Web data management on DHTs

Ioana Manolescu

GEMO/IASI group, INRIA Saclay – Île-de-France & LRI

http://www-rocq.inria.fr/~manolesc

August 24, 2009
XSym workshop

Work supported by the ANR WebContent (2005-2009) and CODEX ANR-08-DEFIS-004 (2009-2012) grants
Outline

1. Introduction
2. KadoP XML indexing
   - Indexing and query processing
   - Scaling up
3. ViP2P: mat. views on DHTs
   - Algebraic query rewriting in ViP2P
   - View materialization
   - View indexing
   - Query rewriting
4. Summary
Motivation

Distributed data management: old goal (1970)
Motivation

Distributed data management: old goal (1970)

- distributed versions of industrial-strength DBMSs
- map/reduce style systems for massively parallel computations
Motivation

Distributed data management: old goal (1970)

- distributed versions of industrial-strength DBMSs
- map/reduce style systems for massively parallel computations

Still missing: the flexible federation

- high independence of the sites: when to be in, what to store
- data distribution transparency
- ... with the usual performance requirements
Motivation: distributed warehouses of Web content

Web content
structured documents, schemas, annotations, concepts, mappings, Web services, inter-document links

Web content warehouse
Distributed database of selected content, whose users may:
- publish resources
- connect (annotate, map, link...) existing resources
- update resources
- enhance resources by combining them

In the style of the RNTL WebContent project (2005-2009)
Distributed hash tables
Distributed hash tables
Distributed hash tables
Distributed hash tables

\texttt{put}(k_1, v_1)
Distributed hash tables

\[
\begin{align*}
\text{put}(k_1, v_1) & \rightarrow p_3 \\
(k_1, v_1) & \rightarrow p_2 \\
& \rightarrow p_4 \\
& \rightarrow p_5 \\
& \rightarrow p_6 \\
& \rightarrow p_7 \\
& \rightarrow p_8 \\
\end{align*}
\]
Distributed hash tables

\[ (k_1, v_1) \]

- \[ p_1 \]
- \[ p_2 \]
- \[ p_3 \]
- \[ p_4 \]
- \[ p_5 \]
- \[ p_6 \]
- \[ p_7 \]
- \[ p_8 \]

\[ \text{put}(k_1, v_1) \]
\[ \text{put}(k_1, v_2) \]
Distributed hash tables

\((k_1, \{v_1, v_2\})\)

\[\text{put}(k_1, v_1)\]

\[\text{put}(k_1, v_2)\]
Distributed hash tables

- put($k_1, v_1$)
- put($k_1, v_2$)
- get($k_1$)

$(k_1, \{v_1, v_2\})$
Distributed hash tables

(k₁, {v₁, v₂})

put(k₁, v₁) → p₁

get(k₁) → p₁

put(k₁, v₂) → p₂

p₁ → p₂ → p₃ → p₄ → p₅ → p₆ → p₇ → p₈
Distributed hash tables

- put($k_1, v_1$)
- put($k_1, v_2$)
- get($k_1$)
- \( (k_1, \{v_1, v_2\}) \)
What distributed hash tables provide

Dynamic peer networks

- each peer is assigned an id $\Rightarrow$ address range
- bound of $\log_2(N)$ hops to route a message to a given address (peer)
- network re-adjustment when peers join or leave: peers’ address spaces stretch and contract
What distributed hash tables provide

Dynamic peer networks

- each peer is assigned an id ⇒ address range
- bound of $\log_2(N)$ hops to route a message to a given address (peer)
- network re-adjustment when peers join or leave: peers’ address spaces stretch and contract

(Key, value) stores = basis for content sharing

- index the resources by keys
- look up resources by keys

Web data management on DHTs
What distributed hash tables provide

Dynamic peer networks

- each peer is assigned an id ⇒ address range
- bound of $\log_2(N)$ hops to route a message to a given address (peer)
- network re-adjustment when peers join or leave: peers’ address spaces stretch and contract

(Key, value) stores = basis for content sharing

- index the resources by keys
- look up resources by keys

DHTs provide location transparency
From DHTs to distributed data management

Functionalities to add:

- data indexing algorithms
- **storage** for application data and even DHT index data
- local query processing
- distributed query processing: operators, including data transfers, optimization . . .

Reliability provided:

- some peer will always answer at a given address, possible after some time
- some (key, value) replication to handle peer failures (broadcast to $k$ replicas)
From DHTs to distributed data management

Functionalities to add:

- data indexing algorithms
- storage for application data and even DHT index data
- local query processing
- distributed query processing: operators, including data transfers, optimization . . .

Reliability provided:

- some peer will always answer at a given address, possible after some time
- some (key, value) replication to handle peer failures (broadcast to $k$ replicas)

Resilience to other loss of data or functionality needs to be implemented
DHT index queries

Query processing involves:

- operations on the DHT or (key, value) store
- other operations
  - evaluating sub-queries to extract partial results
  - combining several partial results
  - transferring results to query peer

Index query

The part of a user query that can be answered directly by consulting the DHT content index

Typically less precise than the user query

- Find the IDs of documents matching the query
- Find the IDs of documents which may match for the query
Trade-offs in DHT indexing and query processing

Level of detail of the indexing algorithm:
- index query precision $\uparrow \Rightarrow$ execution time $\downarrow$
- data publication time $\uparrow$, possibly execution time $\uparrow$

Data re-placement or clustering:
- fewer peers contacted for a query (message no. $\downarrow$, execution time ?)
- data transfers in the absence of queries (message no. $\uparrow$, total message size $\uparrow$)
Our experience building Web data stores on DHTs

Web data:

- standalone XML documents
- RDF data, RDF schemas, mappings
- annotations on XML fragments
- interconnected XML documents
Our experience building Web data stores on DHTs

Web data:
- standalone XML documents
- RDF data, RDF schemas, mappings
- annotations on XML fragments
- interconnected XML documents

Choices:
Our experience building Web data stores on DHTs

Web data:

- standalone XML documents
- RDF data, RDF schemas, mappings
- annotations on XML fragments
- interconnected XML documents

Choices:

- peers retain control over the data they store/publish
  - no global schema
  - documents published independently
  - annotations, triples, links can freely connect content
- peers collaborate (selflessly) for storing and exploring the index
- load balancing
Outline

1. Introduction

2. KadoP XML indexing
   - Indexing and query processing
   - Scaling up

3. ViP2P: mat. views on DHTs
   - Algebraic query rewriting in ViP2P
   - View materialization
   - View indexing
   - Query rewriting

4. Summary
KadoP: DHT-based XML indexing

Joint work with:
Serge Abiteboul, Nicoleta Preda, Gabriel Vasile, Mohamed Ouazara (INRIA Gemo)
Neoklis Polyzotis, Chong Sun (UCSC) [AMP+08]

<table>
<thead>
<tr>
<th>Content</th>
<th>XML documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries</td>
<td>Conjunctive tree patterns (including keywords)</td>
</tr>
<tr>
<td>Index</td>
<td>Structural IDs of all nodes \textit{and words}</td>
</tr>
</tbody>
</table>
KadoP: DHT-based XML indexing
**KadoP: DHT-based XML indexing**

```
<article>
    <title>XML</title>
    </article>
```

```
doc1.xml
```
KadoP: DHT-based XML indexing

```
<doc1.xml
  <article>
    <title>XML</title>
  </article>
```

Web data management on DHTs
**KadoP: DHT-based XML indexing**

- **doc1.xml**
  - `<article>`
  - `<title>XML</title>`
  - `</article>`

- Graph:
  - (article,(doc1,1,3))
  - (title,(doc1,2,2))
  - (’XML’,(doc1,3,1))

Web data management on DHTs
KadoP: DHT-based XML indexing

**doc2.xml**

```xml
<article>
  <title>Web</title>
</article>
```

**doc1.xml**

```xml
<article>
  <title>XML</title>
</article>
```
**KadoP: DHT-based XML indexing**

**doc1.xml**

```
<article>
  <title>XML</title>
</article>
```

**doc2.xml**

```
<article>
  <title>Web</title>
</article>
```

**Diagram:**
- Node `p1` connected to `p8` with edge `(article,(doc1,1,3))`
- Node `p2` connected to `p3` with edge `(article,(doc1,1,3))`
- Node `p2` connected to `p4` with edge `(article,(doc2,1,3))`
- Node `p3` connected to `p4` with edge `(title,(doc1,2,2))`
- Node `p4` connected to `p5` with edge `(title,(doc2,2,2))`
- Node `p5` connected to `p6` with edge `(‘Web’,(doc2,3,1))`
- Node `p6` connected to `p7` with edge `(‘XML’,(doc1,3,1))`
- Node `p7` connected to `p8` with edge `(‘XML’,(doc1,3,1))`
KadoP: DHT-based XML indexing

**doc₂.xml**

```xml
<article>
  <title>Web</title>
</article>
```

**doc₁.xml**

```xml
<article>
  <title>XML</title>
</article>
```

Web data management on DHTs
KadoP: DHT-based XML indexing

```
<table>
<thead>
<tr>
<th>doc2.xml</th>
</tr>
</thead>
</table>
| `<article>`
|    `<title>Web</title>` |
| `</article>` |

//article[cont(.,’XML’)]/title

<table>
<thead>
<tr>
<th>doc1.xml</th>
</tr>
</thead>
</table>
| `<article>`
|    `<title>XML</title>` |
| `</article>` |

(article,(doc1,1,3))
(article,(doc2,1,3))
(title,(doc1,2,2))
(title,(doc2,2,2))
('Web',(doc2,3,1))
('XML',(doc1,3,1))
//article[cont(.,’XML’)]/title

Web data management on DHTs
**KadoP: DHT-based XML indexing**

**doc2.xml**
<pre>
<article>
  <title>Web</title>
</article>
</pre>

**//article[cont(.,’XML’)]//title**

**doc1.xml**
<pre>
<article>
  <title>XML</title>
</article>
</pre>

($(article,(doc1,1,3))$
$(article,(doc2,1,3))$
$(title,(doc1,2,2))$
$(title,(doc2,2,2))$
$(’Web’,(doc2,3,1))$
$(’XML’,(doc1,3,1))$)
KadoP: DHT-based XML indexing

```
//article[cont(.,'XML')]/title
(doc1,2,2)
```

```
doc2.xml
<article>
  <title>Web</title>
</article>
```

```
doc1.xml
<article>
  <title>XML</title>
</article>
```

```
(article,(doc1,1,3))
(article,(doc2,1,3))
(title,(doc1,2,2))
(title,(doc2,2,2))
('XML',(doc1,3,1))
('Web',(doc2,3,1))
```
KadoP: DHT-based XML indexing

**doc2.xml**

```xml
<article>
  <title>Web</title>
</article>
```

//article[cont(.,’XML’)]//title
(doc1,2,2)

**doc1.xml**

```xml
<article>
  <title>XML</title>
</article>
```

(’XML’,(doc1,3,1))

(’Web’,(doc2,3,1))

(Title,(doc1,2,2))

(Title,(doc2,2,2))

(article,(doc1,1,3))

(article,(doc2,1,3))

Web data management on DHTs
KadoP: DHT-based XML indexing

```
<table>
<thead>
<tr>
<th>doc2.xml</th>
</tr>
</thead>
</table>
| <article>
|   <title>Web</title>
| </article>
|          |
```

```
//article[cont(.,’XML’)]//title
(doc1,2,2)
<t|tile>XML</title>
```

```
<table>
<thead>
<tr>
<th>doc1.xml</th>
</tr>
</thead>
</table>
| <article>
|   <title>XML</title>
| </article>
|          |
```

```
(article,(doc1,1,3))
(article,(doc2,1,3))
(title,(doc1,2,2))
(title,(doc2,2,2))
('Web',(doc2,3,1))
('XML',(doc1,3,1))
```

Web data management on DHTs
Scaling up KadoP

Engineering issues:

1. DHT values were (potentially large) posting lists; DHT store would not cope
   - simplistic XMLized storage
   - gzip compression
   - re-implemented

2. Blocking \textit{get} operation; implemented block-based, pipelined method

Scalability issue: longest posting list involved in a query is the bottleneck

- long posting list = frequent term; known problem [LHSH04]
- organized posting lists in distributed B-tree style \(\Rightarrow\) parallelized posting list transfers
Scaling up KadoP

Scalability issue:

To compute \( //a//b \), KadoP transfers the complete posting lists of \( //a \) and \( //b \).

Bloom Filters for KadoP

- Semijoin-like idea
- Posting lists are ordered \( \Rightarrow \) compact representation of the interval covered by \( //a \) in the Bloom Filter of \( //a \)
- **Reduce** the transferred list \( //b \) by the Bloom Filter of \( //a \)
- Similar ancestor reduction
KadoP indexing experiments on Grid5K

![Diagram showing total publishing time vs size of published data for different scenarios of publishers and peers.]
KadoP querying experiments on Grid5K
Lessons learned with KadoP

- Performant message routing (redundant *fingers*)
Lessons learned with KadoP

- Performant message routing (redundant *fingers*)
- Simulation $\neq$ deployment
Lessons learned with KadoP

- Performant message routing (redundant fingers)
- Simulation ≠ deployment
- (Some) DHTs were not built for intensive, detailed indexing. This somehow improved with time.
Lessons learned with KadoP

- Performant message routing (redundant *fingers*)
- Simulation ≠ deployment
- (Some) DHTs were not built for intensive, detailed indexing. This somehow improved with time.
- Indexing takes time (orders of magnitude *wrt* first try)
Lessons learned with KadoP

- Performant message routing (redundant fingers)
- Simulation $\neq$ deployment
- (Some) DHTs were not built for intensive, detailed indexing. This somehow improved with time.
- Indexing takes time (orders of magnitude \textit{wrt} first try)
- Parallelism a big plus
Introduction

KadoP XML indexing
  - Indexing and query processing
  - Scaling up

ViP2P: mat. views on DHTs
  - Algebraic query rewriting in ViP2P
  - View materialization
  - View indexing
  - Query rewriting

Summary
ViP2P: views in peer-to-peer

Joint work with:
Spyros Zoupanos, Alin Tilea, Konstantinos Karanasos, Silviu Julean, Julien Leblay (INRIA Gemo)
Materialized XML 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
  - Execution of a (distributed) physical plan
ViP2P architecture

ViP2P: mat. views on DHTs

Web data management on DHTs
ViP2P architecture

The peers may store:
- documents
ViP2P architecture

The peers may store:
- documents
- views
ViP2P architecture

When $q$ arrives:

Web data management on DHTs
ViP2P architecture

When $q$ arrives:
- view definition
- lookup
ViP2P architecture

When $q$ arrives:
- view definition
- lookup
- rewriting
ViP2P architecture

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

Web data management on DHTs
When $d$ arrives:

1. Search view definitions for which $\forall(v_i(d) \neq \emptyset)$
2. Compute $v_i(d)$
3. Send results
ViP2P architecture

When $d$ arrives:

- search view definitions for which $v_i(d) \neq \emptyset$
ViP2P architecture

When $d$ arrives:
- search view definitions for which $v_i(d) \neq \emptyset$
- compute $v_i(d)$
ViP2P architecture

When \( d \) arrives:
- search view definitions for which \( v_i(d) \neq \emptyset \)
- compute \( v_i(d) \)
- send results
Tree pattern language for views and queries

$\mathcal{a}_{cont}$
Tree pattern language for views and queries

\[ a_{\text{cont}} \quad a_{\text{id}, \text{cont}} \]
Tree pattern language for views and queries

\[ a_{cont} \quad a_{id,cont} \quad a_{id,val} \]
Tree pattern language for views and queries

\[ a_{\text{cont}} \quad a_{\text{id,cont}} \quad a_{\text{id,val}} \]

\[ a_{\text{id}} \quad | \quad b_{\text{val}} \]
Tree pattern language for views and queries

\[ a_{cont} \quad a_{id, cont} \quad a_{id, val} \]

\[ a_{id} \]

\[ b_{val} \]

\[ a_{val} \]

\[ b_{id} \quad d_{cont} \]

\[ c_{cont} \quad e_{id, val} \]
Tree pattern semantics

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

viP2P: mat. views on DHTs

Web data management on DHTs
Tree pattern semantics

```
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_{val} | b_{cont} | f_{id} |
### Tree pattern semantics

```
   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>$a_{val}$</th>
<th>$b_{cont}$</th>
<th>$f_{id}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>some text</td>
<td>$&lt;b&gt;&lt;c&gt;&lt;d/&lt;/e&gt;some&lt;/e&gt;&lt;/c&gt;&lt;/b&gt;$</td>
<td>(7,6)</td>
</tr>
</tbody>
</table>
Tree pattern semantics

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

<table>
<thead>
<tr>
<th>$a_{val}$</th>
<th>$b_{cont}$</th>
<th>$f_{id}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>some text</td>
<td>$\langle b\rangle\langle c\rangle\langle d/\rangle\langle e\rangle\text{some}\langle/e\rangle\langle/c\rangle\langle/b\rangle$</td>
<td>(7,6)</td>
</tr>
<tr>
<td>some text</td>
<td>$\langle b\rangle\langle b\rangle\langle g\rangle\text{text}\langle/g\rangle\langle/b\rangle\langle/b\rangle$</td>
<td>(7,6)</td>
</tr>
</tbody>
</table>
Tree pattern semantics

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

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

Web data management on DHTs
1. Peer $p$ has a view $v$, peer $p_d$ publishes a document $d$
2. $p$ indexes $v$ on the DHT by the $v$ labels
3. $p_d$ looks up the labels and keywords of $d$ $\rightarrow$ 
a superset of all the views $v$ to which $d$ contributes
4. $p_d$ evaluates $v(d)$ for each $v$, sends the results to the peer storing $v$
View building

2000 XMark documents and 500 views (70 views contribute to all the documents)
Indexing views for query rewriting

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

4 different strategies

- **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
  - Look up by all node labels of $q$
Indexing views for query rewriting

- **Leaf path indexing** (LPI):
  - Index \( v \) by all its distinct root-to-leaf paths
  - Look up all the sub-paths of root-to-leaf \( q \) paths

- **Return Path Indexing** (RPI):
  - Index \( v \) by all rooted paths ending in a return node
  - Look up all the sub-paths of root-to-leaf \( q \) paths
We used 1440 views related to but different from query $q$. 

**View look up performance**

![Chart showing view look up performance for different indexing strategies.](chart.png)
We used 1440 views related to but different from query $q$
Finding query rewrites

Rewriting = equivalent algebraic expression over the views

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

Rewriting = equivalent algebraic expression over the views

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

\[ q \quad v_1 \quad v_2 \]

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

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

Rewriting = equivalent algebraic expression over the views

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

Rewriting = equivalent algebraic expression over the views

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

Rewriting = equivalent algebraic expression over the views

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

\[ q \quad \quad v_1 \quad \quad v_2 \]

\[
\begin{align*}
  & a \\
  & \quad b \\
  & \quad \quad c_{val} \\
  & \quad \quad d_{val}
\end{align*}
\]

\[
\begin{align*}
  & a_{id} \\
  & \quad b_{id} \\
  & \quad \quad e \\
  & \quad \quad c_{val} \\
  & \quad \quad d_{val}
\end{align*}
\]
Finding query rewritings

Rewriting = equivalent algebraic expression over the views

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

\[ q \]

\[ a \]
\[ b \]
\[ c_{val} \]
\[ d_{val} \]

\[ v_1 \]
\[ a_{id} \]
\[ b_{id} \]
\[ c_{val} \]
\[ d_{val} \]
\[ e \]

\[ v_2 \]
\[ a_{id} \]
\[ b_{id} \]
\[ c_{val} \]
\[ d_{val} \]

No rewriting
Finding query rewritings

Rewriting = equivalent algebraic expression over the views

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

\[q \quad v_1 \quad v_2\]

\[a \quad a_{id}\]
\[b \quad b\]
\[c \quad c_{id}\]
\[d_{val} \quad e_{val}\]
\[d_{val} \quad e_{val}\]
Finding query rewritings

Rewriting = equivalent algebraic expression over the views

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

No rewriting
Finding query rewritings

Rewriting = equivalent algebraic expression over the views

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

Rewriting = equivalent algebraic expression over the views

Idea:

Compute covers of the query nodes with the view nodes.

No rewriting
Rewriting algorithms

- SE (Subset Enumeration): Try all view subsets
- ISE (Increasing Subset Enumeration): Enumerate subsets from the smallest to the largest
- Bottom-up algorithms can save work: 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
Rewriting algorithms

SE (Subset Enumeration)

- Try all view subsets
Rewriting algorithms

SE (Subset Enumeration)
- Try all view subsets

ISE (Increasing Subset Enumeration)
- Enumerate subsets from the smallest to the largest
Rewriting algorithms

SE (Subset Enumeration)

- Try all view subsets

ISE (Increasing Subset Enumeration)

- Enumerate subsets from the smallest to the largest

Bottom-up algorithms can save work

Use smaller partial rewritings to build bigger ones
Rewriting algorithms

SE (Subset Enumeration)
- Try all view subsets

ISE (Increasing Subset Enumeration)
- Enumerate subsets from the smallest to the largest

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

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

SE (Subset Enumeration)

- Try all view subsets

ISE (Increasing Subset Enumeration)

- Enumerate subsets from the smallest to the largest

Bottom-up algorithms can save work

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

$q$

$a$

$b$

$d_{\text{val}=5}$

$c_{id,\text{cont}}$

$e_{\text{val}}$

$v_1$

$v_2$

$v_3$

$a_{id}$

$b_{id,\text{cont}}$

$c_{id,\text{cont}}$

$d_{\text{val}}$
Bottom-up rewriting

\[ q \]

\[ a \rightarrow b \rightarrow d_{[val=5]} \rightarrow c_{id,cont} \rightarrow e_{val} \]

\[ v_1 \rightarrow v_2 \rightarrow v_3 \]

\[ a_id \rightarrow b_id \rightarrow nav_{b.cont,e.val} \rightarrow v_2 \rightarrow v_1 \]

\[ b_id,cont \rightarrow c_id,cont \rightarrow d_{val} \]
Bottom-up rewriting

```
q

a

b
  \[\text{id, cont}\]
  e_{\text{val}}

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

\bowtie_{\text{a.id} < \text{b.id}}

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

v_1
v_2
v_3

v_1
```
Bottom-up rewriting

Web data management on DHTs
Bottom-up rewriting

$q$

\[ a \]

\[ b \]

\[ d_{val=5} \]

\[ c_{id,cont} \]

\[ e_{val} \]

\[ b_{id,cont} \]

\[ c_{id,cont} \]

\[ d_{val} \]

\[ \pi_{c.cont,e.val} \]

\[ \bowtie a.id=a.id \]

\[ \sigma_{b.id<c.id} \]

\[ \sigma_{val=5} \]

\[ \bowtie a.id<b.id \]

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

\[ v_1 \]

\[ v_2 \]

\[ v_3 \]
Rewriting algorithms trade-offs

SE, ISE, DPR and DFR are correct and complete. They produce all minimal canonical rewritings of \( q \) given \( \mathcal{V} \).

Heuristic for quality of rewriting: number of views

- DFR typically finds one rewriting fast. Not guaranteed to be the best
- ISE, DPR find the best rewriting first, but may take much longer

Could also consider closeness among views, query peers
Performance of rewriting algorithms

![Graphs showing the performance of rewriting algorithms for different query sizes and algorithms.](image)
Query execution: sample plan

project@
bordeau-25.bordeaux.grid5000.fr

receive@
bordeau-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

- project@bordereau-25.bordeaux.grid5000.fr
- receive@bordereau-25.bordeaux.grid5000.fr
- hashJoin@pastel-79.toulouse.grid5000.fr
- receive@pastel-79.toulouse.grid5000.fr
- scan(4nodesView)@pastel-79.toulouse.grid5000.fr
- scan(3nodesView)@griffon-92.nancy.grid5000.fr

Graph:
- Total execution time
- Time to first result

Number of results vs. Time (seconds)
Outline

1. Introduction

2. KadoP XML indexing
   - Indexing and query processing
   - Scaling up

3. ViP2P: mat. views on DHTs
   - Algebraic query rewriting in ViP2P
   - View materialization
   - View indexing
   - Query rewriting

4. Summary
Closest related works

DHT-based sharing of relations [LHSH04]
DHT-based XML indexing [GWJD03, BC06, SHA05, AMP+08]
DHT-based shared XML caches [LP08]
XPath query rewriting [BOB+04, XO05, CDO08, TYÖ+08]
  - XPath: wildcard *, union
  - Rewritings: intersection, navigations, joins

Rewriting with structural constraints [ABMP07]
  - Centralized setting
  - Dataguide [GW97] constraints

Layered architecture for Web content warehousing [AAC+08]
RDF querying and reasoning on DHT [KMK08, LIK06]
Perspectives and ongoing work

Our work:

- Consolidate the lower layers (reliability)
- Native support for RDF, annotated documents, mappings, inter-document links
- RDF views

Other issues:

- Persistence model
- Benchmarks, repeatability
- Connection with other annotated databases
Thank you!


[BC06] Angela Bonifati and Alfredo Cuzzocrea.
### Summary

Storing and retrieving XPath fragments in structured P2P networks.

*Data Knowl. Eng.*, 59(2), 2006.

**[BOB⁺04]**  
A framework for using materialized XPath views in XML query processing.  

**[CDO08]**  
Bogdan Cautis, Alin Deutsch, and Nicola Onose.  
Xpath rewriting using multiple views: Achieving completeness and efficiency.  

**[GW97]**  
Roy Goldman and Jennifer Widom.  
Dataguides: Enabling query formulation and optimization in semistructured databases.  
Locating data sources in large distributed systems.

[KMK08] Zoi Kaoudi, Iris Miliaraki, and Manolis Koubarakis.
RDFS reasoning and query answering on top of DHTs.

[LHSH04] Boon Thau Loo, Ryan Huebsch, Ion Stoica, and Joseph M. Hellerstein.
The case for a hybrid P2P search infrastructure.

[LIK06] Erietta Liarou, Stratos Idreos, and Manolis Koubarakis.
Evaluating conjunctive triple pattern queries over large structured overlay networks.


[XO05] **W. Xu and M. Ozsoyoglu.** Rewriting XPath queries using materialized views.
In *VLDB*, 2005.