Query Optimization for XML
By Jason Mchugh, Jennifer Widom
Dazhi Wang
CPS296.1
Topics in Database Systems

Outline
- Basics for Lore system
- Motivation
- Query Execution Engine
  - Logical Query Plans
  - Statistics and Cost Models
  - Physical Plan Enumeration
- Performance Results

Introduction
- Lore System
  - DBMS for semistructured data
  - Data model: OEM
- What complicate query optimization for XML
  - Variety of indexes and traversal techniques increases the search space
  - How to choose appropriate set of statistics for graph-based data and how to compute and store them efficiently

OEM model for XML
- OEM model: a labeled directed graph
  - Data in OEM is schema-less and self-describing
  - Atomic objects: vertices with data, no outgoing edge
  - Complex objects: vertices with outgoing edge
  - Labeled edges: express property relationships
  - Names: special labels recognizing single objects
- Correspondence between OEM and XML
  - OEM’s objects: elements in XML
  - Subobject relationship in OEM: element nesting in XML
  - Difference: subelements in XML are inherently ordered; XML elements may include a list of attribute-value pairs

OEM model for XML (cont.)
- Simple Path Expression
  - Represents a single-step navigation in the database. Has the form $x.l\ y$
  - e.g. $DBGroup.Project\ x$
- Path Expression
  - Ordered list of simple path expressions
  - e.g. $DBGroup.Member\ x, x.Age\ y$
Lore’s Query Language: Lorel

- Lorel is an extension of OQL, which is an SQL-like declarative language.
- Example:  
  Select x  
  From DBGroup.Member x  
  Where exists y in x.Age: y<30
- Operands:  
  - constants, named objects, variables
- Operations:  
  - Selection, set, quantification, aggregation
- Path expression can appear in select, from or where clause

Lore Query Processing

```
Lore query \rightarrow \text{parse} \rightarrow \text{process} \rightarrow \text{Logical generator} \rightarrow \text{Logical query plan}
```

```
Physical plan \rightarrow \text{choose best plan} \rightarrow \text{Best physical plan}
```

Lore Indexes

- **Vindex**  
  - Find atomic objects given label and predicates
- **Lindex**  
  - Find parents given object and label
- **Bindex**  
  - Find parent-child pairs given an edge label
- **Pindex**  
  - Find the object given a path

Path to a node is as important as the node’s value

Motivation

- The optimal query plan depends on  
  - Values in the database  
  - Shape of the graph model

\[ \downarrow \]

Optimization is both important and difficult

Motivation (cont.)

- e.g.: Select x From A.B x where exists y in x.C: y=5
- Strategy: top-down, bottom-up, hybrid

Query Execution Engine

- **Variable binding**: assign an object to a variable in the query
- **Evaluation**: a list of all variables in the plan along with the object bound to each variable
- The goal: iteratively generate complete evaluations to produce query results
Logical Query Plans

- To execute:
  Select x From DBGroup.Member x Where exists y in x.Age: y < 30
  - Top-down approach: handle From clause first
  - Bottom-up approach: handle where clause first
- For the where clause:
  - Break it into: (a) find all Age subobjects of x
    - (b) test their values
  - Top-down approach: first (a) then (b)
  - Bottom-up approach: first find (b) using Vindex, then (a) using Lindex

Logical Query Plans (cont.)

- Solution:
  - break the query into independent components, where execution order not fixed
  - Rotation points: where the components meet
  - Build a tree of logical operators based on independent components
    - Discover(x, "B", y): bind x and y
    - Chain(r, l): link left & right subplan
    - Glue(r, l): represent a rotation point
  - Each logical operator constructs its optimal physical plan

Logical Query Plans (cont.)

- A complete logical query plan for the previous example

Physical Query Plan

- Physical query plan operators are iterators
  - Request a tuple at a time from its children
  - Performs some operation
  - Returns the result to the parent
  - The “tuples” are evaluations
- Six basic traversal operators:
  - Scan(x, l, y): place into y all subobjects of x connected via l.
  - Lindex(x, l, y): place into x all parents of y connected via l.
  - Pindex(pathexp, x):
  - Bindex(l, x, y): all x-y pairs connected via l
  - Name(x, n): verify if x is the named obj n.
  - Vindex(Op, value, l, x): place into x all atomic objects satisfying “Op value” and having incoming label l.

Physical Query Plan (cont.)

- A complete physical query plan for the previous example

Traversal Operators

- Scan(x, l, y): place into y all subobjects of x connected via l.
- Lindex(x, l, y): place into x all parents of y connected via l.
- Pindex(pathexp, x): all x-y pairs connected via l
- Name(x, n): verify if x is the named obj n.
- Vindex(Op, value, l, x): place into x all atomic objects satisfying “Op value” and having incoming label l.
Statistics and Cost Model

- To estimate the execution cost for a given physical query plan

Statistics

- Need to consult:
  - size & shape of the database
  - range of values
  - Tradeoff between time & space and accuracy

Statistics (cont.)

- For every label subpath p of length ≤ k
  - ∀ atomic type, # atomic objs (of type) reachable
  - ∀ atomic type, min & max values reachable
  - Total # of instances of path p, |p|
  - Total # of distinct objs reachable, |p|₂
  - Total # of l-labeled sub-objects reachable, |p|₃
  - Total # of incoming l-labeled edges, |p|₄

Statistics (cont.)

- Example: evaluate A.B x, x.C y
  - Top-down: need to estimate # C subobjects
    - fan-out: |p| * (|p| / |p|₄)
  - Bottom-up: need to estimate # parents via a B edge for all C’s
    - fan-in: |p| * (|p| / |p|₄)
  - Accurate for (k+1) path, estimate (k+2) path using (k+1) path

Cost Model

- Execution cost:
  - I/O cost = predicted # of objects fetched
  - Rough estimation, due to lack of clustering knowledge in Lore
  - CPU cost – secondary to I/O
  - Expected # of evaluations
  - Node-cost = children cost + self cost
  - The cost model is Simplistic

Plan Enumeration

- Large search space of physical query plan
  - Path of length n ⇒ n-way join
  - Interrelated path expressions
  - Set operation, Sub-query, aggregation, etc.
  - Solution: Greedy approach
    - A logical plan node makes a locally optimal decision based on bound variables passed from its parents
    - May make optimal physical subplan many times
    - Problem: still explore exponentially large # of plan

Aggressive Pruning

- Join 2 simple paths iff sharing common variable
- Pindex operator only starts from a named object, & no variable except the last is used elsewhere
- Select clause always executes last
  - Separate From and Where clauses
  - No reorder on multiple independent paths

<table>
<thead>
<tr>
<th>n: number of simple path expressions</th>
<th>n⁻¹</th>
<th>n⁻²</th>
<th>n⁻³</th>
</tr>
</thead>
<tbody>
<tr>
<td>All possible plans Lore's search space</td>
<td>1456/48</td>
<td>2361/96/228</td>
<td>3,853/137/292/1048</td>
</tr>
</tbody>
</table>

Table 1: Analysis of Search Space Size
Producing physical plans

- Each logical plan node creates optimal local physical plan given a set of bound variables
- Need to track for a variable:
  - bound or not
  - Which operator bound it
  - All other operators that use it
  - Whether it is stored in a temporary result

*Example: Discover(x.Age y)*
- x is bound: scan operator may be optimal
- y is bound: Lindex operator may be optimal

### Performance Results

- Experiment over a 5 MB database on movies
- Indexes size: 8.1 MB
- 62,256 nodes
- 130,402 edges

Avg. optimization time per query ~ ½ sec.
Performance Results (cont.)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.33</td>
<td>Bottom-up</td>
</tr>
<tr>
<td>2*</td>
<td>3.68</td>
<td>Top-down</td>
</tr>
<tr>
<td>3</td>
<td>6.95</td>
<td>Hybrid with Pindex</td>
</tr>
<tr>
<td>4</td>
<td>7.01</td>
<td>Hybrid with pointer-chasing</td>
</tr>
<tr>
<td>5</td>
<td>23.13</td>
<td>Bindex and Pindex then Lindex</td>
</tr>
</tbody>
</table>

a query with two existentially quantified variables

movies with Genre subobject having value “Comedy”

Summary

- Logical Plan
- New Indexes
- Physical Plans
  - Aggressive Pruning
  - Best physical plans