On the Use of Query-driven XML Auto-Indexing

Karsten Schmidt and Theo Härder
Motivation – Self-Tuning '10

- The last 10+ years
  - Index tuning
    - What-if
    - Wizards, Guides, Druids
  - Monitoring
  - Workload analysis
  - Static resources
  - Single-user optimization
  - ...

- Today: dynamic environments (queries, data, resources)
  - Self-tuning has to be done Permanently and Online
Motivation – Index Tuning

- Which column(s) to index?

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>DepID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Norman</td>
<td>28</td>
<td>12A</td>
<td>North</td>
</tr>
<tr>
<td>2</td>
<td>Charles</td>
<td>58</td>
<td>98</td>
<td>West</td>
</tr>
</tbody>
</table>
Motivation – Index Tuning

• Which column(s) to index?

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>DepID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Norman</td>
<td>28</td>
<td>12A</td>
<td>North</td>
</tr>
<tr>
<td>2</td>
<td>Charles</td>
<td>58</td>
<td>98</td>
<td>West</td>
</tr>
</tbody>
</table>
Motivation – Index Tuning

- Which column(s) to index?

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>DepID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Norman</td>
<td>28</td>
<td>12A</td>
<td>North</td>
</tr>
<tr>
<td>2</td>
<td>Charles</td>
<td>58</td>
<td>98</td>
<td>West</td>
</tr>
</tbody>
</table>
Motivation – Index Tuning

- **Which column(s) to index?**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>DepID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Norman</td>
<td>28</td>
<td>12A</td>
<td>North</td>
</tr>
<tr>
<td>2</td>
<td>Charles</td>
<td>58</td>
<td>98</td>
<td>West</td>
</tr>
</tbody>
</table>

Multiple candidates?
Motivation – Index Tuning

- Which column(s) to index?

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>DepID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Norman</td>
<td>28</td>
<td>12A</td>
<td>North</td>
</tr>
<tr>
<td>2</td>
<td>Charles</td>
<td>58</td>
<td>98</td>
<td>West</td>
</tr>
</tbody>
</table>

multi-column index or multiple indexes?
Motivation – Index Tuning

- Which column(s) to index?

ISP: 10 tables each having 8 columns and max 3 columns per index → ~4,000 indexes

redundancy and maintenance!
Motivation – Index Tuning

• **Which column(s) to index?**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
<th>DepID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Norman</td>
<td>28</td>
<td>12A</td>
<td>North</td>
</tr>
<tr>
<td>2</td>
<td>Charles</td>
<td>58</td>
<td>98</td>
<td>West</td>
</tr>
</tbody>
</table>

ISP: 10 tables each having 8 columns and max 3 columns per index → ~4,000 indexes

• **XML specifics on top of the ISP**

  – Semi-structured and schema changes
  – Indexing may focus on structure and/or content
  – Flexible path expressions (axes, wildcard, name)
    → containment problem and generalized indexes
    → descendant axis is evil, but wildcards too
  – Node types (elements, attributes, text, …)
Outline

- XML Index Types
- Statistics for Cost-based Index Selection
- Index Configuration Management
- Evaluation
- Summary & Outlook
Outline

- XML Index Types
- Statistics for Cost-based Index Selection
- Index Configuration Management
- Evaluation
- Summary & Outlook
XML Index Types - Storage

- **XML Mapping**
  - Native
  - No *Shredding* or *Blob*

- **Fundamentals**
  - Node labeling
  - B-tree
XML Index Types - Storage

- **XML Mapping** *Elementless*

```
<table>
<thead>
<tr>
<th>bib</th>
<th>publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>book</td>
<td>book</td>
</tr>
<tr>
<td></td>
<td>book</td>
</tr>
</tbody>
</table>

- Structure
- Content

```

xml

<table>
<thead>
<tr>
<th>year</th>
<th>id</th>
<th>title</th>
<th>author</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>1</td>
<td>TCP/IP</td>
<td>Stevens</td>
<td>65.95</td>
</tr>
</tbody>
</table>

Stevens W.
XML Index Types - Storage

- **XML Mapping** *Elementless*

![XML Structure Diagram]

- Elements:
  - 1 bib
  - 1.3 publication
  - 1.3.3 book
  - 1.3.5 book
  - 1.3.7 book

- Attributes:
  - 1.3.3.1 year: 1994
  - 1.3.3.2.1 id: 1
  - 1.3.3.3 title: TCP/IP
  - 1.3.3.5 author:
    - last: Stevens
    - first: W.
  - 1.3.3.7 price: 65.95

- Content:
  - 1.3.3.1.3
  - 1.3.3.2.1.3
  - 1.3.3.3.3
  - 1.3.3.5.3.3
  - 1.3.3.5.5.3
  - 1.3.3.7.3
XML Index Types - Storage

- XML Mapping *Elementless*

```
1.3.3.1.3 year
1994
1.3.3.1.3

1.3.3.2.1.3 id
1
1.3.3.2.1.3

1.3.3.3 title
TCP/IP
1.3.3.3.3

1.3.3.5 author
1.3.3.5.3 last
Stevens
1.3.3.5.3.3

1.3.3.5.5 first
W.
1.3.3.5.5.3

1.3.3.7 price
65.95
1.3.3.7.3

1 bib
1.3 publication

1.3.5 book

1.3.7 book
... Structure

content```
XML Index Types - Storage

• XML Mapping *Elementless*

Path Synopsis
XML Index Types - Storage

• XML Mapping \textit{Elementless}

Path Synopsis + B-tree layout

Structure

content

1.3.3.1.3
1.3.3.2.1.3
1.3.3.3.3
1.3.3.7.3
1.3.3.5.3.3
1.3.3.5.5.3

1 bib
2 publication
3 book
4 year
5 id
6 type
7 title
8 author
9 price
10 first
11 last
12 journal
13 publisher
14 name
15 loc
16 paper
17 name
18 title

1.3.3.1.3 1.3.3.2.1.3 1.3.3.3.3 1.3.3.5.3.3 1.3.3.5.5.3
1994 1 1.3.9 1.3.11
65.95

DeweyID content PCR
XML Index Types - Storage

- XML Mapping *Elementless*

Primary target: space efficiency
Secondary: more indexing options for free!
XML Index Types

- **Content**
  - John
    - XTC
      - 1.13.7
      - 1.13.11
      - 1.79.5.3
    - Pure
      - ...
    - DBMS
      - 1.3.5.7
      - 1.3.15.3
      - 1.217.5.3
      - ...
    - Chris

- **Element**
  - last
    - B-Tree
      - author
        - B*-Tree
          - 1.3.5
    - title
      - B*-Tree
        - 1.3.5.5
      - B*-Tree
        - 1.3.3
XML Index Types

- **Content**

  - XTC
    - Pure
  - DBMS
    - Chris
  - John

  1.13.7
  1.13.11
  1.79.5.3

  1.3.5.7
  1.3.15.3
  1.217.5.3

- **Path**

  Path: `/bib/title` → PCRs: **5,11,33**

- **Element**

  - B*-Tree
    - 1.3.5
  - B*-Tree
    - 1.3.5.5
  - B*-Tree
    - 1.3.3

- **Content And Structure**

  Path: `/bib/title` → PCRs: **5,11,33**
XML Index Types

- **Content**
  - Easy to define
  - Generic
  - Large
  - Maintenance costs

- **Element**
  - Easy to define
  - Generic
  - False positive filtering
  - Large
  - Maintenance costs

- **Path**
  - Specific
  - Clustering
  - Small
  - Hard to define

- **Content And Structure**
  - Specific
  - Clustering
  - Medium size
  - No document order
  - Type support
  - Hard to define
XML Index Types – Usage Sample

- Simplified Query Graph Model for XMark 01

Query:
let $auction := doc("auction.xml")
return
  for $b in $auction/site/people/person[@id = "person0"]
  return $b/name/text()
XML Index Types – Usage Sample

- **Simplified Query Graph Model for XMark 01**

Query:
let $auction := doc("auction.xml")
return
  for $b in $auction/site/people/person[@id = "person0"]
  return $b/name/text()
XML Index Types – Usage Sample

- Simplified Query Graph Model for XMark 01

Query:
```
let $auction := doc("auction.xml")
return
  for $b in $auction/site/people/person[@id = "person0"]
    return $b/name/text()
```

Diagram:
- Project
  - Expressions
    - fn:text(in)
- Structural Join
  - Axis: child
- Structural Join
  - Axis: attribute
- Structural Join
  - Axis: child
- Twig Join
- Document Root 'site'
- Document Scan
  - NodeTest: 'name'
- Content Idx Scan
  - NodeTest: 'id'
  - Key: person0
- Document Scan
  - NodeTest: 'person'
- Document Scan
  - NodeTest: 'people'
XML Index Types – Usage Sample

● Simplified Query Graph Model for XMark 01

Query:
let $auction := doc("auction.xml")
return
for $b in $auction/site/people/person[@id = "person0"]
return $b/name/text()
Query:
let $auction := doc("auction.xml")
return
  for $b in $auction/site/people/person[@id = "person0"]
  return $b/name/text()
XML Index Types – Usage Sample

- Simplified Query Graph Model for XMark 01

Query:
let $auction := doc("auction.xml")
return
for $b in $auction/site/people/person[@id = "person0"]
return $b/name/text()
XML Index Types – Usage Sample

- **Simplified Query Graph Model for XMark 01**

Query:
```xml
let $auction := doc("auction.xml")
return for $b in $auction/site/people/person[@id = "person0"]
return $b/name/text()
```

**Besides element and content index**

**More than 280** path or CAS indexes possible by varying depth, axes ("/", "//"), clustering, and wildcards
Outline

- XML Index Types
- Statistics for Cost-based Index Selection
- Index Configuration Management
- Evaluation
- Summary & Outlook
Statistics

- **Extended Path Synopsis**
  - Each node is extended:
    - Instance counter | Ø content length | IUD counter

- **Index Statistics**
  - Gathered during storage:
    - B-tree height
    - # leave pages
    - Cardinality
    - Index size

Overhead:
- Extended PS typically <0.1% of XML document
- No optimization yet (e.g., recursion)
- Processing overhead less than 5-6%
Statistics

- **Cost Estimation for Index Candidates**
  1. Evaluate Index Expression on Path Synopsis → **PCR set**
  2. For each PCR **add** node's **statistics** (cardinality, width, IUD counter)
  3. Index **type-dependent estimation** of height, no. of leaves, and size

- **Estimation Accuracy**

![Graph showing estimation error in %]
Outline

• XML Index Types
• Statistics for Cost-based Index Selection
• Index Configuration Management
• Evaluation
• Summary & Outlook
Index Configuration Management

- Integrated into XTC backend
- Asynchronous jobs for
  - Candidate search
  - Index building
- Flexible configuration
  - Type support
  - Pruning thresholds
  - Size limits
  - Job schedule
Index Configuration Management

1. **Record** query processing **costs**
2. *If* **cost > threshold** *do* **Auto Indexing**
3. **Feedback** **Auto Indexing costs**

**Auto Indexing**

1. **Traverse** query plan bottom-up (access operators)
2. Generate index **candidates**
3. **Rerun** optimization including candidates
4. **Analyze** plan(s) for selected candidates
5. **Update** candidate set

Cost/benefit **calculation** → schedule index materialization **jobs**
Candidate Generation

Auto Indexing – Traverse

Query:
let $doc:=doc("sample.xml")
return for $book in
  $doc/book[@xtc:id="book1"]
return $book/title

Candidates:

QGM representation
Candidate Generation

Auto Indexing – Traverse

Query:
let $doc:=doc("sample.xml")
return for $book in
$doc//book[@xtc:id="book1"]
return $book/title

Candidates:
1. //
Candidate Generation

Auto Indexing – Traverse

Query:
let $doc:=doc("sample.xml")
return $book/title

Candidates:
1. //
2. book
Auto Indexing – Traverse

Query:
let $doc:=doc("sample.xml")
return for $book in 
$doc//book[@xtc:id="book1"]
return $book/title

Candidates:
1. //
2. book
3. @id

QGM representation
Candidate Generation

Auto Indexing – Traverse

Query:
let $doc:=doc("sample.xml")
return for $book in
$doc//book[@xtc:id="book1"]
return $book/title

Candidates:
1. //
2. book
3. @id
4. “book1”

QGM representation
Candidate Generation

Auto Indexing – Traverse

Query:
let $doc:=doc("sample.xml")
return $book/title

Candidates:
1. //</noset>
2. book
3. @id
4. “book1”
...
Auto Indexing – Traverse

Query:
let $doc:=doc("sample.xml")
return $book/title

Candidates:
1. //
2. book
3. @id
4. “book1”

QGM representation

Over 18 enumeration rules and optimization rules
Candidate Selection

Cost-Benefit Calculation

- Rank candidates by benefit
- Observe space limitation (Greedy algorithm)
- Search space pruning (merge indexes)

Containment Problem:

- Comparing PCR sets, but index types and semantics may differ

Reconfigure Index Set

- Schedule materialization and deletion jobs
Outline

- XML Index Types
- Statistics for Cost-based Index Selection
- Index Configuration Management
- Evaluation
- Summary & Outlook
Evaluation

• Workload Documents
  – XMark documents (12MB and 112MB)
  – TPoX collection (~250MB)
  – Standard XML documents (dblp, treebank, psd, …)

• Benchmark Scenarios
  – No Indexes
  – Manual (element + content index)
  – Manual + Self-tuning
  – Self-tuning
Evaluation

XMark workload
Evaluation

Processing time (sec)

workload processing time
- no index
- manual

XMark workload
Evaluation

processing time (sec)

workload processing time

index space

XMark workload

no index
manual

100%
50%
Evaluation

processing time (sec)

workload processing time
- no index
- manual
- manual + self-tuning

index space
- manual (2)

XMark workload
Evaluation

XMark workload
Evaluation

XMark workload
Evaluation

processing time (sec)

workload processing time
- no index
- manual
- manual + self-tuning
- self-tuning

XMark workload

index space
- manual (2)
- manual + self-tuning (2 + 23)
- self-tuning (23)
Evaluation

Impact of parallel indexing

Impact of aggressive indexing
Evaluation

Overhead / Pruning effects

Workload shifts

% of workload

Overhead in %

<10%  <30%  <90%  all

Shift Workload (Query Set on 12MB)
Summary

- Self-tuning of XML indexes causes new challenges to the ISP
- Path Synopsis use for storage, indexing, and managing
- Overhead of statistics and management (mostly) pays off
- Self-tuning of tuning “frequency” and pruning are effective

Outlook

- Integrate Update workload
- Analyze workload shift reaction (stability vs. effectiveness)
- Evaluate XML warehouse scenarios
- Integrate data placement decision
- Combine with buffer tuning
Finish

<www.xtc-project.de />