Index Structures for Matching XML Twigs Using Relational Query Processors

Divesh Srivastava
AT&T Labs–Research
http://www.research.att.com/~divesh/

Joint work with Zhiyuan Chen, Johannes Gehrke, Flip Korn, Nick Koudas, Jayavel Shanmugasundaram
Outline of Talk

- Desiderata, indexing problems
- XML indexes, realized using $B^+$-trees
- Experimental results
<book>
  <title>XML</title>
  <allauthors>
    <author>
      <fn>jane</fn> <ln>poe</ln>
    </author>
    <author>
      <fn>jane</fn> <ln>doe</ln>
    </author>
    <author>
      <fn>john</fn> <ln>doe</ln>
    </author>
  </allauthors>
  <year>2000</year>
  <chapter>
    <title>XML</title>
    <section>
      <head>Origins</head> ...
    </section> ...
  </chapter> ...
</book>
Desiderata

- XML data often stored in relational database systems
- Goal: Develop B\(^+\)-tree index structures that support
  - single index lookup for simple (parent-child) XML path queries
  - efficient evaluation of XML ad hoc, recursive, twig queries
  - tight coupling with relational query processors
Related Work

- APEX [4], DataGuide [10], etc. index paths and data separately
- Index Fabric [5] indexes XML paths and data values together
  - no efficient support for ad hoc, recursive, twig queries
- ViST [25], PRIX [21] support ad hoc, recursive, twig queries
  - uses expensive sub-sequence matching even for simple path queries
- Join Index [24], ASR [14] in OODBMS
  - targeted towards single path queries without recursion
- ToXin [22], XRel [29] index XML paths using relational database
  - ToXin like ASR, XRel requires multiple index lookups
Problems: FreeIndex, BoundIndex

- **PCsubpath**: Query subpath with no intermediate “//”
- **FreeIndex**: Return all matches of PCsubpath in one index lookup
  - allows merge or hash join processing in RDBMS
  - `/book/allauthors/author[fn = ‘jane’]`: ([1,5,6,7], [1,5,41,42])
  - `/book/allauthors/author[ln = ‘doe’]`: ([1,5,21,25], [1,5,41,45])
- **BoundIndex**: Return all matches rooted at d in one index lookup
  - allows index nested loop join processing in RDBMS
  - `/book/title = ‘XML’`: ([1,2])
  - `//author[ln = ‘doe’] rooted at 1`: ([21,25], [41,45])
Relational Representation of All Data Paths

Divesh Srivastava

Index Structures for XML
# ROOTPATHS Index

<table>
<thead>
<tr>
<th>HeadId</th>
<th>SchemaPath</th>
<th>LeafValue</th>
<th>IdList</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
<td>null</td>
<td>[]</td>
</tr>
<tr>
<td>1</td>
<td>BT</td>
<td>null</td>
<td>[2]</td>
</tr>
<tr>
<td>1</td>
<td>BT</td>
<td>XML</td>
<td>[2]</td>
</tr>
<tr>
<td>1</td>
<td>BU</td>
<td>null</td>
<td>[5]</td>
</tr>
<tr>
<td>1</td>
<td>BUA</td>
<td>null</td>
<td>[5, 6]</td>
</tr>
<tr>
<td>1</td>
<td>BUAF</td>
<td>null</td>
<td>[5, 6, 7]</td>
</tr>
<tr>
<td>1</td>
<td>BUAF</td>
<td>jane</td>
<td>[5, 6, 7]</td>
</tr>
<tr>
<td>1</td>
<td>BUAL</td>
<td>null</td>
<td>[5, 6, 10]</td>
</tr>
<tr>
<td>1</td>
<td>BUAL</td>
<td>poe</td>
<td>[5, 6, 10]</td>
</tr>
</tbody>
</table>

- **B⁺-tree on LeafValue.reverse(SchemaPath), returns IdList**
  - solves FreeIndex problem in one index lookup
  - //author[fn = ‘jane’]: use prefix match key (‘jane’, FA*)
  - IdList is key to evaluating branching queries efficiently
DATAPATHS Index

<table>
<thead>
<tr>
<th>HeadId</th>
<th>SchemaPath</th>
<th>LeafValue</th>
<th>IdList</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BUAF</td>
<td>null</td>
<td>[5,6,7]</td>
</tr>
<tr>
<td>1</td>
<td>BUAF</td>
<td>jane</td>
<td>[5,6,7]</td>
</tr>
<tr>
<td>1</td>
<td>BUAL</td>
<td>null</td>
<td>[5,6,10]</td>
</tr>
<tr>
<td>1</td>
<td>BUAL</td>
<td>poe</td>
<td>[5,6,10]</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>UAF</td>
<td>null</td>
<td>[6,7]</td>
</tr>
<tr>
<td>5</td>
<td>UAF</td>
<td>jane</td>
<td>[6,7]</td>
</tr>
<tr>
<td>5</td>
<td>UAL</td>
<td>null</td>
<td>[6,10]</td>
</tr>
<tr>
<td>5</td>
<td>UAL</td>
<td>poe</td>
<td>[6,10]</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **B⁺-tree on LeafValue·HeadId·reverse(SchemaPath)**
  - solves FreeIndex and BoundIndex problems in one index lookup
  - /book as a FreeIndex problem with virtual root as HeadId
  - //author[fn = ‘jane’] as a BoundIndex problem for each book id
  - DATAPATHS index is bigger than ROOTPATHS index
Family of Indexes

<table>
<thead>
<tr>
<th>Index</th>
<th>Subset of SchemaPath</th>
<th>Sublist of IdList</th>
<th>Indexed Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value [17]</td>
<td>paths of length 1</td>
<td>only last ID</td>
<td>SchemaPath, LeafValue</td>
</tr>
<tr>
<td>Forward link [17]</td>
<td>paths of length 1</td>
<td>only last ID</td>
<td>HeadId, SchemaPath</td>
</tr>
<tr>
<td>DataGuide [10]</td>
<td>root-to-leaf path prefixes</td>
<td>only last ID</td>
<td>SchemaPath</td>
</tr>
<tr>
<td>Index Fabric [5]</td>
<td>root-to-leaf paths</td>
<td>only first or last ID</td>
<td>SchemaPath, LeafValue,</td>
</tr>
<tr>
<td>ROOTPATHS</td>
<td>root-to-leaf path prefixes</td>
<td>full IdList</td>
<td>reverse SchemaPath</td>
</tr>
<tr>
<td>DATAPATHS</td>
<td>all paths</td>
<td>full IdList</td>
<td>LeafValue, HeadId,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>reverse SchemaPath</td>
</tr>
</tbody>
</table>

- Given 4-ary relational representation, each index in family
  - stores a subset of all possible SchemaPaths
  - stores a sublist of IdList
  - indexes a subset of HeadId, SchemaPath, and LeafValue
Experimental Setup

- XML data: Stored as Edge Table in IBM’s DB2 version 7.2
  - 100MB scaled XMark data, 50MB DBLP data
- Query workload: 15 XMark queries, 3 DBLP queries

<table>
<thead>
<tr>
<th>Query</th>
<th>Branches</th>
<th>Result Size Per Branch</th>
<th>Depth of Branches</th>
<th>Recursions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_{1_x}$ to $Q_{3_x}$</td>
<td>1</td>
<td>1-11062</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>$Q_{1_d}$ to $Q_{3_d}$</td>
<td>1</td>
<td>1-10258</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>$Q_{4_x}$ to $Q_{9_x}$</td>
<td>2-3</td>
<td>1-7519</td>
<td>High</td>
<td>0</td>
</tr>
<tr>
<td>$Q_{10_x}$ to $Q_{11_x}$</td>
<td>2-3</td>
<td>3-59486</td>
<td>Low</td>
<td>0</td>
</tr>
<tr>
<td>$Q_{12_x}$ to $Q_{15_x}$</td>
<td>2-3</td>
<td>41-20946</td>
<td>Low</td>
<td>1</td>
</tr>
</tbody>
</table>

- Experimental platform: 1.7GHz Pentium, Windows 2000
  - 1GB memory, 37GB disk, 40MB buffer pool, OS cache turned off
## Alternative Indexing Strategies: Space Used

<table>
<thead>
<tr>
<th>Data set</th>
<th>RP</th>
<th>DP</th>
<th>Edge</th>
<th>DG+Edge</th>
<th>IF+Edge</th>
<th>ASR</th>
<th>JI</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMark</td>
<td>119</td>
<td>431</td>
<td>127</td>
<td>169</td>
<td>167</td>
<td>464</td>
<td>822</td>
</tr>
<tr>
<td>DBLP</td>
<td>80</td>
<td>83</td>
<td>106</td>
<td>133</td>
<td>151</td>
<td>93</td>
<td>318</td>
</tr>
</tbody>
</table>

- **ROOTPATHS, DATAPATHS**, with differential IdList encoding
- **Edge Table index**, with value, forward link, backward link
- **DataGuide + backward link on Edge Table**
  - backward link to identify branch point id from leaf id
- **Index Fabric + backward link on Edge Table**
  - backward link to identify branch point id from leaf id
Results: Indexing Schema Paths and Data Values

- Use fully specified path queries with varying selectivity
- Index Fabric, ROOTPATHS, DATAPATHS are best approaches
- Edge, DataGuide perform badly, as selectivity decreases
  - multiple expensive join operations need to be performed
Results: Benefit of Returning IdLists

- Twig queries with varying number, selectivity of branches
- ROOTPATHS, DATAPATHS scale gracefully
- Edge, Index Fabric, DataGuide perform badly
  - 5-way expensive join needs to be performed for each branch
Results: Benefit of Supporting BoundIndex

Twig queries with low branch points.

- Twig queries with selective and unselective branches
- DATAPATHS performs uniformly well
- Edge, Index Fabric, DataGuide, ROOTPATHS perform badly
  - ROOTPATHS does not support index nested loop join (BoundIndex)
Conclusions

- Technical contributions of paper
  - framework for classifying a family of XML indexes
  - propose ROOTPATHS, DATAPATHS with different space-time tradeoffs
  - instantiate indexes using B+-trees in RDBMSs
  - tightly integrate indexes with relational query processor
  - evaluate performance tradeoff between index space and matching time

- Future work: Efficient update techniques