainly differ only in the way how they provide immutable labels by supporting an overflow technique for dynamically inserted nodes. Nowadays, the Dewey scheme is used in all major DBMS products and it definitely embodies the core concept to achieve processing performance [Härder et al. 2010].

3. IMPLEMENTATION OF NODE LABELS

Because DeweyIDs tend to be space-consuming, suitable encoding and compression of them in DB pages is a *must*. Effective encoding of (the divisions of) DeweyIDs at the bit level may be accomplished using Huffman codes [Härder et al. 2007]. It is important that the resulting codes preserve their order when compared at the byte level. Otherwise, each comparison, e. g., as keys in B*-trees or entries in reference lists, requires cumbersome and inefficient decoding and inspection of the bit sequences. Because such comparisons occur extremely frequent, schemes such as *Quaternary String* codes [Li et al. 2002] violating this principle may encounter severe performance problems.

When DeweyIDs are stored in document sequence, they lend themselves to prefix-compression and achieve impressive compression ratios. Our experiments using a widely known XML document collection [Miklau 2002] confirmed that prefix-compression reduced the space consumed for dense and non-dense DeweyIDs orders down to ∼15 – ∼35% and ∼25 – ∼40%, respectively [Härder et al. 2007].

On the other hand, DeweyIDs' variable length and prefix compression applied cause some performance problems. Because binary search is not possible on variable-length DeweyIDs inside a database page, each node reference implied a search from the beginning of the (B*-tree) page which required on average to reconstruct half of the DeweyIDs contained. Although a main-memory operation, performance suffered in case of frequent accesses, e. g., when the query result had to be materialized. Caching of pages with uncompressed DeweyIDs relieved such situations. Another performance penalty was decoding of DeweyIDs for specific DeweyID operations. Therefore, we currently look for solutions enabling direct processing of the most important/frequent operations on the encoded byte representations.

4. PATH-ORIENTED XML DOCUMENT STORAGE

The usage scope and flexibility of DeweyIDs was further increased by two concepts: *path synopsis* and *path class references (PCRs)*. A path synopsis [Mathis 2009] is an *unordered* structural summary of all (sub)paths of the document. Each non-content node belongs to a path class which represents all path instances having the same sequence of ancestor labels. To facilitate the use of path classes, we enumerate them with so-called PCRs that serve as a simple and effective path class encoding.

The synopsis size depends on the structural complexity of the document. It can usually be stored on a single page (or few consecutive pages) on external storage. For fast access, it should reside in a small data structure kept in main memory.

We want to emphasize the expressiveness of DeweyIDs and PCRs: a DeweyID delivers all DeweyIDs of its ancestor path, while a PCR delivers—by means of the path synopsis—the element and attribute names of the ancestor path. Together, they form a kind of *coordinate system* for the XML document. Starting from an arbitrary node whose DeweyID and PCR is known, it is easy to reconstruct the complete path instance the node belongs to. As an example, refer to Fig. 1 of [Haustein et al. 2005] (we skipped the text nodes in the taDOM tree): Assume index entry ("TCP/IP...", 1.3.3.3.1, 5) references the XML fragment where the path synopsis labels the corresponding text node with PCR=5. Then, without document access, computation of the entire path `/bib/book/title/''TCP/IP...''` together with all labels is straightforward. Because a DeweyID contains the labels of all ancestor nodes, all DeweyIDs along the path to the document root are delivered for free and are immutable under document updates (see Section 2). For these reasons, we also use the generic name *stable path labeling identifiers* (SPLIDs) for them.

Using this mechanism, we can recompute for any context node its entire path to the root. Therefore, it turned out to be sufficient to only store the leaf (content) nodes of a document together with their DeweyIDs and PCRs. Hence, we could "virtualize" its structure part. The resulting physical document representation is called *path-oriented storage* for which we can guarantee the operational semantics of navigational and declarative XML languages. To evaluate any XPath axis, a single traversal of our B*-tree-based document storage is sufficient. At the same time, substantial I/O is saved when selected document nodes are accessed or the entire document is reconstructed [Härder et al. 2010].

5. FLEXIBILITY OF INDEXING

Similar to path-oriented document storage, the combined use of path information and DeweyIDs enables a flexible and focused indexing mechanism. When index entries are equipped with DeweyIDs, their increased functionality greatly increases the processing effectiveness of conventional *element*, *content*, and *path* indexes on XML documents. *Content-and-structure* (CAS) indexes can take even more advantage: By using PCRs in the index' reference lists, we could build *unique*, *collective*, and *generic* indexes which greatly facilitate their tailor-made usage, enable their adaptation to given workloads, and offer various clustering options [Mathis et al. 2009]. When these index types are applied to documents with virtualized structure, even the non-existing structure nodes can be indexed and processed without any loss of information (using the mechanism exemplified in Section 4).

6. EFFICIENCY OF QUERY PROCESSING

We have shown that all evaluation algorithms for *Structural Joins* and *Holistic Twig Joins* can be effectively implemented using DeweyIDs. Because of the direct and fast resolution of parent and ancestor labels, element and path indexes can efficiently support path queries. CAS indexes support simple path queries as well as twig queries. Twig queries use combinations of path specifications and optionally

content predicates. The evaluation of them by using simple element or path indexes is much more cumbersome and less efficient than their evaluation by (tailor-made) CAS indexes. Furthermore, they are as space-efficient as conventional content indexes, but they can easily be focused to query-relevant paths, if necessary. Hence, they are superior in terms of time and space [Mathis 2009].

7. CONCURRENCY CONTROL

In [Haustein et al. 2005], we claimed that the computation of all ancestor labels was particularly helpful for intention locking in XML document trees. Experiments with our initial taDOM2 protocol revealed "expensive" weaknesses when descendant nodes had to be locked. Leading to severe performance penalties in specific situations, we improved them by the follow-up protocol taDOM2+. By introducing 4 additional intention modes, we could avoid access and explicit locking of child nodes in case of specific conversions of the parent node. As a consequence, we obtained the more complex protocol taDOM2+ having 12 lock modes. The DOM3 standard introduced a richer set of operations which led to several new tailored lock modes for taDOM3 and—again to optimize specific conversions—we added even more intention modes (again indicated by the +-suffix) resulting in the truly complex protocol taDOM3+ specifying compatibilities and conversion rules for 20 lock modes (see [Haustein and Härder 2008] for details).

8. CONCLUSIONS

In recent years, we have accomplished substantial advances and performance enhancements during the optimization of our XDBMS *XTC* [Härder et al. 2010]. For large classes of XML documents, we have proven that DeweyIDs are not only *the key to fine-grained management* but also the prerequisite for flexible handling and performance optimization.

REFERENCES

Härder, T., Haustein, M. P., Mathis, C., and Wagner, M. Node Labeling Schemes for Dynamic XML Documents Reconsidered. *Data & Knowledge Engineering*(1):126–149, 2007.

Härder, T., Mathis, C., Bächle, S., Schmidt, K., and Weiner, A. M. Essential Performance Drivers in Native XML DBMSs (keynote paper). *Current Trends in Theory and Practice of Computer Science, LNCS 5901*:29–46, 2010.

Haustein, M. P. and Härder, T. Optimizing Lock Protocols for Native XML Processing. *Data & Knowledge Engineering*(1):147–173, 2008.

Haustein, M. P., Härder, T., Mathis, C., and Wagner, M. DeweyIDs—The Key to Fine-Grained Management of XML Documents. In *Proc. of SBBD Conf.* Brazil, pp. 85–99, 2005.

Li, C., Ling, T. W., and Hu, M. Efficient Updates in Dynamic XML Data: from Binary String to Quaternary String. *The VLDB Journal*(3):573–601, 2002.

Mathis, C. Storing, Indexing, and Querying XML Documents in Native Database Management Systems. *Ph.D. Thesis, University of Kaiserslautern, Verlag Dr. Hut, Munich*, 2009.

Mathis, C., Härder, T., and Schmidt, K. Storing and Indexing XML Documents Upside Down. *Computer Science – Research and Development*(1-2):51–68, 2009.

Miklau, G. XML Data Repository. In *http://www.cs.washington.edu/research/xmldatasets*, 2002.

O'Neil, P., O'Neil, E. J., Pal, S., Cseri, I., Schaller, G., and Westbury, N. OrdPaths: Insert-Friendly XML Node Labels. In *Proc. of Sigmod Conf.* USA, pp. 903–908, 2004.