Materialized views for P2P XML warehousing

Ioana Manolescu\textsuperscript{1} \quad Spyros Zoupanos\textsuperscript{1}

\textsuperscript{1}GEMO/IASI group, INRIA Saclay – Île-de-France

21 October 2009
BDA 2009, Namur
Problem statement and contributions

Problem

efficient and flexible XML content sharing in peer-to-peer networks based on a DHT.
Problem statement and contributions

**Problem**

efficient and flexible XML content sharing in peer-to-peer networks based on a DHT.

**Contributions**

- An architecture for flexible management of XML materialized views on a DHT
- View advertisement algorithms
- Query rewriting algorithms
- ViP2P: full platform for distributed XML view management
XML materialized views on a DHT

Declare tree pattern XML views over the network data

Fill in the views with XML data

Answer tree pattern queries using the existing views

- View definition lookup
- Query rewriting $\Rightarrow$ logical plan
- Translation to a (distributed) physical plan
- Execution of the physical plan
Architecture overview

Materialized views for P2P XML warehousing
The peers may store:
- documents
The peers may store:
- documents
- views
When $q$ arrives:
When q arrives:

- view definition
- lookup
When $q$ arrives:
- view definition
- rewriting
When \( q \) arrives:
- view definition lookup
- rewriting
- execution of physical plan
When \( d \) arrives:

- Search view definitions for which \( v_i(d) \neq \emptyset \)
- Compute \( v_i(d) \)
- Send results
When \( d \) arrives:
- search view definitions for which \( \nu_i(d) \neq \emptyset \)
When \( d \) arrives:
- search view definitions for which \( v_i(d) \neq \emptyset \)
- compute \( v_i(d) \)
When $d$ arrives:
- search view definitions for which $v_i(d) \neq \emptyset$
- compute $v_i(d)$
- send results

Materialized views for P2P XML warehousing
Tree patterns for views and queries

\[ \text{some text} \]

\[ b_{cont} \]
Tree patterns for views and queries

Materialized views for P2P XML warehousing
Tree patterns for views and queries

```
< b > < c > < d / > < e > some < / e > < / c > < / b >
< b > < b > < g > text < / g > < / b > < h / > < / b >
```

Materialized views for P2P XML warehousing
Tree patterns for views and queries

![Tree pattern diagram]

- `<b><c><d /></e>some</e></c></b>`
- `<b><g>text</g></b>`
- `<b><g>text</g></b>`
Tree patterns for views and queries

\[ \text{Tree representation with nodes labeled with coordinates:} \]

- \( a(1,12) \)
- \( b(2,5) \)
- \( f(7,6) \)
- \( b(8,11) \)
- \( c(3,4) \)
- \( b(9,9) \)
- \( h(12,10) \)
- \( d(4,1) \)
- \( e(5,3) \)
- \( g(10,8) \)
- \( \text{some}(6,2) \)
- \( \text{text}(11,7) \)

**b_{id,cont}**

**b_{id} \quad b_{cont}**
Tree patterns for views and queries

Materialized views for P2P XML warehousing
Tree patterns for views and queries

```
  a(1,12)
 /    |
 b(2,5) f(7,6) b(8,11)
 |
 c(3,4) b(9,9) h(12,10)
 |
 d(4,1) e(5,3) g(10,8)
 |
 some(6,2) text(11,7)
```

<table>
<thead>
<tr>
<th>$b_{id}$</th>
<th>$b_{cont}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2,5)</td>
<td><code>&lt;b&gt;&lt;c&gt;&lt;d/&gt;&lt;e&gt;some&lt;/e&gt;&lt;/c&gt;&lt;/b&gt;</code></td>
</tr>
<tr>
<td>(8,11)</td>
<td><code>&lt;b&gt;&lt;b&gt;&lt;g&gt;text&lt;/g&gt;&lt;/b&gt;&lt;h/&gt;&lt;/b&gt;</code></td>
</tr>
</tbody>
</table>
Tree patterns for views and queries

\begin{center}
\begin{tabular}{c|c}
\hline
$id$ & $cont$ \\
\hline
(2,5) & $<b><c><d/><e>some</e></c></b>$ \\
(8,11) & $<b><b><g>text</g></b><h/></b>$ \\
(9,9) & $<b><g>text</g></b>$ \\
\hline
\end{tabular}
\end{center}
Tree patterns for views and queries

```
  a_{(1,12)}
   /   \\    \
  b_{(2,5)} f_{(7,6)} b_{(8,11)}
      / \     / \     / \      \
    c_{(3,4)} b_{(9,9)} h_{(12,10)}
        /     |      /     |
      d_{(4,1)} e_{(5,3)} g_{(10,8)}
           /     |      /     |
         some_{(6,2)} text_{(11,7)}

a_{id}     b_{val}
```

Materialized views for P2P XML warehousing
Tree patterns for views and queries

\[
\begin{align*}
\text{a}_{(1,12)} & \quad \mid \\
\text{b}_{(2,5)} & \quad \mid \\
\text{c}_{(3,4)} & \quad \mid \\
\text{d}_{(4,1)} & \quad \mid \\
\text{e}_{(5,3)} & \quad \mid \\
\text{f}_{(7,6)} & \quad \mid \\
\text{g}_{(10,8)} & \quad \mid \\
\text{h}_{(12,10)} & \quad \mid \\
\text{b}_{(9,9)} & \\
\text{a}_{id} &
\end{align*}
\]

\[
\begin{align*}
\text{some}_{(6,2)} & \quad \mid \\
\text{text}_{(11,7)} &
\end{align*}
\]

\[
\begin{align*}
\text{a}_{id} \quad & \text{b}_{val} \\
(1,12) \quad & \text{some}
\end{align*}
\]
Tree patterns for views and queries

**Tree Patterns**

- \(a_{id}(1,12)\)
- \(b_{val}(2,5)\)
- \(f_{val}(7,6)\)
- \(b_{id}(8,11)\)
- \(c_{id}(3,4)\)
- \(b_{id}(9,9)\)
- \(h_{id}(12,10)\)
- \(d_{id}(4,1)\)
- \(e_{id}(5,3)\)
- \(g_{id}(10,8)\)
- \(\text{some}(6,2)\)
- \(\text{text}(11,7)\)

**Table**

<table>
<thead>
<tr>
<th>(a_{id})</th>
<th>(b_{val})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,12)</td>
<td>some</td>
</tr>
<tr>
<td>(1,12)</td>
<td>text</td>
</tr>
</tbody>
</table>
Tree patterns for views and queries

```
 a_{val}  b_{cont}  f_{id}
```

```
  \textcolor{red}{a}_{(1,12)}
     \downarrow
   \quad \quad \downarrow
  b_{(2,5)} \quad f_{(7,6)} \quad b_{(8,11)}
     \quad \downarrow
  c_{(3,4)} \quad b_{(9,9)} \quad h_{(12,10)}
     \quad \quad \downarrow
 d_{(4,1)} \quad e_{(5,3)} \quad g_{(10,8)}
     \quad \quad \downarrow
  \textcolor{red}{\text{some\text{\small\text{\textcolor{red}{text}}}}}_{(6,2)}_{(11,7)}
```
Tree patterns for views and queries

```
a_{val}
  b_{cont}
     f_{id}

\begin{array}{ccc}
  a_{val} & b_{cont} & f_{id} \\
  \text{some text} & <b><c><d/>><e>some</e></c></b> & (7,6) \\
\end{array}
```
Tree patterns for views and queries

\[
\begin{array}{c}
\text{a}_{(1,12)} \\
\text{b}_{(2,5)} \quad \text{f}_{(7,6)} \quad \text{b}_{(8,11)} \\
\text{c}_{(3,4)} \quad \text{b}_{(9,9)} \quad \text{h}_{(12,10)} \\
\text{d}_{(4,1)} \quad \text{e}_{(5,3)} \quad \text{g}_{(10,8)} \\
\text{some}_{(6,2)} \quad \text{text}_{(11,7)} \\
\end{array}
\]

<table>
<thead>
<tr>
<th>a_{val}</th>
<th>b_{cont}</th>
<th>f_{id}</th>
</tr>
</thead>
<tbody>
<tr>
<td>some text</td>
<td>&lt;b&gt;&lt;c&gt;&lt;d/&gt;&lt;e&gt;some&lt;/e&gt;&lt;/c&gt;&lt;/b&gt;</td>
<td>(7,6)</td>
</tr>
<tr>
<td>some text</td>
<td>&lt;b&gt;&lt;b&gt;&lt;g&gt;text&lt;/g&gt;&lt;/b&gt;&lt;h/&gt;&lt;/b&gt;</td>
<td>(7,6)</td>
</tr>
</tbody>
</table>
Tree patterns for views and queries

<table>
<thead>
<tr>
<th>$a_{val}$</th>
<th>$b_{cont}$</th>
<th>$f_{id}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>some text</td>
<td>$&lt;b&gt;&lt;c&gt;&lt;d/&gt;&lt;e&gt;some&lt;/e&gt;&lt;/c&gt;&lt;/b&gt;$</td>
<td>(7,6)</td>
</tr>
<tr>
<td>some text</td>
<td>$&lt;b&gt;&lt;b&gt;&lt;g&gt;text&lt;/g&gt;&lt;/b&gt;&lt;h/&gt;&lt;/b&gt;$</td>
<td>(7,6)</td>
</tr>
<tr>
<td>some text</td>
<td>$&lt;b&gt;&lt;g&gt;text&lt;/g&gt;&lt;/b&gt;$</td>
<td>(7,6)</td>
</tr>
</tbody>
</table>
Algebraic rewriting & operators

Let $q \in \mathcal{P}$ be a query and $\mathcal{V} = \{v_1, v_2, \ldots, v_k\}$ a set of views. A rewriting of $q$ using $\mathcal{V}$ is an algebraic expression

$$e(v_1, v_2, \ldots, v_k)$$

such that $e(\mathcal{D}) = q(\mathcal{D})$ for any document set $\mathcal{D}$
Algebraic rewriting & operators

Let \( q \in \mathcal{P} \) be a query and \( \mathcal{V} = \{ v_1, v_2, \ldots, v_k \} \) a set of views. A **rewriting** of \( q \) using \( \mathcal{V} \) is an algebraic expression

\[
e(v_1, v_2, \ldots, v_k)
\]

such that \( e(D) = q(D) \) for any document set \( D \)

**Algebra operators**

- \( \text{scan}(v) \)
- \( \pi_{\text{cols}}(op) \)
- \( \text{sort}_{\text{cols}}(op) \)
- \( \text{nav}_{i, np}(op) \) evaluates \( np \) over the \( \text{cont} \) attribute \( op.i \)

- \( \times \)
- \( \pi^0(op) \)
- \( \sigma_{\text{cond}}(op) \)
Navigation example

\[
\begin{array}{c}
a_{(1,12)} \\
\quad | \\
\quad b_{(2,5)} \quad f_{(7,6)} \quad b_{(8,11)} \\
\quad | \\
\quad c_{(3,4)} \quad b_{(9,9)} \quad h_{(12,10)} \\
\quad | \\
\quad d_{(4,1)} \quad e_{(5,3)} \quad g_{(10,8)} \\
\quad | \\
\quad \text{some}_{(6,2)} \quad \text{text}_{(11,7)}
\end{array}
\]

\[
\overline{b_{\text{cont}}} \quad \text{nav}_{b_{\text{cont}},g_{\text{val}}}(b_{\text{cont}})
\]
Navigation example

\[
\begin{align*}
\text{a}(1,12) & \quad \text{b}(2,5) \quad \text{f}(7,6) \quad \text{b}(8,11) \\
\quad & \quad \text{c}(3,4) \quad \text{b}(9,9) \quad \text{h}(12,10) \\
\text{d}(4,1) \quad \text{e}(5,3) \quad \text{g}(10,8) & \quad \text{some}(6,2) \text{text}(11,7)
\end{align*}
\]

\[
\text{b}_{\text{cont}}
\]

\[
\begin{array}{c}
\text{b}_{\text{cont}} \\
\text{nav}_{b,\text{cont},g_{\text{val}}}(b_{\text{cont}})
\end{array}
\]

\[
\begin{align*}
\langle b \rangle & \langle c \rangle \langle d \rangle \langle e \rangle \text{some} \langle / e \rangle \langle / c \rangle \langle / b \rangle \\
- & \quad -
\end{align*}
\]
Navigation example

```
Materialized views for P2P XML warehousing
```
Navigation example

1. **Algebraic query rewriting**
2. **Rewriting algorithms**
3. **View management**
4. **Experiments**
5. **Conclusion**

**Navigation example**

```
\[ \begin{array}{c}
\text{a}(1,12) \\
| \quad | \quad | \\
\text{b}(2,5) \quad \text{f}(7,6) \quad \text{b}(8,11) \\
| \quad | \quad | \\
\text{c}(3,4) \quad \text{b}(9,9) \quad \text{h}(12,10) \\
| \quad | \\
\text{d}(4,1) \quad \text{e}(5,3) \quad \text{g}(10,8) \\
| \\
\underline{\text{some}(6,2)} \underline{\text{text}(11,7)} \\
\end{array} \]
```

<table>
<thead>
<tr>
<th>(b_{cont} )</th>
<th>( \text{nav}<em>{b</em>{cont},g_{val}}(b_{cont}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;b&gt;&lt;c&gt;&lt;d/ &gt;&gt;e&lt;/e&gt;&lt;/c&gt;&lt;/b&gt;)</td>
<td>-</td>
</tr>
<tr>
<td>(&lt;b&gt;&lt;b&gt;&lt;g&gt;text&lt;/g&gt;&lt;/b&gt;&lt;h/&gt;)</td>
<td>\text{text}</td>
</tr>
<tr>
<td>(&lt;b&gt;&lt;g&gt;text&lt;/g&gt;&lt;/b&gt;)</td>
<td>\text{text}</td>
</tr>
</tbody>
</table>
Rewriting example

$q$

$a$

$b$

$d_{[val=5]}$

$c_{cont}$

$e_{val}$

$v_1$

$v_2$

$v_3$

$a_{id}$

$a_{id}$

$b_{id,cont}$

$c_{id,cont}$

$d_{val}$
Rewriting example

\[
\begin{align*}
q & \quad \pi_{c.cont, e.val} \\
\sigma_{a.id = a.id \land b.id < c.id \land a.id < b.id \land d.val = 5} \\
\times & \quad nav_{b.cont, e.val} \\
\end{align*}
\]
Finding query rewritings

Idea:
Compute covers of the query nodes with the view nodes.
Finding query rewritings

Idea:
Compute covers of the query nodes with the view nodes.

\[ q \quad v_1 \quad v_2 \]

\[ a_{id} \]
\[ b_{id} \quad a_{id} \quad b_{id} \]
Finding query rewritings

Idea:
Compute covers of the query nodes with the view nodes.
Finding query rewritings

Idea:
Compute covers of the query nodes with the view nodes.

\[ q \]
\[ \begin{array}{c}
  a_{id} \\
  \downarrow \\
  b \\
  \downarrow \\
  d_{val}
\end{array} \quad \begin{array}{c}
  a_{id,cont} \\
  \downarrow \\
  b \\
  \downarrow \\
  c \\
  \downarrow \\
  d_{val}
\end{array} \]
Finding query rewritings

Idea:

Compute covers of the query nodes with the view nodes.

```
  q
   a
    b
     c_{val}
   d_{val}

  v_1
    a_id
     b
      b_id
        c_{val}
      d_{val}

  v_2
    a_id
     b_id
        e
        d_{val}
```
Finding query rewritings

Idea:
Compute covers of the query nodes with the view nodes.

No rewriting
Finding query rewritings

Idea:
Compute covers of the query nodes with the view nodes.

\[ q \]

\[ v_1 \]
\[ a \]
\[ b \]
\[ c \]
\[ d_{val} \]
\[ e_{val} \]

\[ v_2 \]
\[ a_{id} \]
\[ a_{id} \]
\[ c_{id} \]
\[ c_{id} \]
\[ e_{val} \]
\[ e_{val} \]
Finding query rewritings

Idea:
Compute covers of the query nodes with the view nodes.

No rewriting
Finding query rewritings

Idea:
Compute covers of the query nodes with the view nodes.
Finding query rewritings

Idea:
Compute covers of the query nodes with the view nodes.

No rewriting
Set enumeration algorithms

SE (Subset Enumeration)

- For each new subset, check if a rewriting can be found
- Test minimality at the end
Set enumeration algorithms

SE (Subset Enumeration)
- For each new subset, check if a rewriting can be found
- Test minimality at the end

ISE (Increasing Subset Enumeration)
- Like SE but enumerates sets from the smallest to the largest
- Finds minimal rewritings first
Set enumeration algorithms

SE (Subset Enumeration)
- For each new subset, check if a rewriting can be found
- Test minimality at the end

ISE (Increasing Subset Enumeration)
- Like SE but enumerates sets from the smallest to the largest
- Finds minimal rewritings first

Inefficiency of SE and ISE
SE and ISE try all possible subsets and repeat work
Bottom-up algorithms

Use smaller partial rewritings to build bigger ones
Bottom-up algorithms

Use smaller partial rewritings to build bigger ones

DPR (Dynamic Programming Rewriting)
  - Dynamic programming style
Bottom-up algorithms

Use smaller partial rewritings to build bigger ones

DPR (Dynamic Programming Rewriting)
  - Dynamic programming style

DFR (Depth First Rewriting)
  - Greedy based on the biggest query coverage
Bottom-up rewriting

$q$

$a$

$b$

$d_{[val=5]}$

$c_{cont}$

$e_{val}$

$v_1$

$v_2$

$v_3$

$a_{id}$

$b_{id,cont}$

$c_{id,cont}$

$d_{val}$

Materialized views for P2P XML warehousing
Bottom-up rewriting

\[ q \]

\[ a \]

\[ b \]

\[ d_{val=5} \]

\[ c_{cont} \]

\[ e_{val} \]

\[ nav_{b.cont.e.val} \]

\[ v_1 \]

\[ v_2 \]

\[ v_3 \]

\[ a_{id} \]

\[ b_{id.cont} \]

\[ c_{id.cont} \]

\[ d_{val} \]

Materialized views for P2P XML warehousing
Bottom-up rewriting

\[ \sigma_{b.id < c.id} \]
\[ \bowtie_{a.id < b.id} \]
\[ \text{nav}_{b.\text{cont}, e.\text{val}} \]

Materialized views for P2P XML warehousing
Bottom-up rewriting

$q$

\[ a \quad d_{[\text{val}=5]} \quad b \quad c_{\text{cont}} \quad e_{\text{val}} \]

\[ v_1 \quad v_2 \quad v_3 \]

\[ b_{\text{id}, \text{cont}} \quad c_{\text{id}, \text{cont}} \quad d_{\text{val}} \]

\[ a_{\text{id}} \quad a_{\text{id}} \]

Materialized views for P2P XML warehousing
Bottom-up rewriting

$q$

$\pi_{\text{c.cont}, \text{e.val}}$

$\bowtie \sigma_{\text{val}=5}$

$\sigma_{\text{b.id} < \text{c.id}}$

$\bowtie \sigma_{\text{a.id} < \text{b.id}}$

$\text{nav}_{\text{b.cont}, \text{e.val}}$

Materialized views for P2P XML warehousing
Bottom-up rewriting

\[
q = \pi_{c.cont, e.val} \sigma_{a.id = a.id \land b.id \prec c.id \land a.id \prec b.id \land d.val = 5} \times \text{nav}_{b.cont, e.val} v_2 v_3 \times v_1
\]
SE, ISE, DPR and DFR are correct and complete. They produce all minimal canonical rewritings of $q$ given $\mathcal{V}$.

- Which rewritings are ”good”?
  - The one which leads to the best physical plan
  - We learn this too late!

- Heuristic: a good rewriting uses the smallest number of views.
  - DFR typically finds fast a solution which is reasonably good
  - ISE, DPR will need more time but return better quality results. They produce rewritings towards the end of the search
View indexing and lookup for query rewriting

Query $q$ asked at peer $p \Rightarrow p$ needs to find useful views

![Diagram](image)

<table>
<thead>
<tr>
<th>LI index keys</th>
<th>LI &amp; RLI index keys</th>
<th>RLI index keys</th>
<th>LPI index keys</th>
<th>LPI &amp; RPI lookup keys</th>
<th>RPI index keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a, b$</td>
<td>$a, b$</td>
<td>$b, c$</td>
<td>$a/b/c, a/b/d, a/e$</td>
<td>$a/b, a/c$</td>
<td>$a/b$</td>
</tr>
<tr>
<td>$c, d$</td>
<td>$c, d$</td>
<td>$b/c, b/d$</td>
<td></td>
<td>$a/d, a/e$</td>
<td>$a/b/c$</td>
</tr>
<tr>
<td>$e$</td>
<td>$e$</td>
<td>$a/b/c, a/b/d$</td>
<td></td>
<td>$b/c, b/d$</td>
<td></td>
</tr>
</tbody>
</table>

Materialized views for P2P XML warehousing
ViP2P platform

- Fully implemented using Java 6 (294 classes, 60,000 lines of code)
- Used Berkeley DB (version 3.3.75) to store view data
- Used FreePastry (version 2.1) as our DHT network
- Experiments carried on Grid5000 using 250 machines
- 1000 ViP2P peers were deployed
We used 1440 views related to but different from query \( q \)
We used 1440 views related to but different from query $q$. 
View building

Indexed 2000 XMark documents and 500 views (70 related to the documents)
Performance of rewriting algorithms
Query execution

```
project@
bordereau-25.bordeaux.grid5000.fr

receive@
bordereau-25.bordeaux.grid5000.fr

hashJoin@
pastel-79.toulouse.grid5000.fr

scan(4nodesView)@
pastel-79.toulouse.grid5000.fr

receive@
pastel-79.toulouse.grid5000.fr

scan(3nodesView)@
griffon-92.nancy.grid5000.fr
```
Query execution

- Total execution time
- Time to first result

Number of results vs. Time (seconds)
XML on DHTs [GWJD03, BC06, SHA05, AMP+08]
- We have the most generic approach

XPath query rewriting [BOB+04, XO05, CDO08, TYÖ+08]
- XPath: wildcard *, union
- Rewritings: intersection, navigations, joins
- They don’t have multiple returning nodes

Rewriting with structural constrains [ABMP07]
- Centralized setting
- Dataguide [GW97] constraints
ViP2P: data access support structures for DHT based XML data management

All the presented algorithms have been fully implemented in a functional Java based platform

Presented at DataX 2009 (no proceedings)

Extended version submitted for publication

Visit us at vip2p.saclay.inria.fr!
Thank you!


<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>Conference</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GWJD03]</td>
<td>L. Galanis, Y. Wang, S.R. Jeffery, and D.J. DeWitt</td>
<td>Locating Data Sources in Large Distributed Systems.</td>
<td>VLDB, 2003</td>
</tr>
</tbody>
</table>
In *OTM Conferences (2)*, 2005.

Multiple Materialized View Selection for XPath Query Rewriting.  

[XO05] W. Xu and M. Ozsoyoglu.  
Rewriting XPath Queries Using Materialized Views.  
In *VLDB*, 2005.