Materialized views for P2P XML warehousing

Ioana Manolescu¹  Spyros Zoupanos¹

¹ GEMO group, INRIA Saclay – Île-de-France, France

3 April 2009
Materialized views for P2P XML warehousing
What is ViP2P (Views in Peer-to-Peer) ?

A fully deployed system that permits us to:

- Declare tree pattern XML views
- Fill in the views with XML data
- Reply to tree pattern queries using the existing views
  - View definition lookup
  - Query rewriting
  - Production of a 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:

Materialized views for P2P XML warehousing
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 $\forall v_i(d) \neq \emptyset$
- compute $v_i(d)$
- send results
When \( d \) arrives:

- Search view definitions for which \( v_i(d) \neq \emptyset \)
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
Tree pattern dialect $P$

Representing our queries and views.

- Each pattern node carries a label (element name or attribute name or word).
- All pattern edges correspond to ancestor-descendant relationships between nodes.
- A node may be annotated with zero or more among the following labels: $id$, $cont$ and $val$.
- A node may be annotated with a predicate of the form $[val = c]$ where $c \in A_w$. 
Pattern equivalence

\( p(d) \) is the set of tuples obtained by lining together in a tuple, all \( ids \) and/or \( val \) and/or serialized \( cont \), for each embedding of \( p \) in \( d \).

Two patterns \( p_1, p_2 \) are equivalent, denoted \( p_1 \equiv p_2 \), if for any database \( D \), \( p_1(D) = p_2(D) \).
Let $q \in \mathcal{P}$ be a query and $\mathcal{V} = \{v_1, v_2, \ldots, v_k\}$ a set of views, $v_i \in \mathcal{P}, 1 \leq i \leq k$. A **rewriting** of $q$ using $\mathcal{V}$ is an algebraic expression $e(v_1, v_2, \ldots, v_k)$, such that $e \equiv q$.

**Algebra operators:**

- $scan(v)$
- $\pi_{pList}(op)$
- $\sigma_{cond}(op)$ is a **selection** over $op$, where $cond$ is a conjunctive predicate using the comparison operants $=$ and $\prec$
Algebraic rewriting & operators

- $\text{nav}(op, i, np)$ is a **navigation** operator, applying the navigation described by the pattern $np$ over the $i$ attribute of $op$

- $op \bowtie_{pred} op'$ is a **join** operator

Interesting cases:

- equality joins on node $ids$.
- structural joins on node $ids$. 
Rewriting problem statement

Given a set of views \( \mathcal{V} \) and a query \( q \), the **problem of rewriting** \( q \) based on \( \mathcal{V} \) consists of finding all minimal equivalent rewritings of \( q \), up to **algebraic equivalence**.

Two plans \( a_1, a_2 \in \mathcal{A} \) are **algebra-equivalent** if \( a_2 \) can be obtained from \( a_1 \) via:

- usual rewriting rules from the relational algebra (e.g. pushing selections and projections, join re-ordering etc.);
Rewriting problem statement

Given a set of views \( \mathcal{V} \) and a query \( q \), the **problem of rewriting** \( q \) based on \( \mathcal{V} \) consists of finding all minimal equivalent rewritings of \( q \), up to **algebraic equivalence**.

Two plans \( a_1, a_2 \in \mathcal{A} \) are **algebra-equivalent** if \( a_2 \) can be obtained from \( a_1 \) via:

- usual rewriting rules from the relational algebra (e.g. pushing selections and projections, join re-ordering etc.);
- transitive closure of ancestor-descendant predicates;
Rewriting problem statement

Given a set of views $\mathcal{V}$ and a query $q$, the problem of rewriting $q$ based on $\mathcal{V}$ consists of finding all minimal equivalent rewritings of $q$, up to algebraic equivalence.

Two plans $a_1, a_2 \in \mathcal{A}$ are algebra-equivalent if $a_2$ can be obtained from $a_1$ via:

- usual rewriting rules from the relational algebra (e.g. pushing selections and projections, join re-ordering etc.);
- transitive closure of ancestor-descendant predicates;
- or pattern composition.
Rewriting example

Materialized views for P2P XML warehousing
Rewriting example

Materialized views for P2P XML warehousing
Materialized views for P2P XML warehousing
Rewriting example

Materialized views for P2P XML warehousing
Rewriting example

Materialized views for P2P XML warehousing
What data structures to use for rewriting?

We rewrite a tree pattern (target).

We build algebraic plans (tool).

Rewriting manipulates (plan, pattern) pairs

- the plan is always $\equiv$ to the pattern
- initial pattern = $v$, plan = $\text{scan}(v)$
- we build increasingly larger plans and incrementally more complex patterns
- when pattern $\equiv$ query, plan is a solution
Important property

Let $v$ be a view and $q$ be a query. If $v$ can not be embedded in $q$ then no rewriting of $q$ will use $v$.

Applications:

- prune the initial views used for rewriting
- discard intermediary (plan,pattern) pairs which do not lead to complete rewritings
DPR - dynamic programming rewriting algorithm

- Dynamic programming style
- Proceeds in layers
  - build all `ppps` joining `n` views before building a `ppp` of `n + 1` views
- Builds left-deep plans (to ensure uniqueness) up to algebraic equivalence
Second algorithm DFR - depth first rewriting algorithm

DFR organizes and explores differently its ppps.

- Tries to combine the ppp with the **biggest query coverage** with a *ppp* of 1 view.
- Explores left deep plans, like DPR.
**Rewriting algorithms trade-offs**

- What kind of rewritings are "good"?
  - the one which leads to the best physical plan.
  - we learn this too late!
- heuristic: a good rewriting is the one that uses the **smallest** number of views.
  - DFR is going to find fast enough a good solution but not the best.
  - DPR will need more time but returns better quality results.
Performance of rewriting algorithms

- DFR Total
- DPR Total
- DFR First
- DPR First

Rewriting time (ms)

Number of views
Peer $p$ has a view $v$, peer $p_d$ publishes a document $d$.

- $p$ indexes $v$ on the DHT by the labels of the view.
- $p_d$ traverses $d$, looks up all its labels.
- $p_d$ ends up with a superset of answers $S_a$. It evaluates $v(d)$ for each $v \in S_a$.

- Many views can be evaluated in one document traversal.
Indexing and lookup view definitions

When a query $q$ arrives at peer $p$, it has to find useful view definitions for the rewriting algorithm.

4 different strategies have been implemented.

- **Label indexing (LI):**
  - index $v$ by each $v$ node label.
  - look up by all node labels of $q$.

- **Return label indexing (RLI):**
  - index $v$ by the labels of all $v$ nodes which project some attributes (at most $|v|$).
  - same as for LI: use the labels of all $q$ nodes as lookup keys.
Indexing and lookup view definitions

- **Leaf path indexing** (LPI):
  - let $LP(v)$ be the set of all the distinct root-to-leaf label paths of $v$.
  - Index $v$ using each element of $LP(v)$ as key.
  - look up details in the paper.

- **Return Path Indexing** (RPI):
  - let $RP(v)$ be the set of all rooted paths in $v$ which end in a node that returns some attribute.
  - Index $v$ using each element of $LP(v)$ as key.
  - same as for LPI.
System implementation and configuration

- Platform fully implemented using Java 6.
- Used Berkeley DB (version 3.2.76) to store view data.
- Used FreePastry (version 2.1) as our DHT network.
- Experiments carried on a cluster of 10 PCs with Intel Xeon 5140 CPU @ 2.33 GHz and 4GB of Ram.
For the experiments we used 80 peers, indexed 1440 views related to but different from query $q$. 
For the experiments we used 30 peers, indexed 100 XMark [SWK+02] documents and 30 views related to these documents.
Materialized views for P2P XML warehousing
Query execution

(9) SimpleProject
   @195.83.212.160:7000,7001

(5) STD($1 and $1)
   @195.83.212.160:7000,7001

(0) Scan(4nodesView)
   @195.83.212.160:7000,7001

(4) Receive
   @195.83.212.160:7000,7001

(1) Scan(2nodesView)
   @172.20.1.2:7000,7001

![Execution time graph]

Materialized views for P2P XML warehousing
Benefits of ViP2P views

We use a data set of 750 XMark [SWK+02] documents having the total size of 20MB, 2 peers and three different view sets to rewrite the query \textit{site}(item(description_{cont})).
Benefits of ViP2P views

We use a data set of 750 XMark [SWK+02] documents having the total size of 20MB, 2 peers and three different view sets to rewrite the query $\text{site(item(description}_{\text{cont}}))$.

- $\forall_1$ contains the view $\text{site}_{\text{cont}}$. 
Benefits of ViP2P views

We use a data set of 750 XMark [SWK⁺02] documents having the total size of 20MB, 2 peers and three different view sets to rewrite the query \( \text{site}(\text{item}(\text{description}_{\text{cont}})) \).

- \( \nu_1 \) contains the view \( \text{site}_{\text{cont}} \).

- \( \nu_2 \) contains three views: \( \text{site}_{\text{id}}, \text{item}_{\text{id}} \) and \( \text{description}_{\text{id,cont}} \). Node-granularity indexing used in [AMP⁺08] (we also time the transfer of the XML result snippets to the query peer).
Benefits of ViP2P views

We use a data set of 750 XMark [SWK+02] documents having the total size of 20MB, 2 peers and three different view sets to rewrite the query $site(item(description_{cont})).$

- $\mathcal{V}_1$ contains the view $site_{cont}$.
- $\mathcal{V}_2$ contains three views: $site_{id}$, $item_{id}$ and $description_{id,cont}$. Node-granularity indexing used in [AMP+08] (we also time the transfer of the XML result snippets to the query peer).
- $\mathcal{V}_3$ contains one view which is exactly $q$. 
Benefits of ViP2P views

We use a data set of 750 XMark [SWK⁺02] documents having the total size of 20MB, 2 peers and three different view sets to rewrite the query \( \text{site}(\text{item}(\text{description}_{\text{cont}})) \).
Related work

Indexing documents in the DHT
In [GWJD03, BC06, SHA05, AMP+08] the focus is on indexing documents on DHT so that XML queries can be processed fast.

XPath query rewriting
In [BOB+04, XO05] XPath query rewriting has been considered. They focus on handling more XPath axis, operators such as union etc. We consider richer views, offer more rewriting opportunities and view management in a DHT network.

Rewriting with structural constrains
[ABMP07] is a centralized system where they used structural constraints encapsulated in a Dataguide [GW97] to perform rewriting.
Summing up

• Efficient management of large XML warehouses in structured P2P networks requires the ability to deploy data access support structures, which can be tuned closely to fit application needs.

• ViP2P offers the ability to build and maintain complex materialized views.

• 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!
Materialized views for P2P XML warehousing


<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>Details</th>
</tr>
</thead>
</table>
W. Xu and M. Ozsoyoglu.
Rewriting XPath queries using materialized views.
In *VLDB*, 2005.

[RO05]