Efficient Structural Bulk Updates on the Pre/Dist/Size XML Encoding

Motivation

XQuery is a functional programming language. It is developed with a strong focus on querying and manipulating XML data. XQuery 1.0 became a W3C recommendation in 2007. The XQuery Update Facility (XQUF) extension followed in 2011 and is widely adopted today.

BaseX is an open-source, native XML database and XPath/XQuery 3.1 processor. Its internal XML representation is the interval-based Pre/Dist/Size encoding schema. Very compact records of fixed length support efficient evaluation of all XPath axes.

In contrast, prefix-based encodings have a high updatability. Dynamic labels require little (no) relabeling after inserts or deletes. Yet, accessing nodes on disk requires a layer of indirection as records vary in length.

Each encoding has its advantages and disadvantages. But have we really reached the limits with Pre/Dist/Size with regards to updatability?

Pending Update List (PUL)

• ‘deep’ cache of XQUF atomic update primitives
• memory-intensive
• changes applied as last step of a transaction
• “update effects only accessible after the transaction”
• stages: 1. collect primitives 2. prepare application 3. apply updates

Pre/Dist/Size Updatability

While value updates are cheap, structural updates require re-labeling a potentially large amount of records. Implicitly stored Pre values of successor records are shifted efficiently. Only Size values of ancestor nodes are affected. However, a Dist value is updated if a node is inserted or deleted between an existing node and its parent. The costs depend on the update location and document structure.

Bulk updates are especially expensive. Each structural update primitive affects a set of Dist values. These sets are heavily intersected.

Possible Solution

• leverage Pending Update List to pre-compute final distance values
• adjust each Dist value only once

Efficient Structural Bulk Updates

The Atomic Update Cache is an efficient data structure that complements the PUL.

AUC

• collects all update information of a snapshot in document order
• points to affected table records and Dist values
• maps Pre values before/after updates
• is applied in reverse document order
• -> enables us to delay and pre-calculate Dist updates

Bulk Update Stages

1. location
2. subtree
3. target
4. rank

Filling the AUC is non-trivial. How to map XQUF primitives to atomic updates without violating the document order? Each update primitive operates on a target node and affects a specific table location. Two primitives can be ordered by a four-step comparison:

Results

2010 Apple iMac, Intel Core i3 3.2 GHz, 8 GB RAM, 1 TB hard-disk drive, BaseX 7.3, 7.7 (AUC), OS X 10.8.4, Oracle Java 7.12, -Xmx6G, XMark data

Q1 delete node //date

Q2 for $d in //date return insert node <date return> /date after $d

Observations

• AUC improves performance by magnitudes
• much faster distance adjustment
• still quadratic complexity for bigger documents
• insertion/deletion of records is the new bottleneck (structural)

Keeping records in document order is the new limiter

Conclusion

• Efficient Structural Bulk Updates exploit the XQUF conditions
• little overhead is added in the form of the AUC
• the tree structure is efficiently restored after updates
• Pre/Dist/Size reduces advantage of prefix labelling schemas (bulk updates)
• Document order is the new limiter