Materialized views for P2P XML warehousing

Ioana Manolescu[1] **Spyros Zoupanos**[1]

[1] GEMO/IASI group, INRIA Saclay – Île-de-France

21 October 2009
BDA 2009, Namur

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche SACLAY - ÎLE-DE-FRANCE

gemo

Problem statement and contributions

Problem

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

Problem statement and contributions

Problem

efficient and flexible XML content sharing in peet-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

Outline

1. **Introduction**
2. **Algebraic query rewriting**
   - Tree patterns for views and queries
   - Algebraic rewriting & operators
3. **Rewriting algorithms**
4. **View management**
   - View definitions index/lookup
5. **Experiments**
6. **Conclusion**

XML materialized views on a DHT

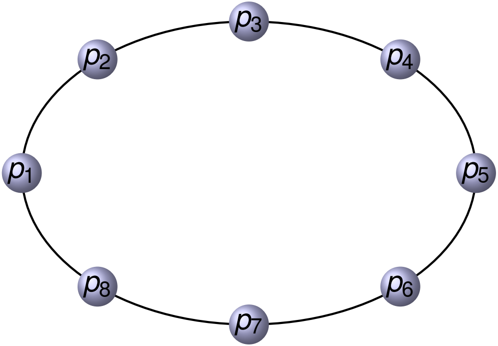
Declare tree pattern XML views over the network data

Fill in the views with XML data

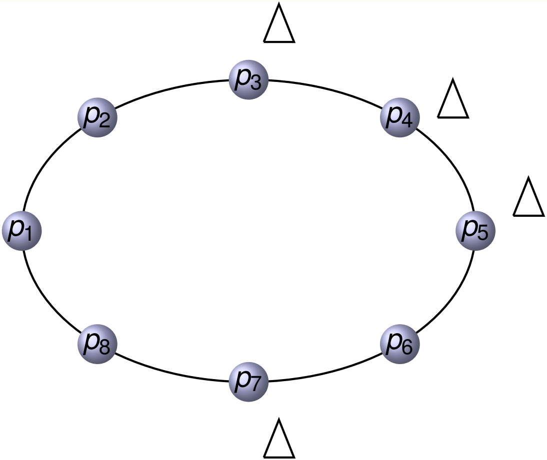
Answer tree pattern queries using the existing views

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

Architecture overview



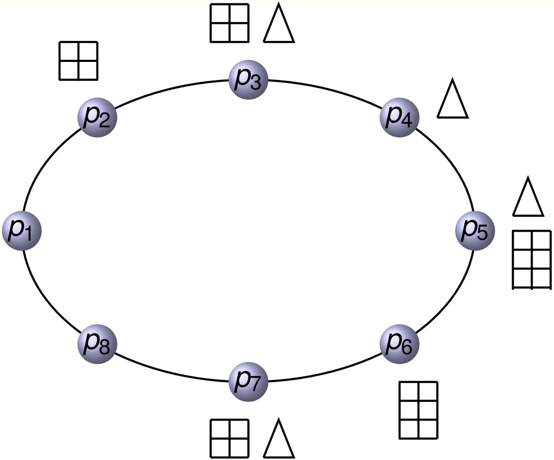
Architecture overview



The peers may store:

- documents

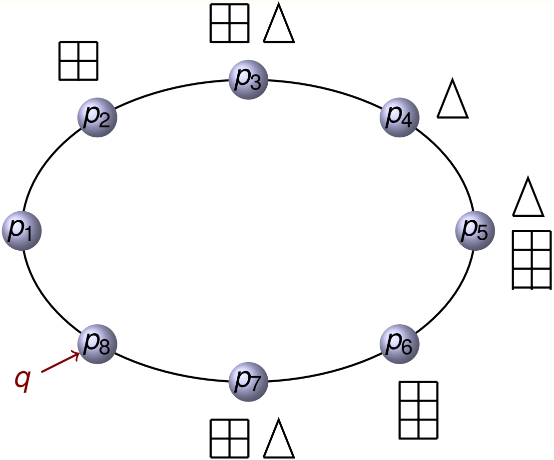
Architecture overview



The peers may store:

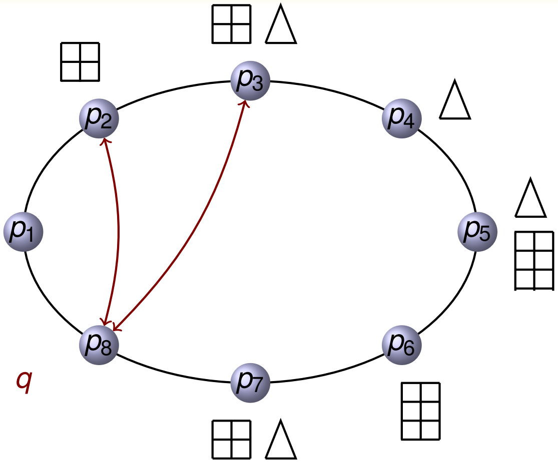
- documents
- views

Architecture overview



When q arrives:

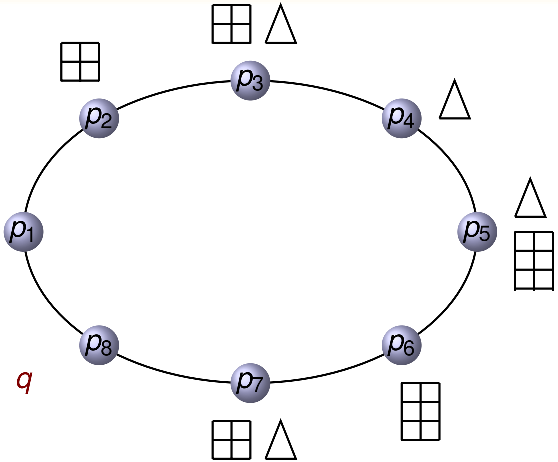
Architecture overview



When q arrives:

- view definition lookup

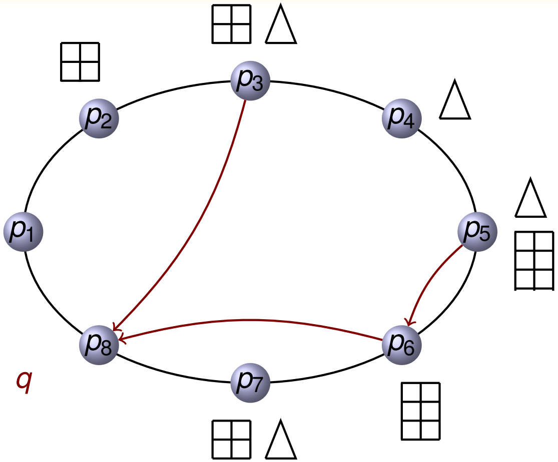
Architecture overview



When q arrives:

- view definition lookup
- rewriting

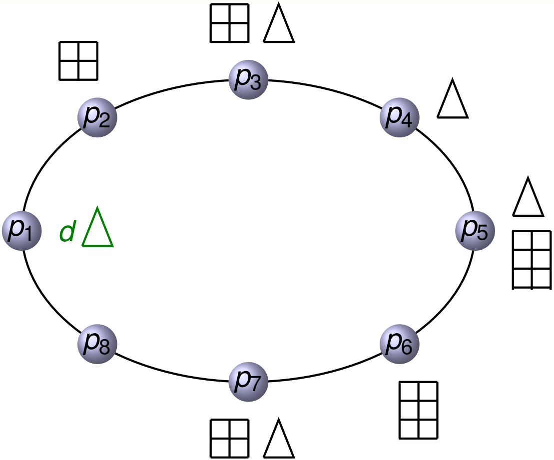
Architecture overview



When *q* arrives:

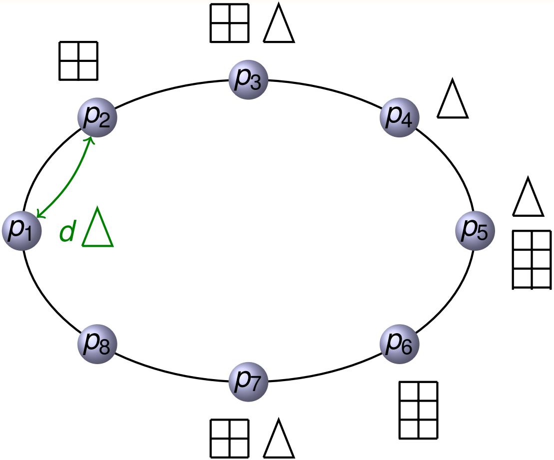
- view definition lookup
- rewriting
- execution of physical plan

Architecture overview



When d arrives:

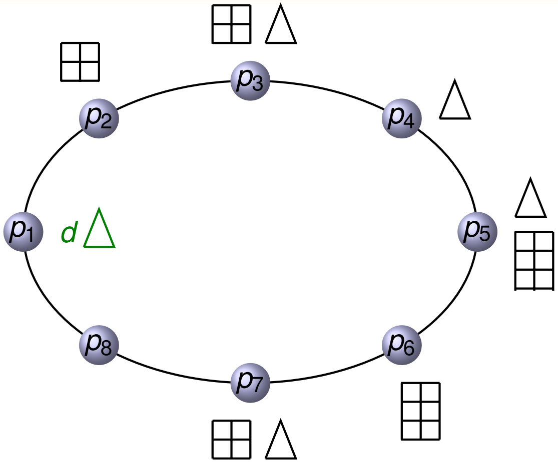
Architecture overview



When d arrives:

- search view definitions for which $v_i(d) \neq \emptyset$

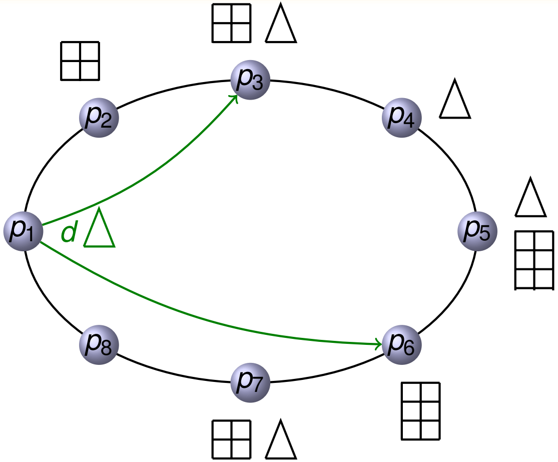
Architecture overview



When d arrives:

- search view definitions for which $v_i(d) \neq \emptyset$
- compute $v_i(d)$

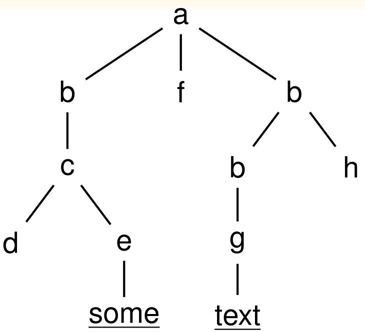
Architecture overview



When d arrives:

- search view definitions for which $v_i(d) \neq \emptyset$
- compute $v_i(d)$
- send results

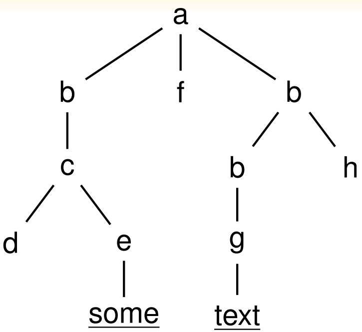
Tree patterns for views and queries



b_{cont}

b_{cont}

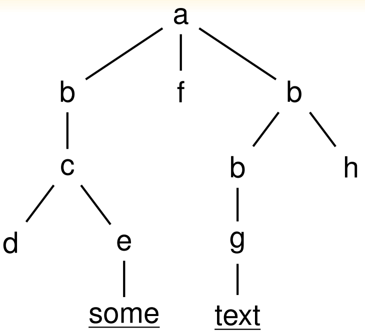
Tree patterns for views and queries



b_{cont}

b_{cont}
<c><d / ><e>some< / e>< / c>< / b>

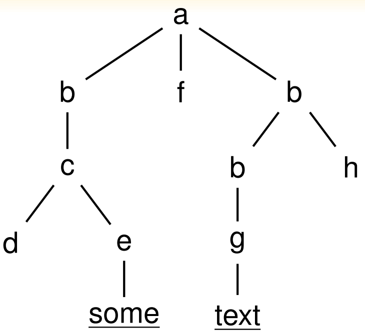
Tree patterns for views and queries



b_{cont}

b_{cont}
<c><d/><e>some</e></c>
<g>text</g><h/>

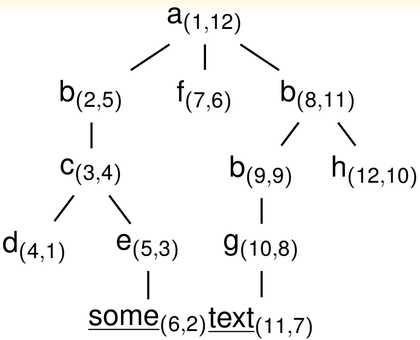
Tree patterns for views and queries



b_{cont}

b_{cont}
<c><d/><e>some</e></c>
<g>text</g><h/>
<g>text</g>

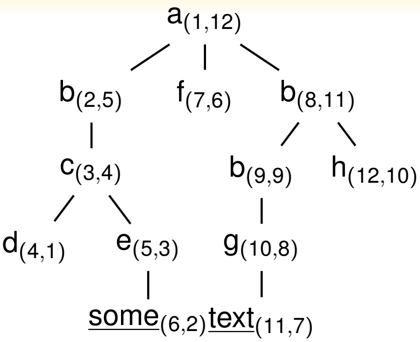
Tree patterns for views and queries



$b_{id,cont}$

b_{id}	b_{cont}

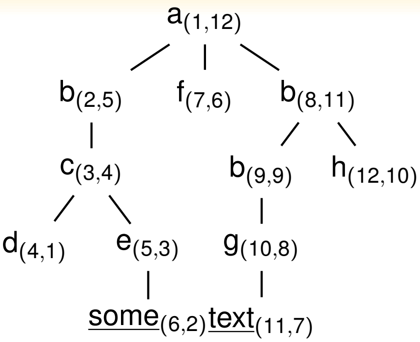
Tree patterns for views and queries



$b_{id,cont}$

b_{id}	b_{cont}
(2,5)	<c><d/><e>some</e></c>

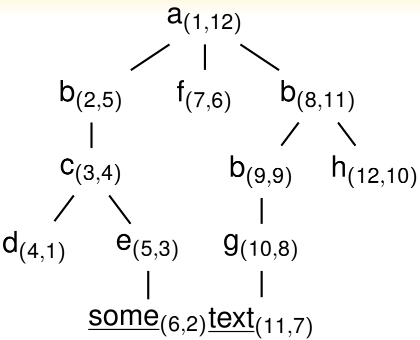
Tree patterns for views and queries



$b_{id,cont}$

b_{id}	b_{cont}
(2,5)	<c><d/><e>some</e></c>
(8,11)	<g>text</g><h/>

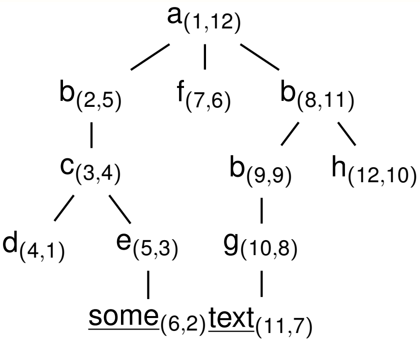
Tree patterns for views and queries



$b_{id,cont}$

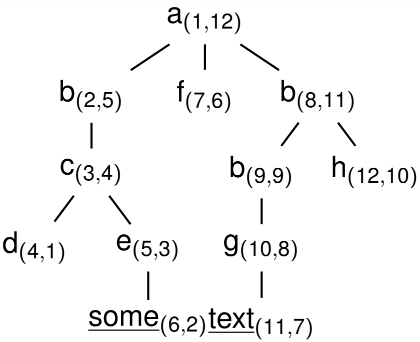
b_{id}	b_{cont}
(2,5)	<c><d/><e>some</e></c>
(8,11)	<g>text</g><h/>
(9,9)	<g>text</g>

Tree patterns for views and queries



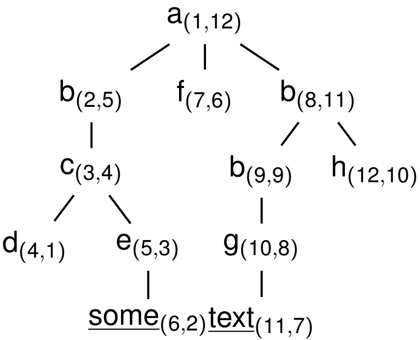
a_{id}	b_{val}

Tree patterns for views and queries



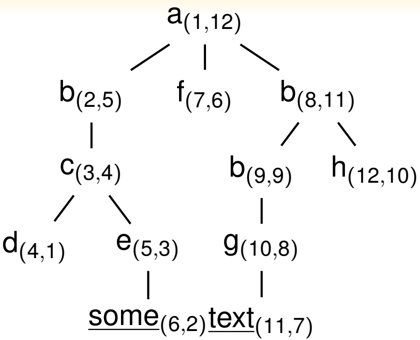
a_{id}	b_{val}
(1,12)	some

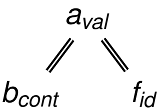
Tree patterns for views and queries



a_{id}	b_{val}
(1,12)	some
(1,12)	text

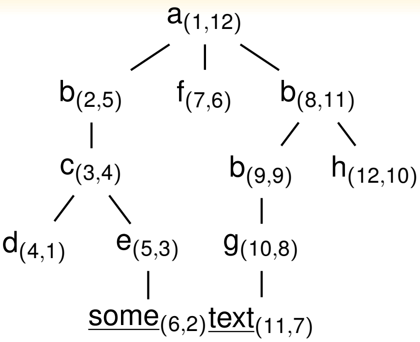
Tree patterns for views and queries

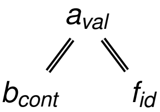




a_{val}	b_{cont}	f_{id}

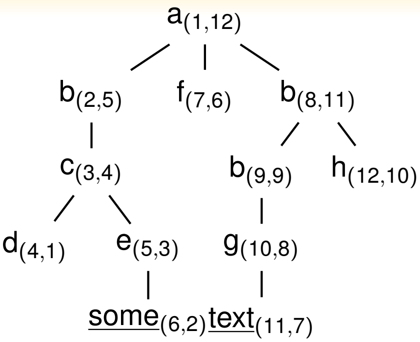
Tree patterns for views and queries

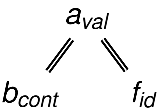




a_{val}	b_{cont}	f_{id}
some text	<c><d / ><e>some< / e>< / c>< / b>	(7,6)

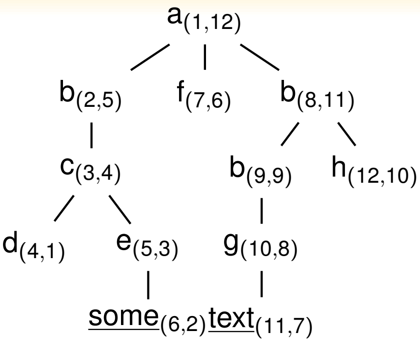
Tree patterns for views and queries

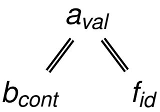




a_{val}	b_{cont}	f_{id}
some text	<c><d/><e>some</e></c>	(7,6)
some text	<g>text</g><h/>	(7,6)

Tree patterns for views and queries





a_{val}	b_{cont}	f_{id}
some text	<c><d/><e>some</e></c>	(7,6)
some text	<g>text</g><h/>	(7,6)
some text	<g>text</g>	(7,6)

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

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

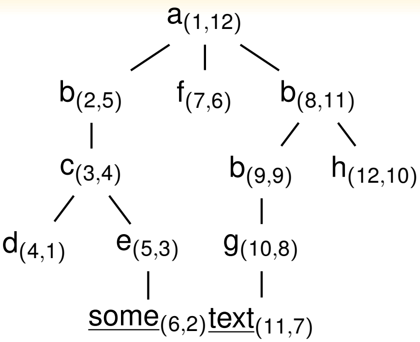
such that $e(\mathcal{D}) = q(\mathcal{D})$ for any document set $\mathcal{D}$

Algebra operators

$scan(v)$	$\times$
$\pi_{cols}(op)$	$\pi^{o}(op)$
$sort_{cols}(op)$	$\sigma_{cond}(op)$

$nav_{i,np}(op)$ evaluates *np* over the *cont* attribute *op.i*

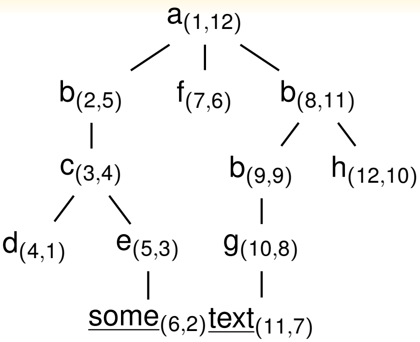
Navigation example



b_{cont}

b_{cont}	$nav_{b.cont,g_{val}}(b_{cont})$

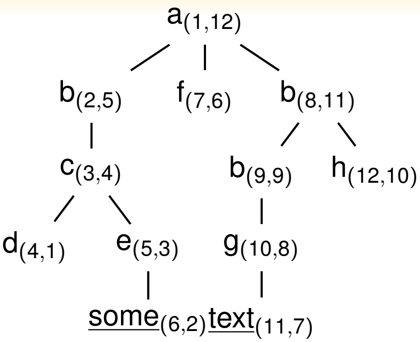
Navigation example



b_{cont}

b_{cont}	$nav_{b.cont,g_{val}}(b_{cont})$
<c><d/><e>some</e></c>	-

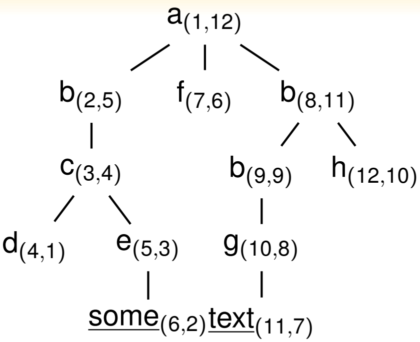
Navigation example



b_{cont}

b_{cont}	$nav_{b.cont,g_{val}}(b_{cont})$
<c><d/><e>some</e></c>	-
<g>text</g><h/>	text

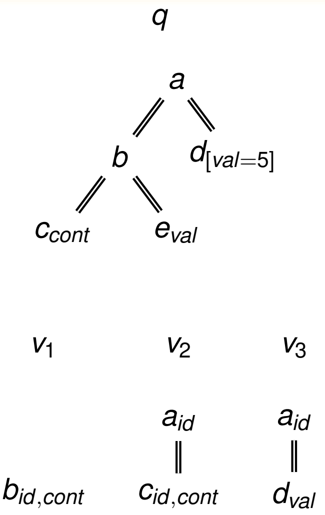
Navigation example



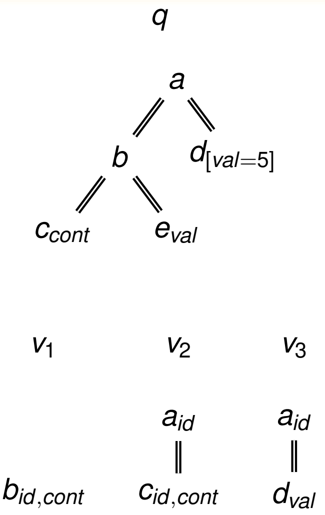
b_{cont}

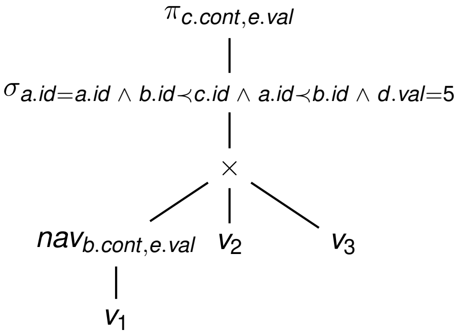
b_{cont}	$nav_{b.cont,g_{val}}(b_{cont})$
<c><d/><e>some</e></c>	-
<g>text</g><h/>	text
<g>text</g>	text

Rewriting example



Rewriting example





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.

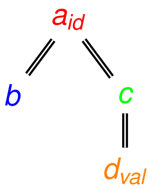
q	v_1	v_2
a_{id}		
$\Vert$		
b_{id}	a_{id}	b_{id}

Finding query rewritings

Idea:

Compute covers of the query nodes with the view nodes.

q



v

$a_{id,cont}$

Finding query rewritings

Idea:

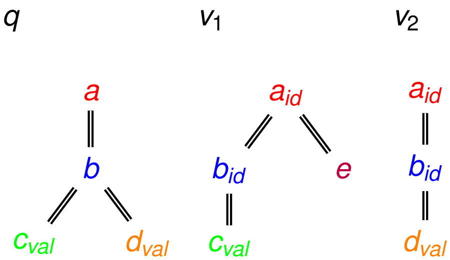
Compute covers of the query nodes with the view nodes.

q $\qquad\qquad\qquad\qquad$ v

Finding query rewritings

Idea:

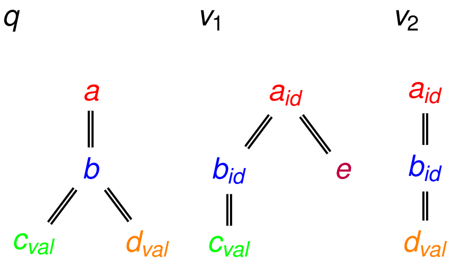
Compute covers of the query nodes with the view nodes.



Finding query rewritings

Idea:

Compute covers of the query nodes with the view nodes.

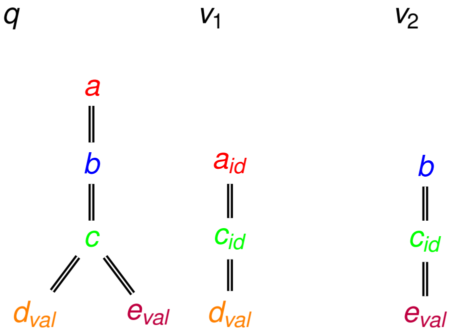


No rewriting

Finding query rewritings

Idea:

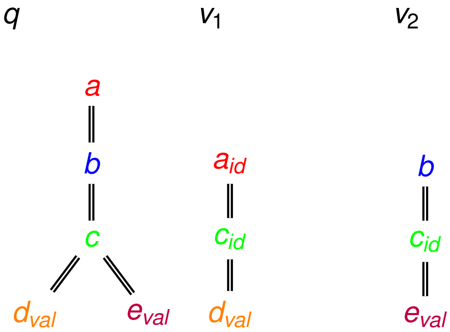
Compute covers of the query nodes with the view nodes.



Finding query rewritings

Idea:

Compute covers of the query nodes with the view nodes.

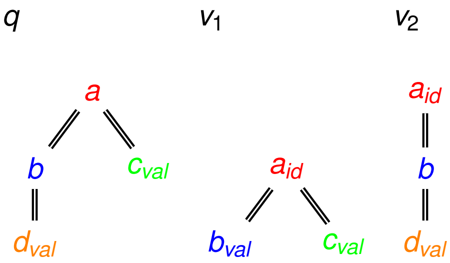


No rewriting

Finding query rewritings

Idea:

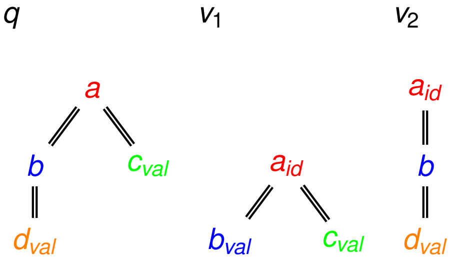
Compute covers of the query nodes with the view nodes.



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

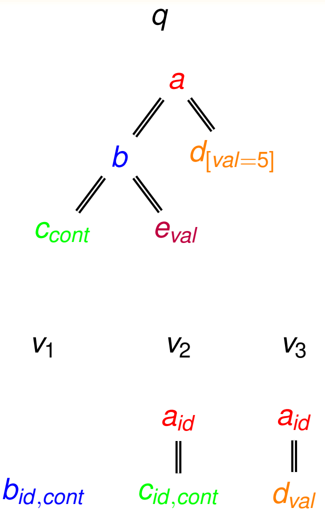
DPR (Dynamic Programming Rewriting)

- Dynamic programming style

DFR (Depth First Rewriting)

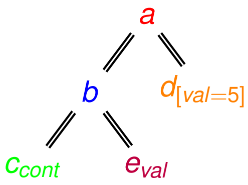
- Greedy based on the biggest query coverage

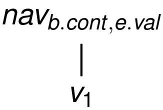
Bottom-up rewriting

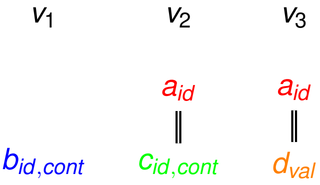


Bottom-up rewriting

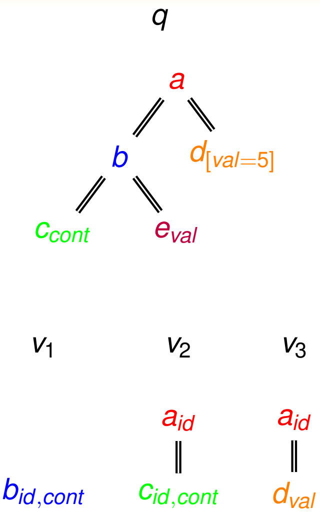
q

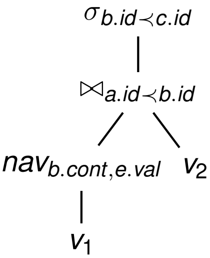




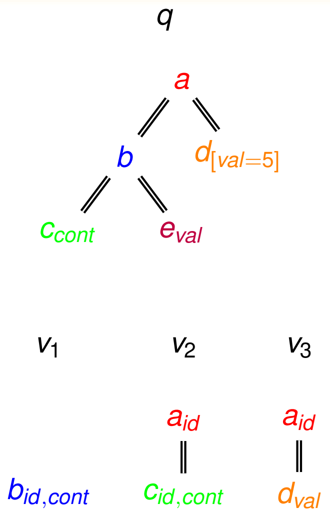


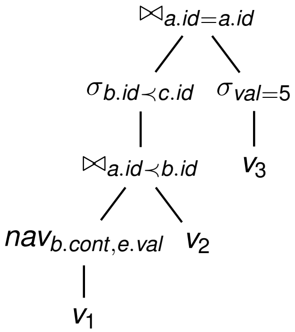
Bottom-up rewriting



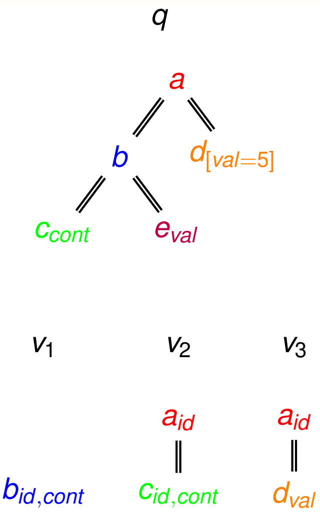


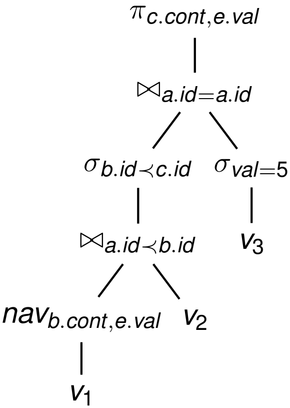
Bottom-up rewriting



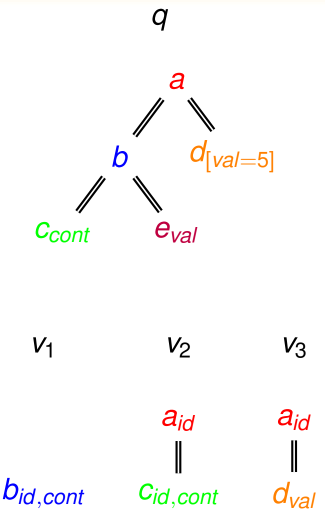


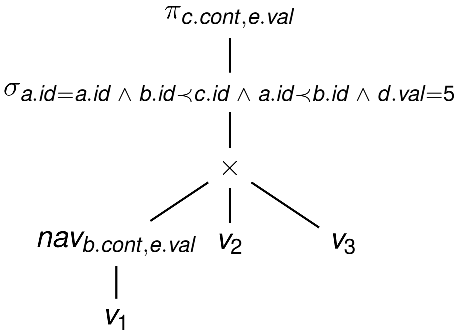
Bottom-up rewriting





Bottom-up rewriting





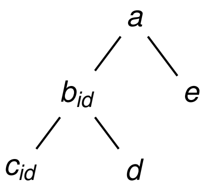
Rewriting algorithms trade-offs

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



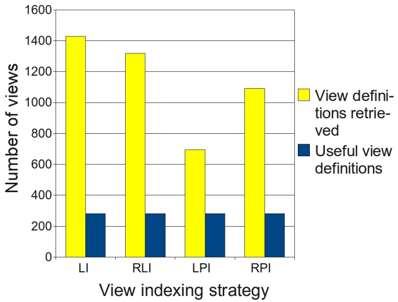
LI index keys	**LI** & **RLI** lookup keys	**RLI** index keys	**LPI** index keys	**LPI** & **RPI** lookup keys	**RPI** index keys
a, b c, d e	a, b c, d e	b, c	$a/b/c$, $a/b/d$, a/e	a/b, a/c a/d, a/e b/c, b/d $a/b/c$, $a/b/d$	a/b $a/b/c$

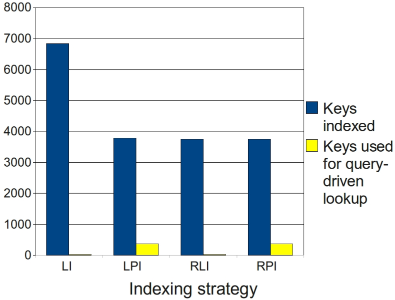
ViP2P platform

- 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

View look up performance

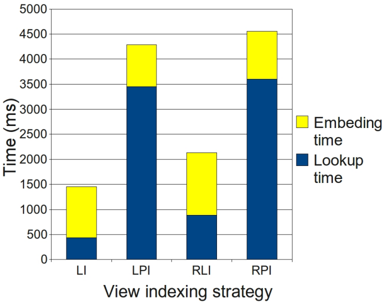
We used 1440 views related to but different from query *q*





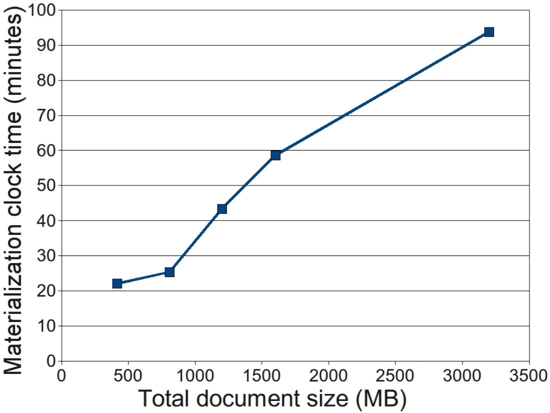
View look up performance

We used 1440 views related to but different from query *q*

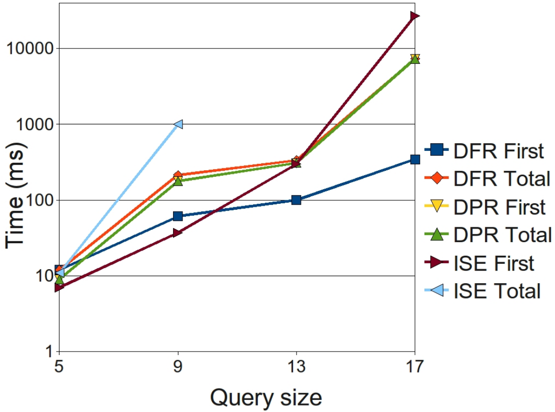


View building

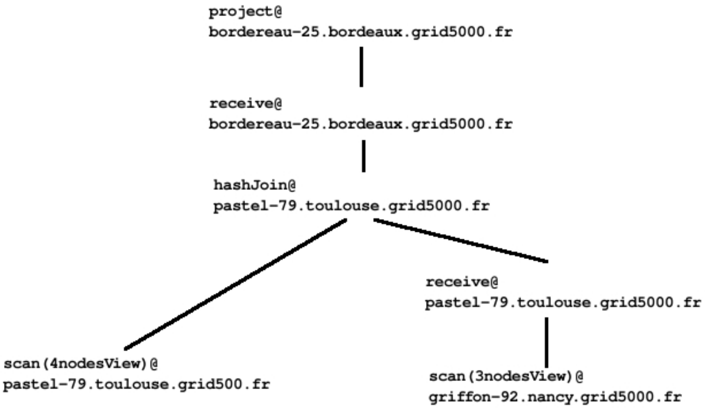
Indexed 2000 XMark documents and 500 views (70 related to the documents)



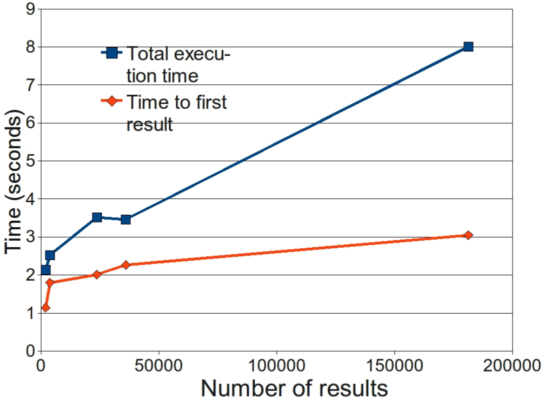
Performance of rewriting algorithms



Query execution



Query execution



Related work

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

Summing up

- 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!

[ABMP07] **Andrei Arion, Véronique Benzaken, Ioana Manolescu, and Yannis Papakonstantinou.**
Structured Materialized Views for XML Queries.
In *VLDB*, pages 87–98, 2007.

[AMP+08] **S. Abiteboul, I. Manolescu, N. Polyzotis, N. Preda, and C. Sun.**
XML processing in DHT networks.
In *ICDE*, 2008.

[BC06] **Angela Bonifati and Alfredo Cuzzocrea.**
Storing and retrieving XPath fragments in structured P2P networks.
Data Knowl. Eng., 59(2), 2006.

[BOB+04] **A. Balmin, F. Ozcan, K. Beyer, R. Cochrane, and H. Pirahesh.**
A Framework for Using Materialized XPath Views in XML Query Processing.
In *VLDB*, 2004.

[CDO08] **Bogdan Cautis, Alin Deutsch, and Nicola Onose.**
XPath Rewriting Using Multiple Views: Achieving Completeness and Efficiency.
In *WebDB*, 2008.

[GW97] **Roy Goldman and Jennifer Widom.**
DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases.
In *VLDB*, 1997.

[GWJD03] **L. Galanis, Y. Wang, S.R. Jeffery, and D.J. DeWitt.**
Locating Data Sources in Large Distributed Systems.
In *VLDB*, 2003.

[SHA05] **Gleb Skobeltsyn, Manfred Hauswirth, and Karl Aberer.**
Efficient Processing of XPath Queries with Structured Overlay Networks.

In *OTM Conferences (2)*, 2005.

[TYÖ+08] **Nan Tang, Jeffrey Xu Yu, M. Tamer Özsu, Byron Choi, and Kam-Fai Wong.**
Multiple Materialized View Selection for XPath Query Rewriting.
In *ICDE*, pages 873–882, 2008.

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