XML Storage and Indexing
Web Data Management and Distribution

Serge Abiteboul  Ioana Manolescu  Philippe Rigaux
Marie-Christine Rousset  Pierre Senellart

Webdam
Web Data Management and Distribution
http://webdam.inria.fr/textbook

September 29, 2011
Outline

1. Introduction
2. XML fragmentation
3. XML identifiers
Motivation

- PTIME algorithms for evaluating XPath queries:
  - Simple tree navigation
  - Translation into logic
  - Tree-automata techniques
- These techniques assume XML data in memory
- Very large XML documents: what is critical is the number of disk accesses
Naïve page-based storage
Processing Simple Paths

- `/auctions/item` requires the traversal of two disk pages;
- `/auctions/item/description` and
  `/auctions/open_auctions/auction/initial` both require traversing three disk pages;
- `//initial` requires traversing all the pages of the document.
Remedies

**Smart fragmentation.** Group nodes that are often accessed simultaneously, so that the number of pages that need to be accessed is globally reduced.

**Rich node identifiers.** Sophisticated node identifiers reducing the cost of join operations.
Outline

1. Introduction

2. XML fragmentation
   - Tag-partitioning
   - Path-partitioning

3. XML identifiers
Simple node identifiers
Partial instance of the *Edge* relation

<table>
<thead>
<tr>
<th>pid</th>
<th>cid</th>
<th>clabel</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>1</td>
<td>auctions</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>item</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>comment</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>name</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>description</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>open_auctions</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>auction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Processing XPath queries with simple IDs

//initial  Direct index lookup.
\[
\pi_{cid}(\sigma_{clabel=\text{initial}}(Edge))
\]

/auctions/item  A join is needed.
\[
\pi_{cid}(\sigma_{clabel=auctions}(Edge) \bowtie_{cid=pid} (\sigma_{clabel=item}(Edge)))
\]

//auction//bid  Unbounded number of joins!

//auction/bid  \[
\pi_{cid}(A \bowtie_{cid=pid} B)
\]
//auction/*/bid  \[
\pi_{cid}(A \bowtie_{cid=pid} Edge \bowtie_{cid=pid} B)
\]
//auction/*//*/bid  \[
\cdots
\]
Outline

1. Introduction

2. XML fragmentation
   - Tag-partitioning
   - Path-partitioning

3. XML identifiers
Reduces the disk I/O needed to retrieve the identifiers of elements having a given tag. Partitioning of queries with // steps in non-leading position remains as difficult as before.
Outline

1. Introduction

2. XML fragmentation
   - Tag-partitioning
   - Path-partitioning

3. XML identifiers
Path-partitioned storage

<table>
<thead>
<tr>
<th>Path</th>
<th>pid</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>/auctions</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>/auctions/item</td>
<td>1 2</td>
<td>1 4</td>
</tr>
<tr>
<td>/auctions/item/name</td>
<td>2 6</td>
<td>4 14</td>
</tr>
</tbody>
</table>

Paths

<table>
<thead>
<tr>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>/auctions</td>
</tr>
<tr>
<td>/auctions/item</td>
</tr>
<tr>
<td>/auctions/item/comment</td>
</tr>
<tr>
<td>/auctions/item/name</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Query evaluation using path partitioning

`//item//bid` can be evaluated in two steps:

- Scan the `path` relation and identify all the parent-child paths matching the given linear XPath query;
- For each of the paths thus obtained, scan the corresponding path-partitioned table.

Many branches still require joins across the relations. For very structured data, the `path` relation is typically much smaller than the data set itself. Thus, the cost of the first processing step is likely negligible, while the performance benefits of avoiding numerous joins are quite important.
Outline

1. Introduction

2. XML fragmentation

3. XML identifiers
   - Region-based identifiers
   - Dewey-based identifiers
   - Structural identifiers and updates
Identifiers

Within a persistent XML store, each node must be assigned a unique identifier (or ID, in short). We want to encapsulate structure information into these identifiers to facilitate query evaluation.
Outline

1. Introduction

2. XML fragmentation

3. XML identifiers
   - Region-based identifiers
   - Dewey-based identifiers
   - Structural identifiers and updates
Region-based identifiers

The so-called region-based identifier scheme simply assigns to each XML node $n$, the pair composed of the offset of its begin tag, and the offset of its end tag. We denote this pair by $(n.\text{begin}, n.\text{end})$.

- the region-based identifier of the `<a>` element is the pair (0, 90);
- the region-based identifier of the `<b>` element is pair (30, 50).
Using region-based identifiers

Comparing the region-based identifiers of two nodes $n_1$ and $n_2$ allows deciding whether $n_1$ is an ancestor of $n_2$. Observe that this is the case if and only if:

- $n_1.start < n_2.start$, and
- $n_2.end < n_1.end$.

No need to use byte offset:

- (Begin tag, end tag). Count only opening and closing tags (as one unit each) and assign the resulting counter values to each element.
- (Pre, post). Preorder and postorder index (see next slide).
- (Pre, post, depth). The depth can be used to decide whether a node is a parent of another one.

Region-based identifiers are quite compact, as their size only grows logarithmically with the number of nodes in a document.
(pre, post, depth) node identifiers
Outline

1. Introduction

2. XML fragmentation

3. XML identifiers
   - Region-based identifiers
   - Dewey-based identifiers
   - Structural identifiers and updates
Dewey node identifiers

1. auctions
   1.1 item
     1.1.1 comment
     1.1.2 name
     1.1.3 description
   1.2 open_auctions
     1.2.1 auction
       1.2.1.1 initial
       1.2.1.2 bids
     1.2.2 auction
       1.2.2.1 initial
       1.2.2.2 bids
   1.3 item
     1.3.1 name
     1.3.2 description
Using Dewey identifiers

- Ancestor decided by looking if an ID is a prefix of another one (largest non-identical prefix for parent).
- Possible to decide preceding-sibling, following-sibling, preceding, following.
- Given two Dewey IDs $n_1$ and $n_2$, one can find the ID of the lowest common ancestor (LCA) of the corresponding nodes. The ID of the LCA is the longest common prefix of $n_1$ and $n_2$. Useful for keyword search.

Main drawback: length (large, variable) of the identifiers.
Outline

1. Introduction

2. XML fragmentation

3. XML identifiers
   - Region-based identifiers
   - Dewey-based identifiers
   - Structural identifiers and updates
Consider a node with Dewey ID 1.2.2.3. Suppose we insert a new first child to node 1.2. Then the ID of node 1.2.2.3 becomes 1.2.3.3.
In general:

- Offset-based identifiers need to be updated as soon as a character is inserted or removed in the document.
- (start, end), (pre, post), Dewey IDs need to be updated when the elements of the documents change.
- Possible to avoid relabeling on deletions, but gaps will appear in the labeling scheme.
- Relabeling operation quite costly.