ROLEX

Relational On-Line Exchange with XML

Phil Bohannon,
Sumit Ganguly,
Hank Korth,
PPS Narayan,
Pradeep Shenoy

Database Principles Research Department
Information Sciences Research Center
Bell Labs
Evolution of Applications

- In the beginning was client-server
- Many layers of software later, applications are portable and browser-based…but slow!
- So, cache some data, manage consistency in application
- Newest applications speak XML
- Supported by XML Publishing through View Queries
- Even slower, more caching
Example: A ROLEX View Query

- **Relational Schema**
  - Metroarea (metroid, metroname)
  - Hotel (hotelid, hotelname, starrating, chain_id, metro_id, city, pool, gym)
  - Guestroom (r_id, roomnumber, type, rackrate)
  - Confroom (c_id, chotel_id, croomnumber, capacity, rackrate)

- **Desired Hierarchical Structure**

```
<metroarea>
  <hotel>
    <confstat>
    <guestrooms>
```
Example: Add “tag queries”

$m = \text{SELECT metroid, metroname FROM metro}$

$h = \text{SELECT hotelid, hotelname, starrating, state_id FROM hotel WHERE metro_id = $m.metroid AND starrating > 4}$
Example: “binding variables”

$m = \text{SELECT metroid, metroname FROM metro}

$h = \text{SELECT hotelid, hotelname, starrating, state_id FROM hotel}
\text{WHERE metro_id = } m.\text{metroid}
\text{AND starrating > 4}
In the beginning was client-server

Many layers of software later, applications are portable and browser-based…but but slow!

So, cache some data, manage consistency in application

Newest applications speak XML

Even slower, more caching

Worse, applications need to interoperate for OLTP
Current XML Publishing

Issues
- Inherent inefficiencies
- DOM Tree = Cache?
- Updates?
Six Steps to ROLEX: Step 1

1. Combine DBMS and Publisher (many vendors)
Six Steps to ROLEX: Step 2

1. DBMS += Publisher
2. Optimize to produce tree results
Six Steps to ROLEX: Step 3

1. DBMS += Publisher
2. Optimize to produce trees
3. Main-Memory DBMS

DOM Tree
Logic
Parser
Application

View Query
XML Text

Optimize Tree Query
Tagger
Tree
Execute

DataBlitz™
DBMS
Six Steps to ROLEX: Step 4

4. Share memory, produce DOM tree directly, eliminate parsing
Six Steps to ROLEX: Step 5

4. Share memory, produce DOM tree directly, eliminate parsing
5. Virtual DOM, lazy evaluation
Six Steps to ROLEX: Step 6

View Query + Navigation Profile

Optimize Tree Query

Execute

Logic

DOM Tree

DataBlitz™

Application

6. Optimize for the expected navigation profile
Considering Expected Use

- Recent interest in making use of user feedback during long queries
- MQO takes advantage of co-incident query execution
- What about consecutive execution?
- OLTP interaction with DBMS very stylized, yet little work looks at pattern of related queries
- Notable exception: [Florescu, Levy, Suciu, Yagoub, 1999]
Navigational Profile

- Navigational Profile, an input to query optimization
- Simple approach to expected use for a tree-based data model
- Example Profile:

```
<metro>

<hotel>

Pr{hotel|metro} -> 0.2

<confstat>

Pr{confstat|hotel} -> 1.0

<guestrooms>
```

Pr{confstat|hotel} -> 0.1
Remainder of Talk

- The Virtual DOM
  - “Document Object Model”
- A Decorrelation Plan Space
- A VOLCANO-Based Optimizer
- Cost Estimation
- Performance
- Conclusion
Let the query for a schema-tree node be $Q_b(s_1, .., s_k)$ where $s_1, .., s_k$ are parameters and $b$ is a binding variable.
The Virtual DOM

- A query “sub-plan” and navigation index, NI, for each node
  - maps s1,...,sk to query results or “absent”
  - supports sibling navigation
  - Allow support for DOM operations within DB
- If s1,...,sk “absent” in NI, execute sub-plan for node.
**Decorrelation Plan-Space**

- Usual strategy: *decorrelate as much as possible* to explore alternate join orders

- ROLEX strategy: may want to leave sub-query correlated to avoid work when navigation probability is low

- Approach: develop a decorrelation plan-space and let the optimizer decide what to do
$m = \text{SELECT metroid, metroname FROM metro}$

$h = \text{SELECT hotelid, hotelname, starrating, state_id FROM hotel, metro WHERE metro_id = metroid AND starrating > 4 GROUP BY hotelid}$

$Q_{h}(m)$

$Q_{h}^{m}()$
Alternatives for $Q_v(m,a)$:

$$\{Q_v(m,a), Q_v^a(h,m), Q_v^{a,h}(m), Q_v^{a,h,m}()\}$$
Notes

We always replace bottom query rather than top query

- Keeps space from being exponential
- Outer-join rules avoided
- Cost of greater emphasis on CSE required by optimizer
VOLCANO-Based Optimizer

- VOLCANO [Graefe McKenna 1993]-style optimizer [Roy 2000] modified to support ROLEX optimization

- New operators (NavNodes) added to model schema-tree nodes from the query

- Decorrelation plan-space modeled as distinct but equivalent NavNodes

- VOLCANO automatically collapses equivalent plan nodes across the tree
**Cost Model**

**Goal:**
- estimate expected unique calls for each node *given a Navigation Profile*

**Approach:**
- Estimate, for each schema-tree node, \( n \), in query plan:
  - …expected # visits: \( EVis(n) \)
  - …expected output from one call: \( ESize(n) \)
  - …exp. unique parameter bindings: \( EUniqBind(s_1,..,s_k) \)

**Unique calls** = \( \min(EVis(n), EUniqBind(s_1,..,s_k)) \)
Materialization

- For individual query operators, consider only the binding parameters present in that operator.

- Materialize “opportunistically” as bindings increase:
  - Compute this sub-expression only when $m$ changes.
  - Generalizes [Rao Ross 98]

- If $EVis(n) > EUniqBind(s1,..,sk)$ for a node, then materialize that node’s result.
Experimental Setup

- **Execution engine built on top of DataBlitz™ tuple-layer interface**
  - Handles complex queries including group-by and aggregation

- **Virtual DOM under construction**

- **Prototype for performance testing scans tree result**
  - Obeys navigation probabilities
  - Does a local ‘sprintf’ to model consumption of a node
Experimental Results

- First, validate the basic idea:

- Vary the size of a simple view query

- Compare time needed to:
  - Execute query and traverse results
  - Parse result with Xerces C++ DOM
  - Parse result with ... IDOM

![Graph showing time in milliseconds vs. probability of exploring <avail> nodes]
Do navigation probabilities matter?

Existence proof...

Create view v1 as
<hotel>
($h =
Select hotelid, name, 
   starrating, state_id
From hotel)
</hotel>

<avail>
($a =
SELECT rhotel_id, startdate, roomnumber 
   FROM availability, guestroom 
   WHERE type > 5 and rhotel_id = $h.hotelid 
       AND startdate > 12/15/02 and r_id = a_r_id
)
</avail>

Optimizer says:
P1: 0 to .15
P2: .15 to .25
P3: .25 to 3.0
Size of Plan Space

- Start with a complicated query
- Vary the probabilities on a subset of the edges so that expected number of output tuples varied exponentially (1, 2, 4, 8, …)
- While holding the other probabilities constant at 1 (or 0).
- Compute the number of distinct plans generated
# Importance of Navigation Profile

## H-S-V

### $Q_{m}$

Select metroid, metroname from metroarea.

### $Q_{s}$

Select SUM(capacity) from confroom where chotel_id = $h$.hotelid.

### $Q_{c}$

Select * from confroom where chotel_id = $h$.hotelid.

## H-A-C

### $Q_{h}$

Select * from hotel where metro_id = $m$.metroid and starrating > 4.

### $Q_{a}$

Select COUNT(a_id) from availability_guestroom where hotel_id = $h$.hotelid and startdate <= 1/1/01 and enddate >= 2/1/01 and a_r_id = r_id group by startdate.

### $Q_{v}$

Select COUNT(a_id) from availability, hotel, guestroom where hotel_id = hotelid and metro_id = $m$.metroid and startdate >= $a$.startdate and enddate >= 2/1/01 and a_r_id = r_id.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Binding Var.</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;metro&gt;</td>
<td>$m$</td>
<td></td>
</tr>
<tr>
<td>$Q_{m}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;hotel&gt;</td>
<td>$h$</td>
<td>$m$</td>
</tr>
<tr>
<td>$Q_{h}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;confstat&gt;</td>
<td>$s$</td>
<td>$h$</td>
</tr>
<tr>
<td>$Q_{s}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;confroom&gt;</td>
<td>$c$</td>
<td>$h$</td>
</tr>
<tr>
<td>$Q_{c}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Binding Var.</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;hotel_available&gt;</td>
<td>$a$</td>
<td>$h$</td>
</tr>
<tr>
<td>$Q_{a}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;metro_available&gt;</td>
<td>$v$</td>
<td>$a$, $m$</td>
</tr>
<tr>
<td>$Q_{v}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Number of plans generated
Summary: ROLEX Benefits

- Lazy Evaluation
- Optimize for expected use
  - Cost model
  - Materialization
- Use DBMS data structures to support navigation
- Replace application cache
- Potential to support updates via DOM
- Incremental support for new XML tools
Related Work

Relational Publishing:
- XPERANTO (IBM, Uwisc, Cornell)
- SilkRoute (AT&T & UWash)
- Both compose application query and view into one expression that is optimized
- Neither supports navigation: full document must be materialized

[Florescu, Levy, Suciu, Yagoub, 1999]
- considers navigation probabilities on a web site,
- performs a variety of heuristic optimizations

[Ludascher, Papakonstantinou, Velikhov, LNCS 2000]
- lazy evaluation of DOM tree
- For slow sources, not optimized for expected pattern
Future Work

- Complete Virtual DOM
- Updates
- Key applications -- XSLT
- Beyond Navigation Profiles
  - More sophisticated profiles
  - Adaptability
- Optimizing large, complex correlated queries