Deferred node-copying scheme for XQuery processors

Jan Kurš and Jan Vraný

Software Engineering Group, FIT ČVUT,
Kolejn 550/2, 160 00, Prague, Czech Republic
kurs.jan@post.cz, jan.vrany@fit.cvut.cz

Abstract. XQuery is generic, widely adopted language for querying and manipulating XML data. Many of currently available native XML databases are using XQuery as its primary query language. The XQuery specification requires each XML node to belong to exactly one XML tree. In case of the XML subtree is appended into a new XML structure, the whole subtree has to be copied and afterwards the copy is appended to the new structure. This may lead into excessive and unnecessary data copying and duplication. In this paper, we present a new XML node copying scheme that defers the node data copy operation unless necessary. We will show that this schemes significantly reduces the XML node copy operations required during the query processing.

Keywords: XML, XQuery, XQuery Processor, Smalltalk

1 Introduction

XQuery is an XML query language designed by the World Wide Web Consortium. Nowadays, almost every XML tool set or database vendor provides its own XQuery implementation. Although widely adopted, fast and efficient implementation is still lacking. Optimization techniques for XQuery are still a subject to an active research.

XQuery evolves into a full-featured language. XML data could be not only queried, but also created using feature called element constructors. One particular problem deals with this feature. XQuery 1.0 and XPath 2.0 Data Model specification [1] forbids sharing of data model among multiple XML node hierarchies. Section 2.1 says:

> ...
> *Every node belongs to exactly one tree, and every tree has exactly one root node.*
> ...

If a XML node is added into a new XML tree, the naive realization of this requirement would create a new node (by copying the original one) and the copy

```
1 let $authors = element authors { fn:doc("doc.xml")//authors }
2 let $titles = element titles { fn:doc("doc.xml")//titles }
3 return element result { $titles }
```

Fig. 1. Simple document-creating query

would be placed into the new XML tree. Consider the query at figure 1 is to be evaluated and its output is to be serialized to an output file.

In this case, a whole XML subtree that matches `fn:doc("doc.xml")//authors` is never used. This may lead into excessive node copying depending on the subtree size. Consequently, such unnecessary copy operations leads into higher memory (or swap storage) consumption and thus badly affects overall performance.

In this paper we will describe an efficient node-copying scheme that avoids unnecessary copying while preserving XQuery semantics. We will also discuss its correctness and benchmark results.

The paper is organized as follows: section 2 give an overall description of the node-copying scheme mentioned above. Section 3 discusses experimental results based on running XMark benchmarks. Section 4 provides a brief overview of related work. Section 5 concludes by summarizing presented work.

2 Deferred Node-copying

The basic idea is simple: share existing XML nodes between node hierarchies and defer node-copy operation unless absolutely inevitable. In our implementation the XML node can belong into multiple node hierarchies, but the XQuery specification requriement mentioned in section 1 is preserved.

The deferred node copying scheme has been developed to meet two main goals:

- separate query processing logic from underlying physical data model and
- reduce memory consumption by preventing unnecessary data copying

The first requirement has software engineering origin. XQuery is being considered as content integration language and thus XQuery processors should be able to operate over various data models, not necessarily XML-based. Moreover, good separation of query processor from physical data model provides possibility to use one XQuery implementation in multiple environments – as a standalone XQuery tools or within a database management machine.

The latter goals also came from practical needs. In case of large documents and complex queries, naive implementation of an XQuery may consume – in edge cases – twice more memory than actually needed.

2.1 XDM Adaptor

XDM specification defines a *sequence* to be an instance of data model. Each sequence consists of zero or mode *items*. An *item* is either a *node* or *atomic value*.

The specification also defines a bunch of node properties such as `dm:node-name` or `dm:parent`.

To meet our first goal we separates node from its physical data storage though an *XDM adaptor* which operates on so called *node ids*. Node id is an unique identifier of an XML node within particular physical storage. The structure of the node id is not defined – in fact node id could be anything depending on the storage: reference to a DOM node in memory, pointer to a database file or simple integer. For given XDM property and node id, the XDM adaptor returns value of that property. For properties containing other node (such as `dm:parent` or `dm:children`), the adaptor returns node's node id.

Usage of XDM adaptor give us relatively easy and straightforward way how to access different physical data models. The only requirement is to implement a new XDM adaptor object with 17 accessor methods corresponding with 17 properties defined by the XDM specification. The particular XDM adaptor astract any kind of data source and may use any kind of optimization (such as extensive caching) to access data effectively. However the physical data storage and access stategies are hidden to the rest of the XQuery processor.

2.2 Node States

During query processing there are two kinds of nodes[1]. First kind of nodes are such nodes, that comes from external data source (*e.g.,* XML file or database). We call this nodes *accessed nodes*. More specifically, usage of XQuery functions `fn:doc` and `fn:collection` will result in collection of accessed nodes.

Second kind of nodes are such nodes, that are created during query processing either by element constructors or as a copy of accessed node. We call this nodes *constructed nodes*.

In order to defer copy operation, a new node property called *node state* is introduced. This property attached to every node regardless whether the node is accessed node or constructed node. Each node is in exactly one state from following three states:

Accessed State. All accessed nodes are initially in *accessed state*.

Constructed State. All nodes which are created during the query processing are in a constructed state.

Hybrid State. Nodes which should be copied (according to XQuery specification), but the its copying is actually deferred are in a *hybrid state*. Nodes in hybrid state may have its `dm:parent` property temporarily overridden by another value. In fact, nodes in hybrid state may have two parents. We call nodes in hybrid state *hybrid nodes*.

[1] Please note that these "kinds" has no relation to node kinds as defined by XDM specification, section 6.

2.3 Transitions

During the query processing, the state of the node may change. The state diagram of the node is shown at figure 2. A state transition is triggered by an action issued on a particular node. There are three kinds of actions:

Copy Action. The copy action is performed whenever the processor creates a copy of given node, as required by the XQuery specification.

Change Action. The change action models any change in a data model such as setting a new parent.

"Child Read" Action. The "child read" action represents the situation when the XQuery processor accesses child nodes of given node. The most common case includes XPath expression processing.

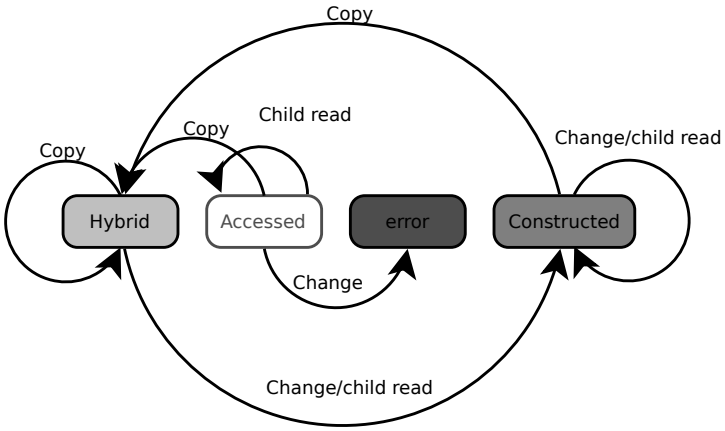

Fig. 2. Node State Transitions

```
1 <?xml version="1.0"?>
2 <root>
3   <elem>elem1</elem>
4   <elem>elem2</elem>
5 </root>
```

Fig. 3. `doc.xml` contents

Consider a document `doc.xml` (it's content is shown at figure 3) and query 1 (figure 4). During execution of the query, following actions are performed:

```
1 element myroot {
2   attribute attr { 'value' },
3   fn:doc("doc.xml")/elem[0]
4 }
```

Fig. 4. Example Query 1

1. The `myroot` element is created in the constructed state. Then three *change actions* are issued on that node: one for setting the node name to "myroot", second for adding attribute "attr" and finally third for adding text node with "value" as a child node of attribute node.
2. Afterwards, the `doc.xml` is read and two *child read actions* are performed in order to evaluate XPath expression – first on document node (which is initially in accessed state) and second on root element node (also initially in accessed state).
3. Finally, the first `elem` (accessed) node from `doc.xml` is to be added into the `myroot` (constructed) node. In that case, the `elem` node and all its descendants should be copied – the XQuery processor issue a *copy action* on `elem` node.
   Instead of doing physical (deep) copy of given node, we change its state to the hybrid state and set a "alternate" parent node to `myroot`. None of nodes descendants are ever touched. Effectively, hybrid nodes are temporarily shared among multiple node hierarchies, as depicted at figure 5.

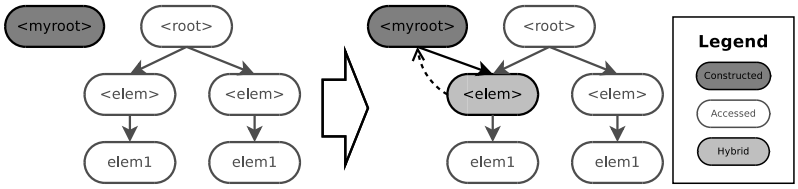

Fig. 5. Two XML trees sharing one hybrid node

In following sections we will give a more detailed discussion of state transitions.

Accessed Node Transitions. As we said before, accessed node are nodes that comes from external data sources. When a copy of an accessed node is required, the node state is changed from accessed to hybrid and no physical data copy is made. Changes to accessed nodes are not permitted – any change will immediately lead into an error. It's an XQuery processor responsibility not to try to change accessed nodes.

Constructed Node Transitions. Copy operations on constructed nodes behaves exactly as on accessed nodes. Changes to constructed nodes are permitted.

Hybrid Node Transitions. Transitions based on actions on hybrid nodes are bit more interesting:

Copy Action. Copy action on hybrid nodes is a no-op. As a result, the same node is returned with its state unchanged.

Change Action. Whenever any of node properties is to be changed (its value is changed, new child node is added or removed, etc.), the node state is changed to constructed and all node properties are copied.

The copy of a node is inevitable if data has changed. See the query at figure 6. When processing expression at line 5, two things happen (in that order):

1. The text node *"elem1"* (a result of `$doc/elem[0]/text()` expression) is added to the `myroot` element. States of nodes after this addition are depicted at left side of figure 7.
2. Afterwards, the *" is the first"* text is to be appended to the (now hybrid) text node *"elem1"*. Because XQuery specification forbids two or more adjacent text nodes, value of XDM property of the text node is changed from string "elem1" to "elem1 is the first". Obviously, the hybrid text node must be copied. The XML data accessible though `$doc` must remain unchanged.

Child Read Action. When a child read action is issued, the node state is changed to constructed, all node properties are copied and all its direct descendants are copied *i.e.,* a copy action is performed for node children.

The reason for such a behavior is the fact that the adaptor returns node ids for node-like properties and that the new (overridden) parent property value is not a part of underlying data model.

Let's have an hybrid element node `elem` that contains accessed text node *"elem1"*. Suppose that `./text()/../../..` expression is to be evaluated starting with hybrid node `elem` as the context node. Evaluation of such expression will result in repeated access to parent XDM property. Because the value parent property is obtained using XDM adaptor that operates only with node ids and because the overridden parent value for given node is maintained outside the primary data storage, there is no way how obtain correct (overridden) parent.

To overcome this issue, we convert hybrid node to a constructed one on child read action. Such a behavior illustrated at figure 8.

Data are physically copied only when hybrid node is either being changed or its children are being read.

Serialization of Result Set. Once the query is processed, serialization of result set may not lead into XML node copying. Because query is already processed, no node kind transions must be performed during serialization and thus no node copies must be created. Obviously, if the application wants work with the result set as with nodes in memory and wants to perform some modification on it, the result set must be copied.

```
let $doc:= doc("doc.xml")
return
  element myroot {
    element myelem {
      { $doc/elem[0]/text() } is the first
    }
  }
```

Fig. 6. Example Query 2

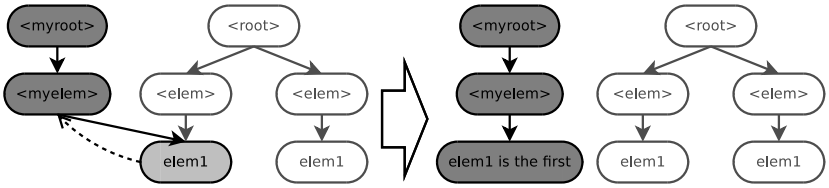

Fig. 7. Change of hybrid node into the constructed node

3 Discussion

3.1 Specification Conformance

Although deferred node-copying scheme does not require the XML nodes to belong to exactly one node hierarchy it preserves original XQuery semantics. Our claim is based on the results from the XQuery Test Suite [3].

The *axes tests* and *element constructors tests* from *Minimal Conformance - Expressions* section of XQTS Catalogue cover the node identity semantics and were used to test the correctness of deferred node-copying scheme. Our proof-of-concept implementation successfully passes all the mentioned test cases.

3.2 Benchmarks

Presented deferred node-copying scheme has been developed in order to increase XQuery processor performance by reducing number of copy operations. A natural question is whether this scheme has substantial effect in real-world applications. The table 3.2 shows number of copy operations for selected XMark [2] queries[2] on a file created with the XMark data generator.

Number of saved copies is dependent on a query characteristics. There are no new nodes created in a Q1 command and that is why there is no difference in results.

[2] Plus one nonstandard query marked INC. Its code is `element a {doc("file:///auctions.xml") }`. We include it as an illustration of extreme case.

Q. #	**DNC**		**IC**		Q. #	**DNC**		**IC**	
	N_h	N_c	N_h	N_c		N_h	N_c	N_h	N_c
Q1	0	0	0	0	Q2	106	0	0	106
Q3	0	44	0	44	Q4	0	0	0	0
Q5	0	0	0	0	Q6	0	0	0	0
Q7	0	0	0	0	Q8	25	25	0	50
Q9	12	25	0	39	Q10	402	1	0	1244
Q11	12	25	0	39	Q12	3	3	0	6
Q13	22	22	0	560	Q14	0	0	0	0
Q15	7	0	0	7	Q16	0	6	0	6
Q17	0	138	0	138	Q18	0	0	0	0
Q19	217	217	0	434	Q20	8	0	0	12
INC	2074	114	0	5857					

Legend:

N_h – number of hybrid nodes created
N_c – number of physically copied nodes
DNC – evaluated using deferred node-copying scheme
IC – evaluated using immediate copy as specified by the XQuery specification

Table 1. Benchmark results

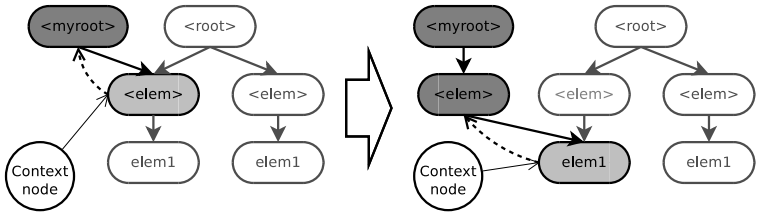

Fig. 8. Change of hybrid node while exploring the children

On the other hand, there are text nodes appended to elements in a Q2 command. The number shows that all the text nodes does not need to be copied, only transformed to the hybrid state.

Interesting results are provided by a Q13 command. There is a subtree appended to each result item during the query execution. Without the optimization, each element of a tree has to be copied. But if the optimization is turned on, only a few of nodes nodes has to be converted to the hybrid state and unnecessary copy of its sub-nodes is prevented.

4 Related Work

eXist XQuery Processor. eXist[3] is an open-source XML-native database with XQuery as its primary query language. As far as we know, eXist XQuery implementation unconditionally copies nodes whenever the node is to be added into a different node hierarchy. Our approach is different since we avoid unnecessary copy operations.

Saxon XQuery Processor. Saxon[4] is well-known, widely adopted XML tool set including XSLT 2.0, XPath 2.0 and XQuery 1.0 processor. Saxon's XQuery processor introduces concept of *virtual nodes* – a light-weigh node shallow copies that shares as many properties as possible with their origin.

Similarly to our approach, for a given virtual node some of standard XDM properties may be overridden – namely the parent property. When the Saxon XQuery processor iterates over virtual node's children, those are converted to virtual nodes.

However, presented deferred node copying scheme differs from virtual nodes approach in several aspects:

1. Creating virtual copies requires a new object to be allocated in the memory. Deferred node copying scheme shares the same object.

[3] http://exist.sourceforge.net/
[4] http://saxon.sourceforge.net/

2. Creation of virtual copies is a part of XQuery processing logic and must be explicitly expressed, whereas our approach separates copying logic of an XDM model from the query evaluation logic.

5 Conclusion and Future Work

This paper presents a deferred XML node-copying scheme for XQuery processors that significantly reduces number of source nodes copy operations required during query processing. This scheme defers the copy operation unless absolutely inevitable. Whether the node is actually copied depends on a node state, a new property which is maintained for each node in addition to standard XDM properties. Correctness of this approach has been successfully tested by XQuery Test Suite.

The main benefits of deferred node-copying scheme are: (i) efficiency, (ii) easy to implement, (iii) independent on physical data model and (iv) independent on XQuery processing logic.

As a future plan, we plan to extend this scheme for use with various XML indexing approaches, Ctree [4] and [5] in particular.

References

1. M. N. Mary Fernández, Ashok Malhotra. Jonathan Marsh and N. Walsh. *XQuery 1.0 and XPath 2.0 Data Model (XDM).* W3C, 1st edition, 2006. `http://www.w3.org/TR/xpath-datamodel`.
2. A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse. Xmark: A benchmark for xml data management. In *In VLDB*, pages 974–985, 2002.
3. W3C XML Query Working Group. *XML Query Test Suite.* W3C, 1st edition, 2006. `http://www.w3.org/XML/Query/test-suite/`.
4. Q. Zou, S. Liu, and W. W. Chu. Ctree: a compact tree for indexing xml data. In *WIDM '04: Proceedings of the 6th annual ACM international workshop on Web information and data management*, pages 39–46, New York, NY, USA, 2004. ACM.
5. Q. Zou, S. Liu, and W. W. Chu. Using a compact tree to index and query xml data. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 234–235, New York, NY, USA, 2004. ACM.