Efficient XML Storage based on DTM for Read-oriented Workloads

Graduate School of Information Science, Nara Institute of Science and Technology

Makoto Yui
Jun Miyazaki, Shunsuke Uemura, Hirokazu Kato

International Workshop on Advanced Storage Systems 2007 (ADSS 2007)
Outline

- Motivation
- Related work
- Document Table Model (DTM)
  - XML storage based on DTM
    - System Overview
    - Physical Layout
    - Buffer Management
- Experimental Evaluation
- Conclusions
Motivation - Backgrounds

- Past research topics in XML data management
  - Labeling and indexing XML trees
    - Dewey ordering
    - Ordpaths
    - XR-Trees
  - Join processing
    - Structural-joins
    - Twig-joins
  - Indexing on paths
    - XRel
    - Index Fabric

Well studied topics
Towards efficient XML data processing
Less studied topics

Internal data model
- Relational
- Hybrid (SystemRX)
- Tree (e.g., DOM)

Buffer management
- Natix

Our focus
Motivation – Design goals

- Design an XML storage scheme optimized for read-oriented workload
  - Node-level update is not always required
  - Updating capabilities have more or less drawbacks

The design of read-oriented database has been suggested for relational databases but not studied for XML databases yet.

- Focus on iterative XQuery processing in which an operator tree consists of iterators
  - Ideal XML storage scheme depends on the processing model (e.g., tuple-at-a-time or set-at-a-time)

The reason we selected this model is set-at-a-time processing of XQuery involves lots of joins, and it causes performance deterioration.

How the data access is achieved for each of the two processing model?

Tree traversal of set-at-a-time processing

Context node

- auction
- site
- regions
- regions
- regions

- sites under actions are selected
- regions under sites are selected

Each edges are connected by join on nodes’ identifiers

Traversed in a breadth-first search manner
Tree traversal of tuple-at-a-time processing

Context node

- Each two nodes are connected by links
- Path processing is achieved by navigating link edges

Traversed in a depth-first search manner

It is as same manner as a document ordering
Motivation

- Design an **XML storage scheme** optimized for read-oriented workload

- Focus on **iterative XQuery processing** in which an operator tree consists of iterators

---

We examined **actual data access patterns** when evaluating XQuery queries **in order to design the suitable data layout**
Outline

- Motivation
- Related Work
- Document Table Model (DTM)
- XML Storage based on DTM (pDTM)
  - System Overview
  - Physical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions
Related work (1) storing scheme based on subtrees

Natix (University of Mannheim, Germany)

Allocates a page based on subtrees

- **Pros**  Effective for breadth-first traversals
- **Cons**  Not effective for depth-first traversals

![Diagram showing related work storing scheme based on subtrees]

- **Proxy Node**
- **Helper aggregate nodes:**
Related work (2) storing scheme based on a schema

OrientStore (Renmin University, China)

- **Pros**
  - Effective for path processing
- **Cons**
  - Schema information is required
  - Not effective for serialization and
dictionary

Clusters records according to a semantic segmentation

![Diagram](image_url)
Outline

- Motivation
- Related work
- Document Table Model (DTM)
  - System Overview
  - Physical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions
Document Table Model (DTM)

- Originally used by Apache Xalan XSLT processor
- Expresses an XML document as a table form

DOM has object footprints (e.g., object instantiation and memory consumptions)

DTM can avoid such object footprints

DTM table consists of primitive data types

An XML tree structure can be represented as a table by using link values
## System Overview

### Abstract Model

<table>
<thead>
<tr>
<th>TYPE</th>
<th>E</th>
<th>E</th>
<th>E</th>
<th>T</th>
<th>E</th>
<th>E</th>
<th>E</th>
<th>E</th>
<th>E</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARENT</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>NEXTSIB</td>
<td>-</td>
<td>6</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>10</td>
<td>9</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CID</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- **PageBuffer**
- **QNameTable**
- **StringChunk**
- **Physical Model**
- **Page in**
- **Cache miss**

---

#1 site
#2 region
#3 ..
Analyzing data access patterns

- Before designing physical layout of XML documents, we analyzed actual data access patterns of XQuery queries.

In general,

- Pages are required in the document-order
- Sequential accesses are frequently appeared
Access pattern analysis of XMark queries (1)

This plot explains that page #40,000 is assigned to nodes nearby end of the document.

Page #40,000 is assigned to nodes nearby end of the document.

Document order is reflected to the Y-axis.

This plot explains that pages are required in a document order from the lower left to the upper right.

Page #0 is assigned to nodes nearby the root.

Required Page

Access count
Recall that we claimed that tuple-at-a-time processing of XPath queries, in general, traverses XML-tree according to the document-ordering.

XQuery FLWR queries also shows the similar access patterns with few exceptions where queries contains lots of backward axes.

Note that the overall tendency is not restricted to XMark queries but also other benchmark queries.
Physical layout

- Pages are required in a document-order
- Sequential accesses are frequently appeared

- Document-ordered block allocation is suitable
- Prefetching is effective

The prefetching entries can compete for hot cache entries

We conducted informed prefetching with scan-resistant buffer management

Scan-resistant buffer management

Is also effective to sequential scans in XML query processing
Outline

- Motivation
- Related work
- Document Table Model (DTM)
  - System Overview
  - Physical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions
Experimental evaluation

- Compared to Natix version 2.1.1 where XMark SF = 5 and SF = 10
- Experimental settings

Today's normal PC setting

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel Pentium D 2.8GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>SuSE Linux 10.2 (Kernel 2.6.18)</td>
</tr>
<tr>
<td>RAM</td>
<td>2GB</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>SATA 7200rpm</td>
</tr>
<tr>
<td>Java</td>
<td>Sun JDK 1.6</td>
</tr>
<tr>
<td>JVM option</td>
<td>-server -Xms1400m -Xmx1400m</td>
</tr>
</tbody>
</table>

Elapsed time in a log scale

Query number

SF = 5

SF = 10
Our method is effective for IO-intensive workloads.

Has following-sibling

Contains "//"

Natix showed better performance for breadth-first traversals (e.g., following-sibling)

Queries whose outputs are large
Conclusions

Summary

- Proposed an efficient XML storage scheme base on DTM for iterative XQuery processing

- Our approach is effective for IO-intensive workloads such as queries including ‘//’.
  - Document-ordered block allocation
  - Informed prefetching and scan-resistant caching

Future work

- Automatic database tuning based on online analysis of data access patterns (e.g., buffer replacement policy and prefetching)
Thank you for your attention!

Questions?
Problem in XML-Relational mapping

XML

```
<student>
  <id>0551133</id>
  <name>ichiro</name>
</student>
```

XML Tree

Mapping

<table>
<thead>
<tr>
<th>id</th>
<th>parent</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>0551133</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>ichiro</td>
</tr>
</tbody>
</table>
Updating facilities and versioning

<table>
<thead>
<tr>
<th>TYPE</th>
<th>R E E E T E T E E E</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARENT</td>
<td>– 0 1 2 3 2 1 1 7 7 1 2</td>
</tr>
<tr>
<td>NEXTSIB</td>
<td>– – 6 4 – – 7 – 9 – – 4</td>
</tr>
<tr>
<td>CID</td>
<td>– 1 2 3 0</td>
</tr>
</tbody>
</table>

When accessing to a record,

 Logical address  ➔  Physical address

For updating facilities, we change this method as follows:

 Logical address  ➔  Address conversion table  ➔  Physical address
Buffer management (XMark Q10 as an example)

20% of original document is returned as the result

<table>
<thead>
<tr>
<th></th>
<th>Elapsed time (msec)</th>
<th>total read blocks</th>
<th>buffer replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>211.83</td>
<td>1,919,586</td>
<td>567,702</td>
</tr>
<tr>
<td>2Q</td>
<td>185.56</td>
<td>80,673</td>
<td>0</td>
</tr>
</tbody>
</table>

14.2% speedups
let $auction := fn:doc("auction.xml")
return
  for $i in distinct-values($auction/site/people/person/profile/interest/@category)
  let $p := for $t in $auction/site/people/person
    where $t/profile/interest/@category = $i
  return
    <personne>
      <statistiques>
        <sexe>{$t/profile/gender/text()}</sexe>
        <age>{$t/profile/age/text()}</age>
        <education>{$t/profile/education/text()}</education>
        <revenu>{$fn: data($t/profile/@income)}</revenu>
      </statistiques>
      <coordonnees>
        <nom>{$t/name/text()}</nom>
        <rue>{$t/address/street/text()}</rue>
        <ville>{$t/address/city/text()}</ville>
        <pays>{$t/address/country/text()}</pays>
        <reseau>
          <courrier>{$t/emailaddress/text()}</courrier>
          <pagePerso>{$t/homepage/text()}</pagePerso>
        </reseau>
      </coordonnees>
      <cartePaiement>{$t/creditcard/text()}</cartePaiement>
    </personne>
return <categorie>{<id>{$i}</id>, $p}</categorie>
Physical layout

Row Skeleton

Rows

Pages

Read in Pages

Blocks

accessed row

null

dynamically configured
Access pattern analysis of XBench queries

DC/SD Normal

TC/SD Normal
We present a memory mapped scheme extending the DTM model, it has boost the performance significantly.
Efficient XML Storage based on DTM for Read-oriented Workloads

Graduate School of Information Science, Nara Institute of Science and Technology

Makoto Yui
Jun Miyazaki, Shunsuke Uemura, Hirokazu Kato

International Workshop on Advanced Storage Systems 2007 (ADSS 2007)
Outline

- Motivation
- Related work
- Document Table Model (DTM)
  - XML storage based on DTM
    - System Overview
    - Physical Layout
    - Buffer Management
- Experimental Evaluation
- Conclusions
Motivation - Backgrounds

Past research topics in XML data management

- Labeling and indexing XML trees
  - Dewey ordering
  - Ordpaths
  - XR-Trees

- Join processing
  - Structural-joins
  - Twig-joins

- Indexing on paths
  - XRel
  - Index Fabric

Towards efficient XML data processing

Internal data model
- Relational
- Hybrid (SystemRX)
- Tree (e.g., DOM)

Buffer management
- Natix

Physical data layout

Well studied topics

Less studied topics

Our focus
Motivation – Design goals

- Design an **XML storage scheme** optimized for read-oriented workload
  - Node-level update is not always required
  - Updating capabilities have more or less drawbacks

The design of read-oriented database has been suggested for relational databases but not studied for XML databases yet.

- Focus on **iterative XQuery processing** in which an operator tree consists of iterators
  - Ideal XML storage scheme depends on the processing model (e.g., tuple-at-a-time or set-at-a-time)

**How the data access is achieved for each of the two processing model?**

Tree traversal of set-at-a-time processing

Context node

- auction

- site
  - regions
  - regions

- site
  - regions

sites under actions are selected
regions under sites are selected

Each edges are connected by join on nodes’ identifiers

Traversed in a breadth-first search manner
Tree traversal of tuple-at-a-time processing

- Each two nodes are connected by links
- Path processing is achieved by navigating link edges

Traversed in a depth-first search manner

It is as same manner as a document ordering
Motivation

- Design an **XML storage scheme** optimized for read-oriented workload

- Focus on **iterative XQuery processing** in which an operator tree consists of iterators

We examined **actual data access patterns** when evaluating XQuery queries **in order to design the suitable data layout**
Outline

- Motivation
- Related Work
- Document Table Model (DTM)
  - XML Storage based on DTM (pDTM)
    - System Overview
    - Physical Layout
    - Buffer Management
- Experimental Evaluation
- Conclusions
Related work (1) storing scheme based on subtrees

Natix (University of Mannheim, Germany)

Allocates a page based on subtrees

- **Pros**: Effective for breadth-first traversals
- **Cons**: Not effective for depth-first traversals

![Diagram of related work storing scheme based on subtrees]

- **Proxy Node**
- **Helper aggregate nodes**
Related work (2) storing scheme based on a schema

OrientStore (Renmin University, China)

- **Pros**
  - Effective for path processing

- **Cons**
  - Schema information is required
  - Not effective for serialization and

Clusters records according to a semantic segmentation
Motivation
Related work
Document Table Model (DTM)
XML Storage based on DTM (pDTM)
  - System Overview
  - Physical Layout
  - Buffer Management
Experimental Evaluation
Conclusions
Document Table Model (DTM)

- Originally used by Apache Xalan XSLT processor
- Expresses an XML document as a table form

- DOM has object footprints (e.g., object instantiation and memory consumptions)

- DTM can avoid such object footprints
  DTM table consists of primitive data types

An XML tree structure can be represented as a table by using link values
System Overview

Abstract Model

<table>
<thead>
<tr>
<th>TYPE</th>
<th>E</th>
<th>E</th>
<th>E</th>
<th>T</th>
<th>E</th>
<th>E</th>
<th>E</th>
<th>T</th>
<th>E</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARENT</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>NEXTSIB</td>
<td>-</td>
<td>6</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>10</td>
<td>9</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CID</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

PageBuffer

QNameTable

Cache miss

Page in

Pyshical Model

StringChunk

Asia

NorthAmerica

Africa...
Analyzing data access patterns

- Before designing physical layout of XML documents, we analyzed actual data access patterns of XQuery queries.

In general,

- Pages are required in the document-order
- Sequential accesses are frequently appeared
Access pattern analysis of XMark queries (1)

This plot explains that 28,000 data access pages are required in a document order from the lower left to the upper right.

Page #0 is assigned to nodes nearby the root.

Page #40,000 is assigned to nodes nearby the end of the document.

Document order is reflected to the Y-axis.

Access count
Access pattern analysis of XMark queries (2)

Recall that we claimed that tuple-at-a-time processing of XPath queries, in general, traverses XML-tree according to the document-ordering.

XQuery FLWR queries also shows the similar access patterns with few exceptions where queries contains lots of backward axes.

Note that the overall tendency is not restricted to XMark queries but also other benchmark queries.
Physical layout

- Pages are required in a document-order
- Sequential accesses are frequently appeared

- Document-ordered block allocation is suitable

- Prefetching is effective
  - The prefetching entries can compete for hot cache entries
  - We conducted informed prefetching with scan-resistant buffer management

- Scan-resistant buffer management
  - Is also effective to sequential scans in XML query processing
Outline

- Motivation
- Related work
- Document Table Model (DTM)
- XML Storage based on DTM (pDTM)
  - System Overview
  - Physical Layout
  - Buffer Management
- Experimental Evaluation
- Conclusions
Experimental evaluation

- Compared to Natix version 2.1.1 where XMark SF = 5 and SF = 10
- Experimental settings

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel Pentium D 2.8GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>SuSE Linux 10.2 (Kernel 2.6.18)</td>
</tr>
<tr>
<td>RAM</td>
<td>2GB</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>SATA 7200rpm</td>
</tr>
<tr>
<td>Java</td>
<td>Sun JDK 1.6</td>
</tr>
<tr>
<td>JVM option</td>
<td>-server -Xms1400m -Xmx1400m</td>
</tr>
</tbody>
</table>

Elapsed time in a log scale

Today's normal PC setting

Query number

SF = 5

SF = 10
**XMark SF=10**

Our method is effective for IO-intensive workloads.

Has following-sibling

Contains “//”

Natix showed better performance for breadth-first traversals (e.g., following-sibling)

Queries whose outputs are large
Conclusions

Summary

- Proposed an efficient XML storage scheme base on DTM for iterative XQuery processing

- Our approach is effective for IO-intensive workloads such as queries including ‘//’.
  - Document-ordered block allocation
  - Informed prefetching and scan-resistant caching

Future work

- Automatic database tuning based on online analysis of data access patterns (e.g., buffer replacement policy and prefetching)
Thank you for your attention!

Questions?
Problem in XML-Relational mapping

XML

```xml
@student>
  <id>0551133</id>
  <name>ichiro</name>
</student>
```

XML Tree

```
Student
  B
    id
    name
  C
    id
    name
```

Hard disk

Mapping

```
<table>
<thead>
<tr>
<th>id</th>
<th>parent</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>0551133</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>ichiro</td>
</tr>
</tbody>
</table>
```

Relational Table
Updating facilities and versioning

When accessing to a record,

Logical address  $\rightarrow$  Physical address

For updating facilities, we change this method as follows:

Logical address  $\rightarrow$  Address conversion table  $\rightarrow$  Physical address
Buffer management (XMark Q10 as an example)

<table>
<thead>
<tr>
<th>Buffer replacement</th>
<th>Elapsed time (msec)</th>
<th>total read blocks</th>
<th>14.2% speedups</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>211.83</td>
<td>1,919,586</td>
<td>567,702</td>
</tr>
<tr>
<td>2Q</td>
<td>185.56</td>
<td>80,673</td>
<td>0</td>
</tr>
</tbody>
</table>

20% of original document is returned as the result
let $auction := fn:doc("auction.xml")
return
for $i in distinct-values($auction/site/people/person/profile/interest/@category)
let $p :=
    for $t in $auction/site/people/person
        where $t/profile/interest/@category = $i
        return
    <personne>
    <statistiques>
        <sexe>{ $t/profile/gender/text() }</sexe>
        <age>{ $t/profile/age/text() }</age>
        <education>{ $t/profile/education/text() }</education>
        <revenu>{ fn:doc($t/profile/@income) }</revenu>
    </statistiques>
    <coordonnees>
        <nom>{ $t/name/text() }</nom>
        <rue>{ $t/address/street/text() }</rue>
        <ville>{ $t/address/city/text() }</ville>
        <pays>{ $t/address/country/text() }</pays>
    </coordonnees>
    <reseau>
        <courrier>{ $t/emailaddress/text() }</courrier>
    </reseau>
    </personne>
return <categorie>{<id>{ $i }}</id>, $p }</categorie>
Pyshical layout

accessed row

Row Skeleton

Rows

Pages

Read in Pages

Blocks

dynamically configured
Access pattern analysis of XBench queries

DC/SD Normal

TC/SD Normal
Memory Mapped DTM

```c
#include <sys/mman.h>

void *mmap(void *start, size_t length, int prot, int flags,
           int fd, off_t offset);

int munmap(void *start, size_t length);
```

We present a memory mapped scheme extending the DTM model, it has boost the performance significantly.