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

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

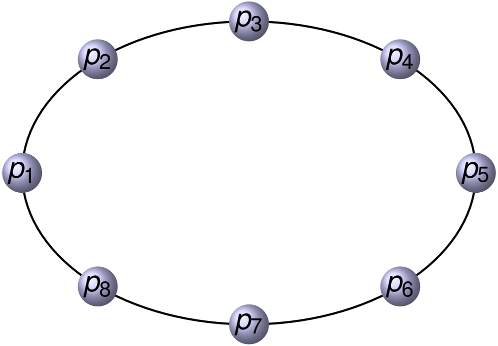
Web content warehouse

Distributed database of selected content, whose users may:

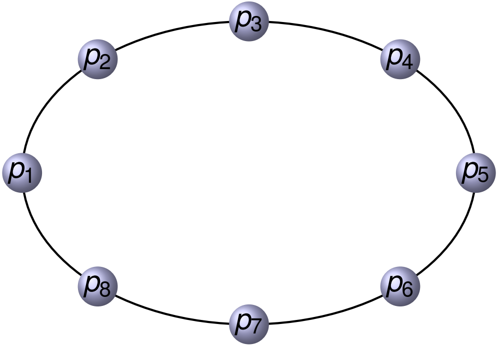
- publish resources
- connect (annotate, map, link...) existing resources
- update resources
- enhance resources by combining them

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

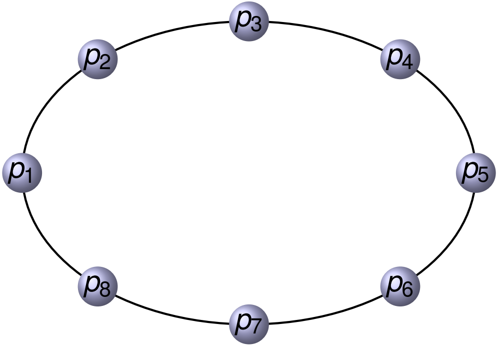
Distributed hash tables



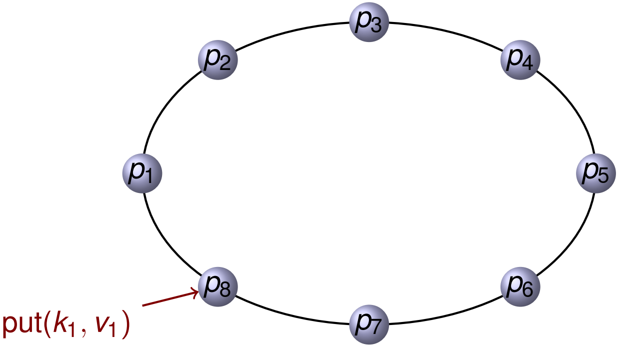
Distributed hash tables



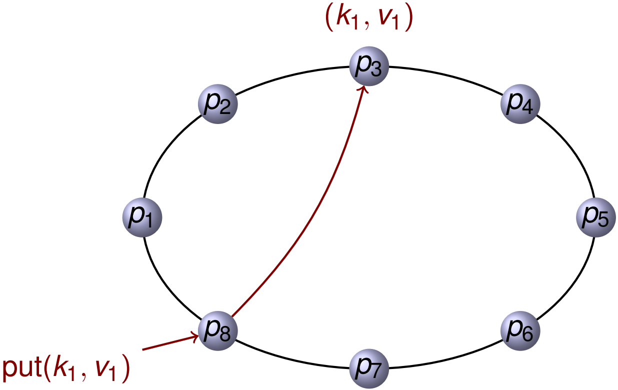
Distributed hash tables



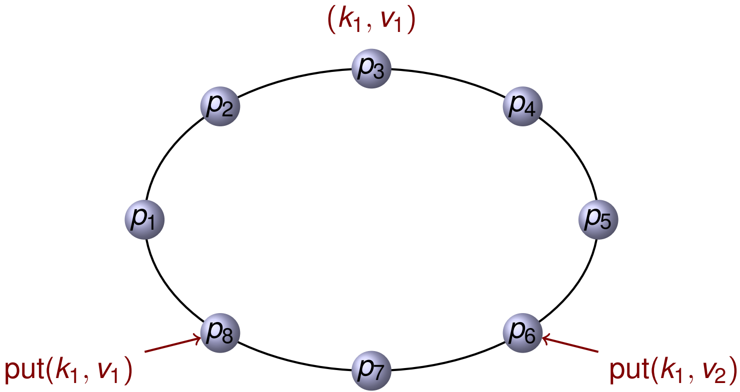
Distributed hash tables



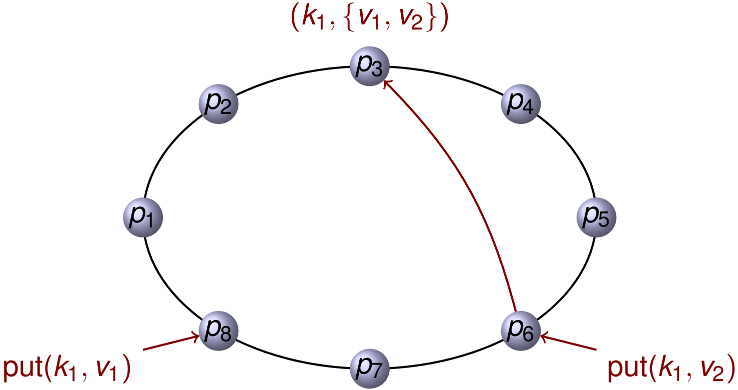
Distributed hash tables



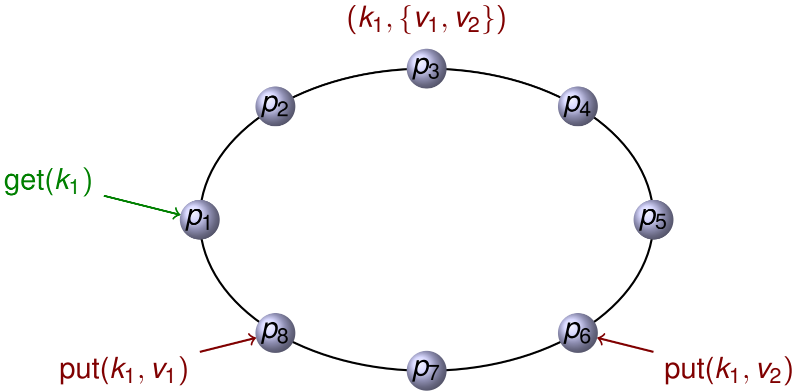
Distributed hash tables



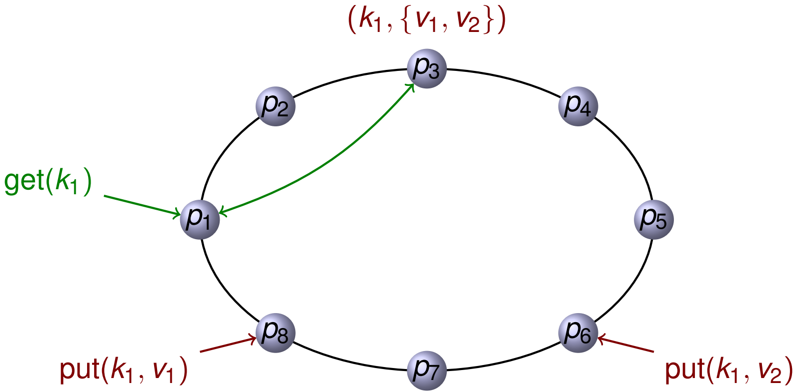
Distributed hash tables



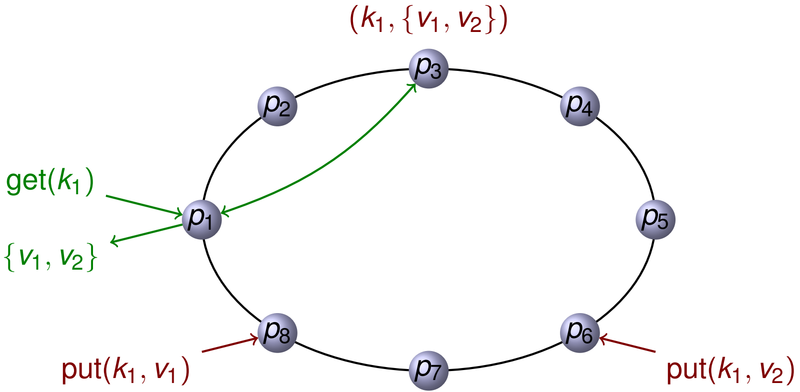
Distributed hash tables



Distributed hash tables



Distributed hash tables



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

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

(Key, value) stores = basis for content sharing

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

D

HTs 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 $\nearrow \Rightarrow$ execution time $\searrow$
- data publication time $\nearrow$, possibly execution time $\nearrow$

Data re-placement or clustering:

- fewer peers contacted for a query (message no. $\searrow$, execution time ?)
- data transfers in the absence of queries (message no. $\nearrow$, total message size $\nearrow$)

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

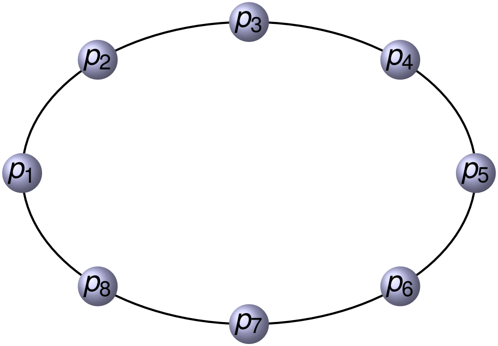
KadoP: DHT-based XML indexing

Joint work with:

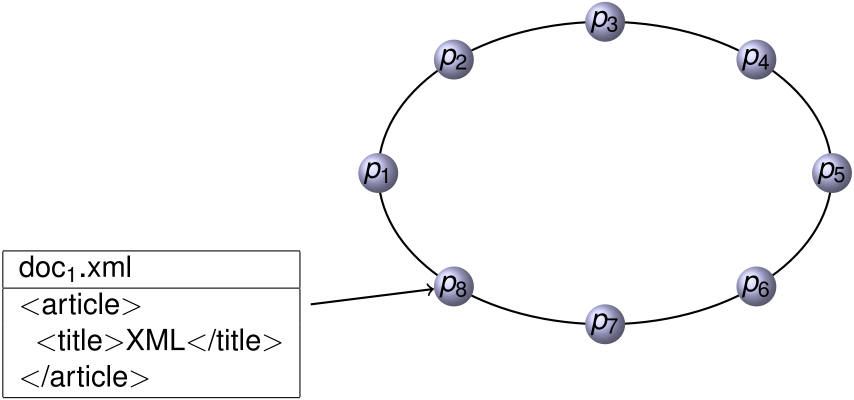
Serge Abiteboul, Nicoleta Preda, Gabriel Vasile, Mohamed Ouazara (INRIA Gemo)
Neoklis Polyzotis, Chong Sun (UCSC) [AMP+08]

Content	XML documents
Queries	Conjunctive tree patterns (including keywords)
Index	Structural IDs of all nodes *and words*

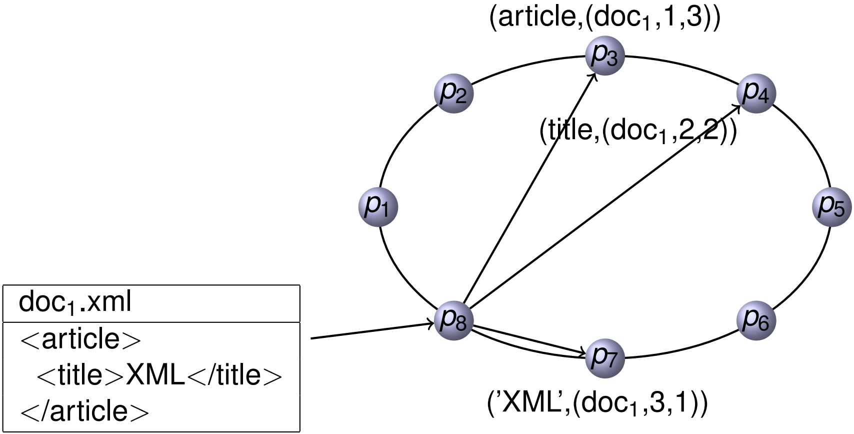
KadoP: DHT-based XML indexing



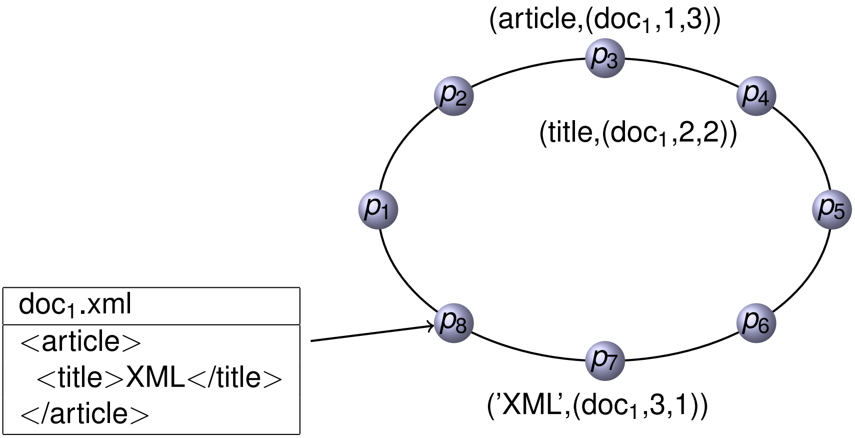
KadoP: DHT-based XML indexing



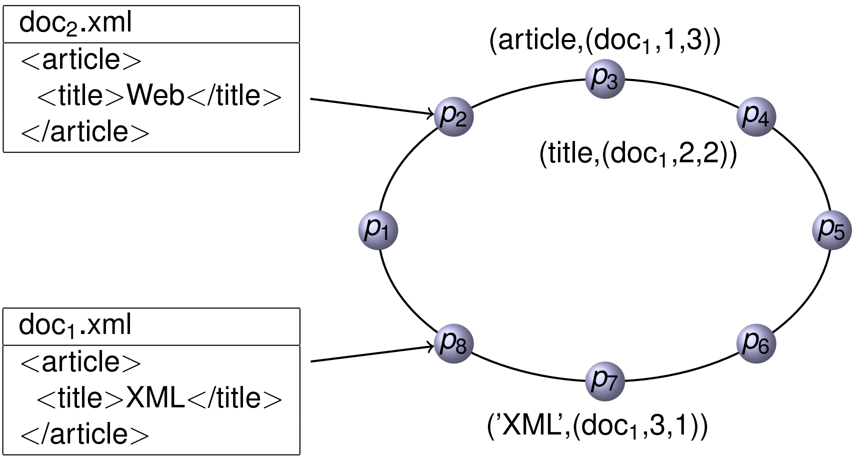
KadoP: DHT-based XML indexing



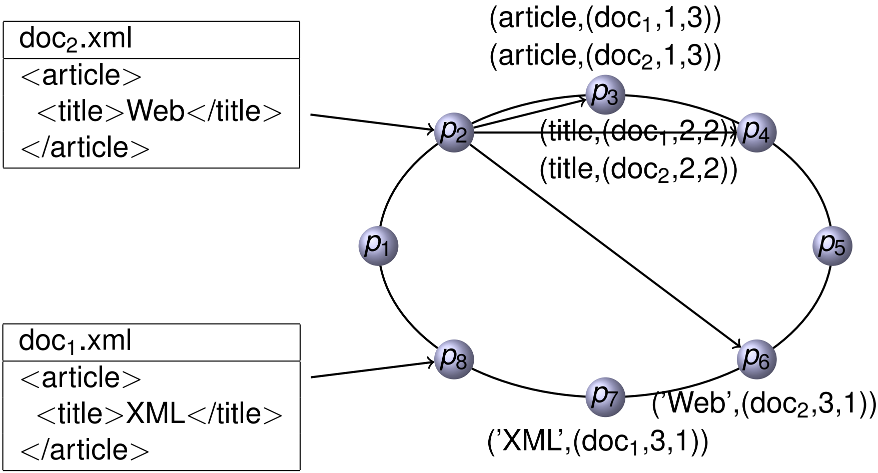
KadoP: DHT-based XML indexing



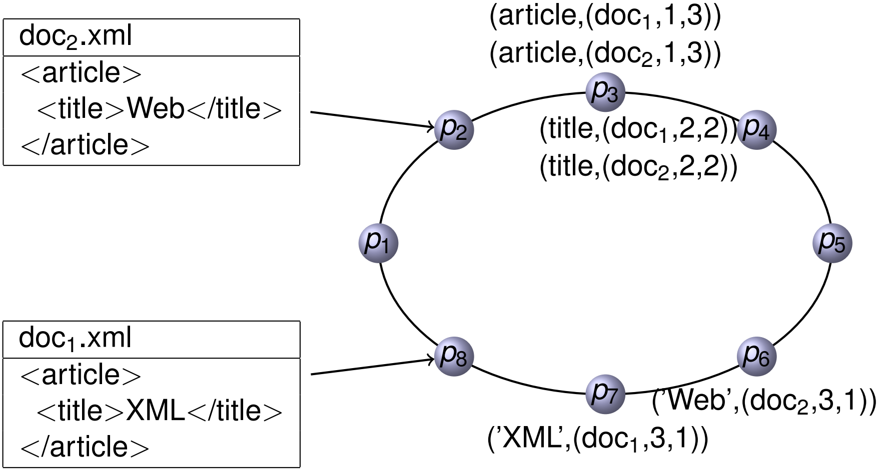
KadoP: DHT-based XML indexing



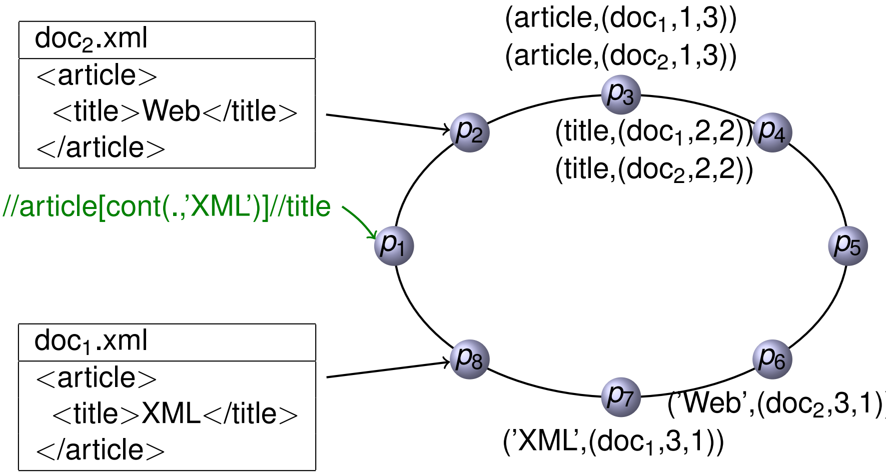
KadoP: DHT-based XML indexing



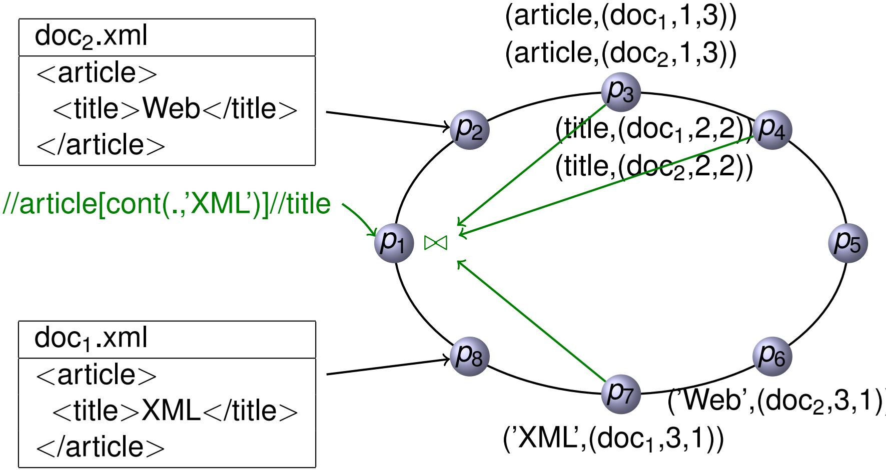
KadoP: DHT-based XML indexing



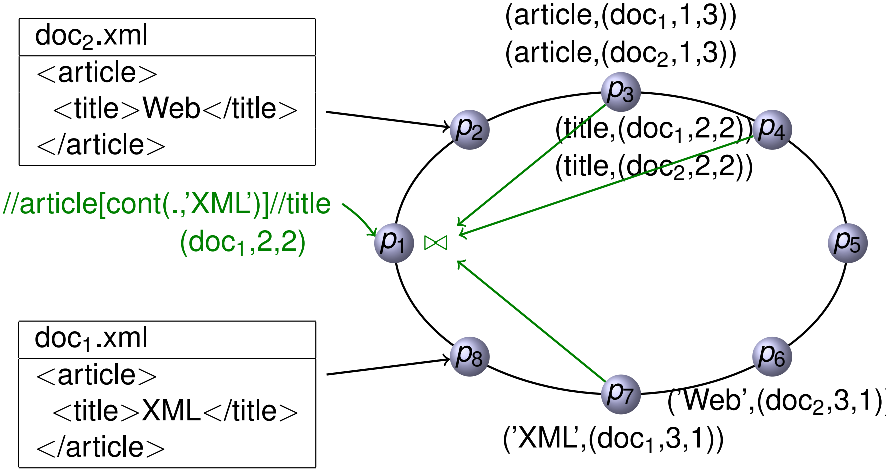
KadoP: DHT-based XML indexing



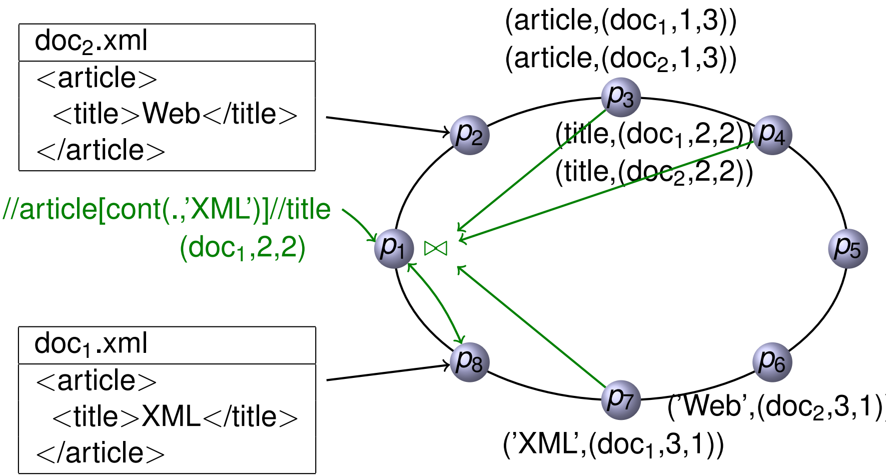
KadoP: DHT-based XML indexing



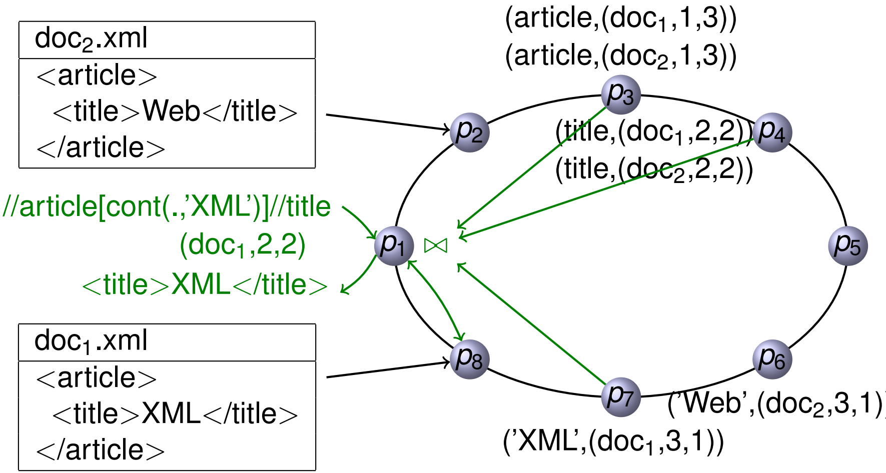
KadoP: DHT-based XML indexing



KadoP: DHT-based XML indexing



KadoP: DHT-based XML indexing



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 *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

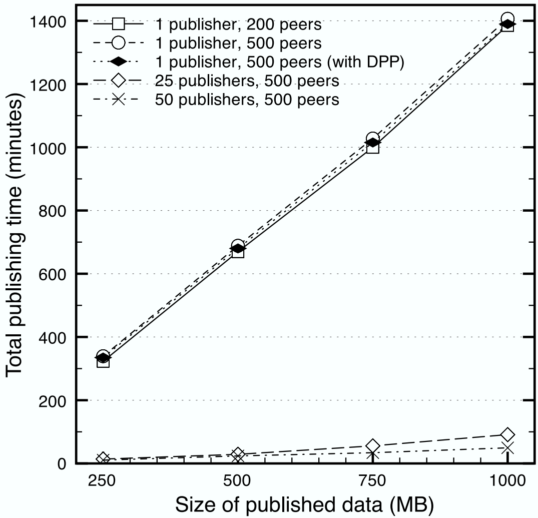
Scalability issue:

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

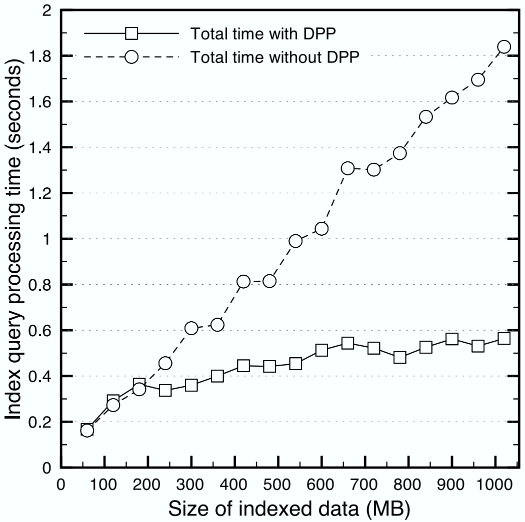
Bloom Filters for KadoP

- Semijoin-like idea
- Posting lists are ordered ⇒ 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



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 $\neq$ 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 $\neq$ 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 *wrt* first try)
- Parallelism a big plus

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

ViP2P: views in peer-to-peer

Joint work with:

Spyros Zoupanos, Alin Tilea, Konstantinos Karanasos, Silviu Julean, Julien Leblay (INRIA Gemo)

ViP2P: views in peer-to-peer

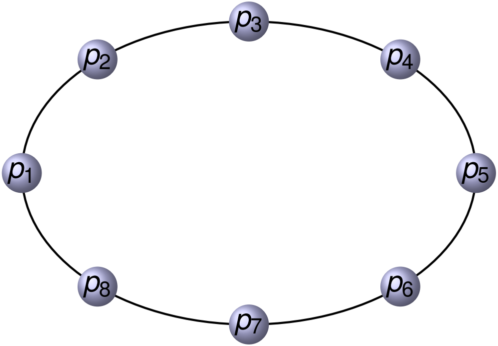
Joint work with:

Spyros Zoupanos, Alin Tilea, Konstantinos Karanasos, Silviu Julean, Julien Leblay (INRIA Gemo)

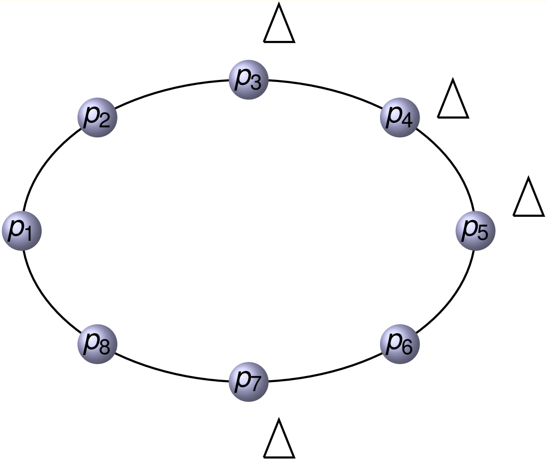
Materialized XML views on a DHT

- 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 ⇒ logical plan
  - Execution of a (distributed) physical plan

ViP2P architecture



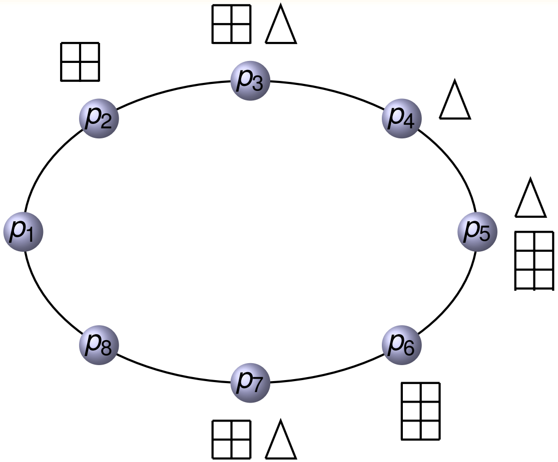
ViP2P architecture



The peers may store:

- documents

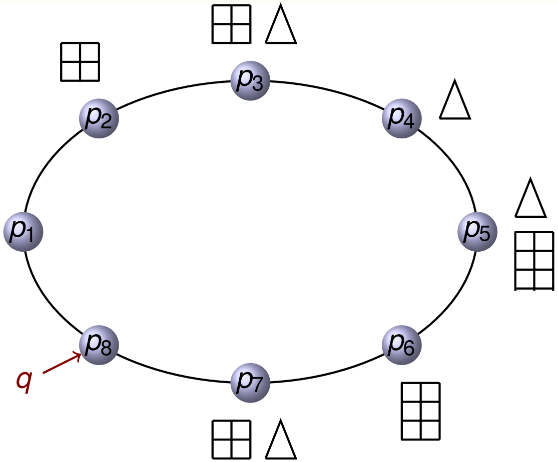
ViP2P architecture



The peers may store:

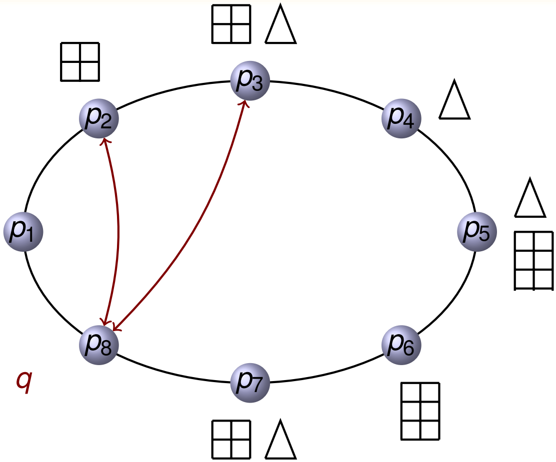
- documents
- views

ViP2P architecture



When q arrives:

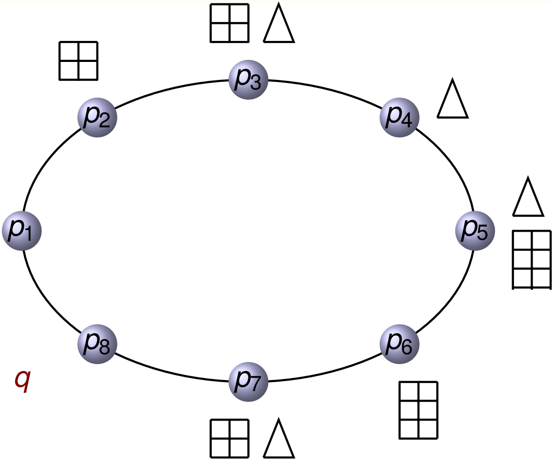
ViP2P architecture



When q arrives:

- view definition lookup

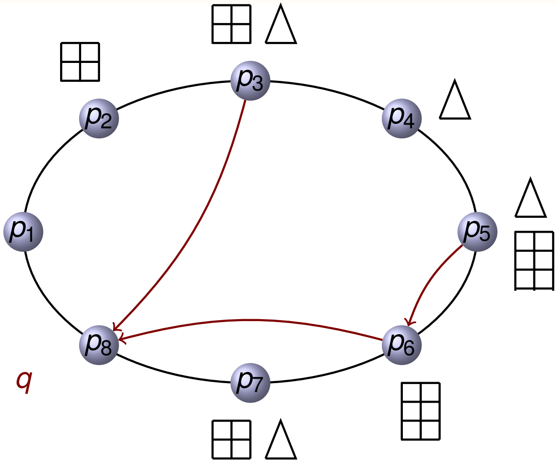
ViP2P architecture



When q arrives:

- view definition lookup
- rewriting

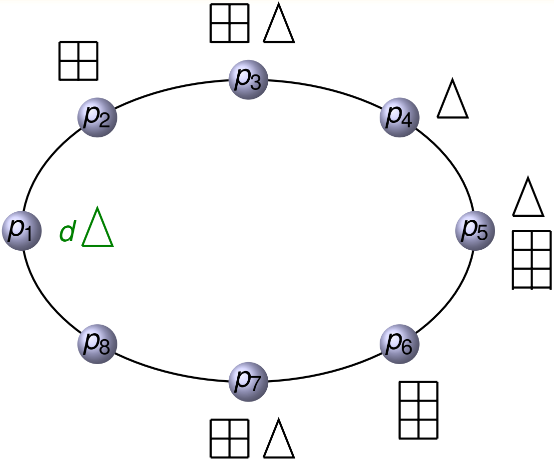
ViP2P architecture



When q arrives:

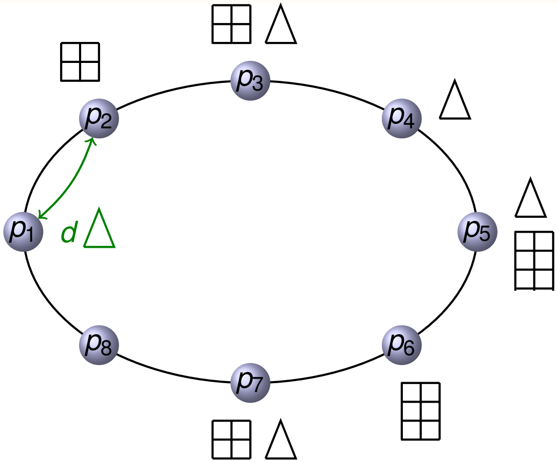
- view definition lookup
- rewriting
- execution of physical plan

ViP2P architecture



When *d* arrives:

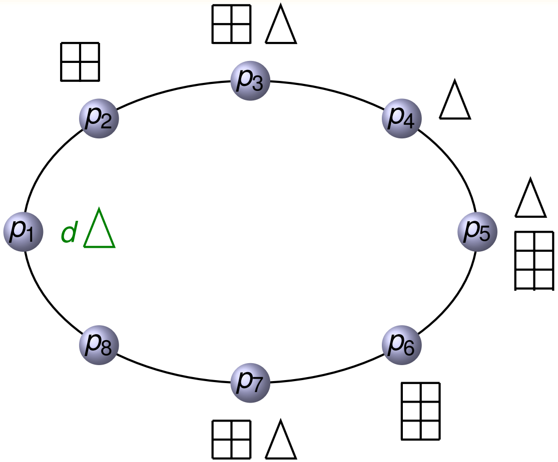
ViP2P architecture



When d arrives:

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

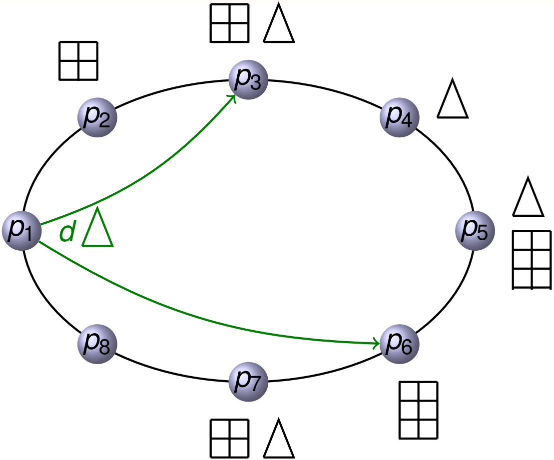
ViP2P architecture



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

a_{cont}

Tree pattern language for views and queries

a_{cont} $a_{id,cont}$

Tree pattern language for views and queries

a_{cont} $a_{id,cont}$ $a_{id,val}$

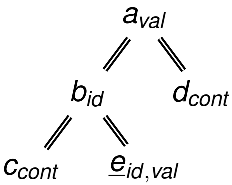
Tree pattern language for views and queries

a_{cont} $a_{id,cont}$ $a_{id,val}$

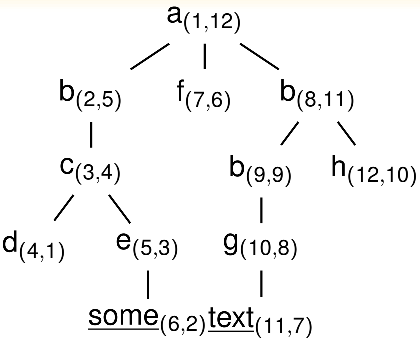
$$\begin{array}{c} a_{id} \\ | \\ b_{val} \end{array}$$

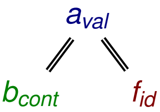
Tree pattern language for views and queries

a_{cont} $a_{id,cont}$ $a_{id,val}$

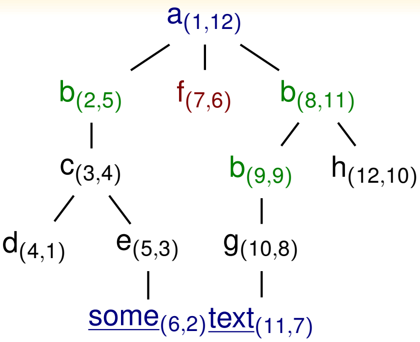


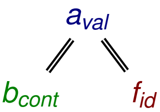
Tree pattern semantics





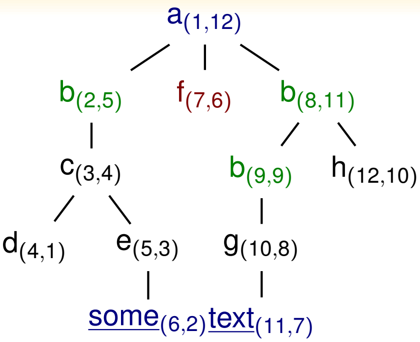
Tree pattern semantics

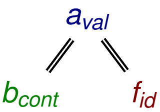




a_{val}	b_{cont}	f_{id}

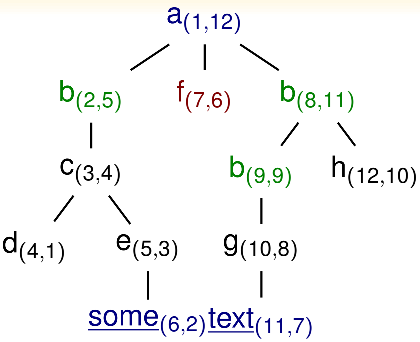
Tree pattern semantics

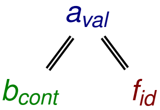




a_{val}	b_{cont}	f_{id}
some text	<c><d/><e>some</e></c>	(7,6)

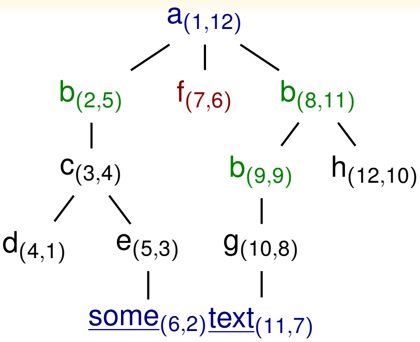
Tree pattern semantics

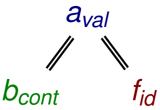




a_{val}	b_{cont}	f_{id}
some text	<c><d/><e>some</e></c>	(7,6)
some text	<g>text</g>	(7,6)

Tree pattern semantics





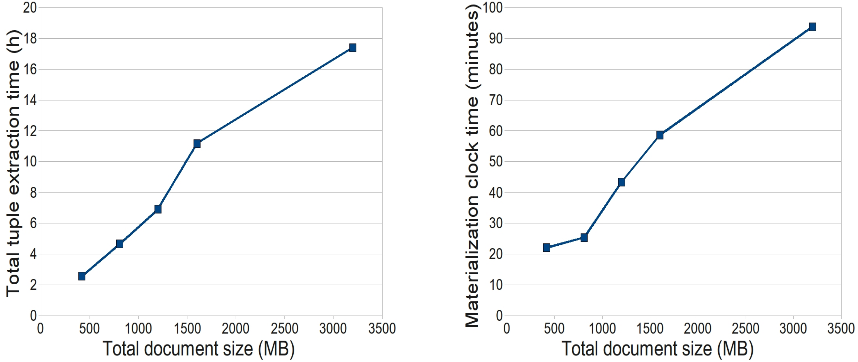
a_{val}	b_{cont}	f_{id}
some text	<c><d/><e>some</e></c>	(7,6)
some text	<g>text</g>	(7,6)
some text	<g>text</g>	(7,6)

View materialization

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

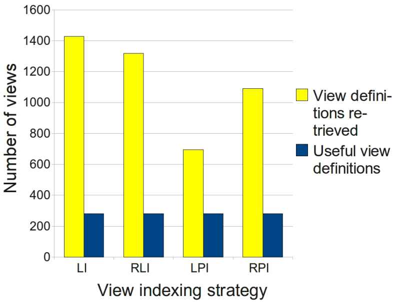
- **Label indexing** (LI):
  - Index v by each v node label
  - Look up by all node labels of q
- **Return label indexing** (RLI):
  - Index v by the labels of all v nodes which project some attributes
  - Look up by all node labels of q

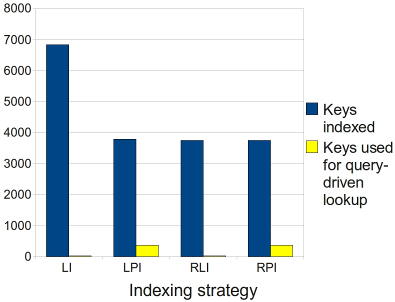
Indexing views for query rewriting

- **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

View look up performance

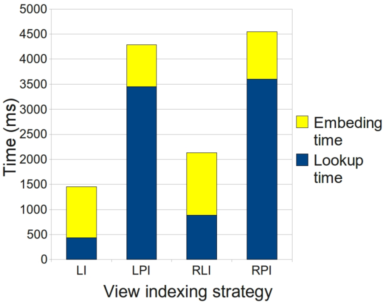
We used 1440 views related to but different from query *q*





View look up performance

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



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.

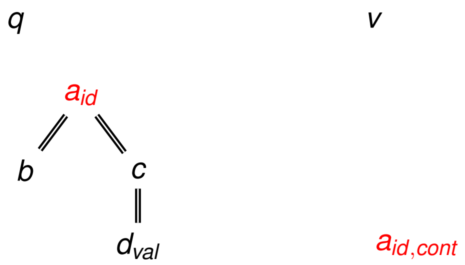
q	v_1	v_2
a_{id}		
$\|$		
b_{id}	a_{id}	b_{id}

Finding query rewritings

Rewriting = equivalent algebraic expression over the views

Idea:

Compute covers of the query nodes with the view nodes.

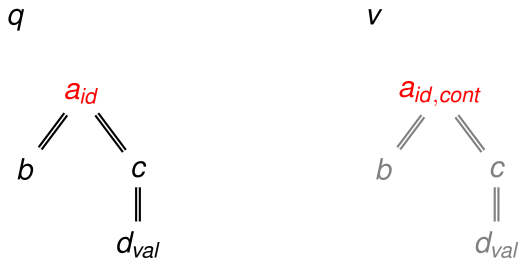


Finding query rewritings

Rewriting = equivalent algebraic expression over the views

Idea:

Compute covers of the query nodes with the view nodes.

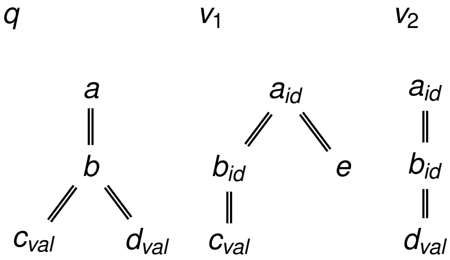


Finding query rewritings

Rewriting = equivalent algebraic expression over the views

Idea:

Compute covers of the query nodes with the view nodes.

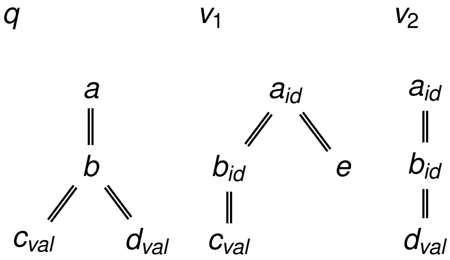


Finding query rewritings

Rewriting = equivalent algebraic expression over the views

Idea:

Compute covers of the query nodes with the view nodes.



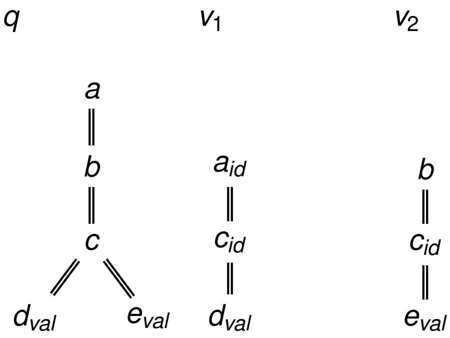
No rewriting

Finding query rewritings

Rewriting = equivalent algebraic expression over the views

Idea:

Compute covers of the query nodes with the view nodes.

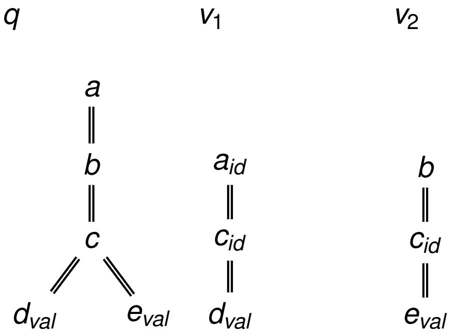


Finding query rewritings

Rewriting = equivalent algebraic expression over the views

Idea:

Compute covers of the query nodes with the view nodes.



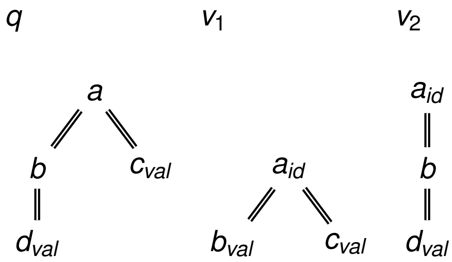
No rewriting

Finding query rewritings

Rewriting = equivalent algebraic expression over the views

Idea:

Compute covers of the query nodes with the view nodes.

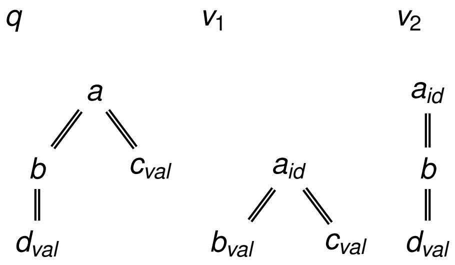


Finding query rewritings

Rewriting = equivalent algebraic expression over the views

Idea:

Compute covers of the query nodes with the view nodes.



No rewriting

Rewriting algorithms

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

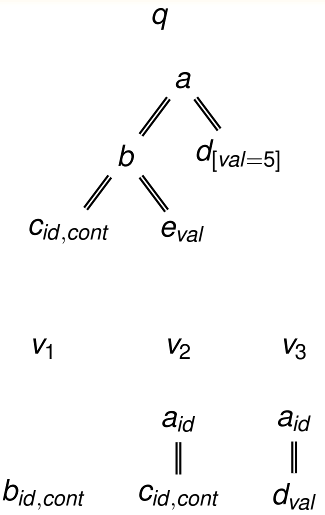
DPR (Dynamic Programming Rewriting)

- Dynamic programming style

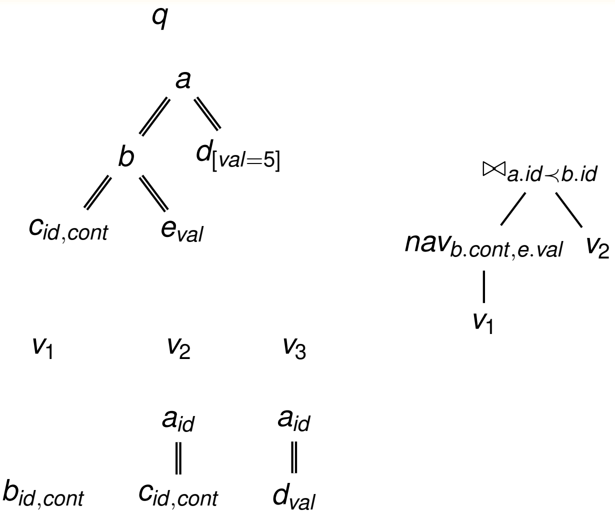
DFR (Depth First Rewriting)

- Greedy based on the biggest query coverage

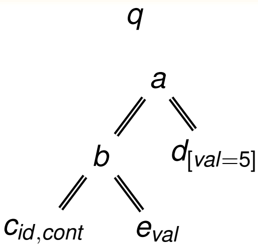
Bottom-up rewriting

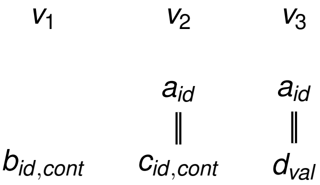


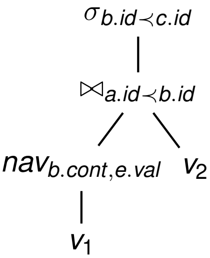
Bottom-up rewriting



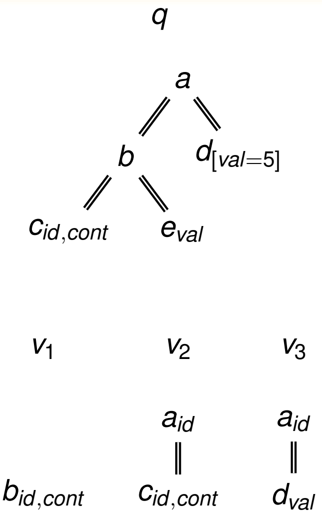
Bottom-up rewriting

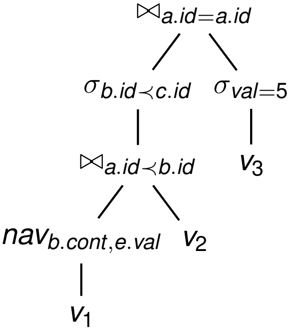




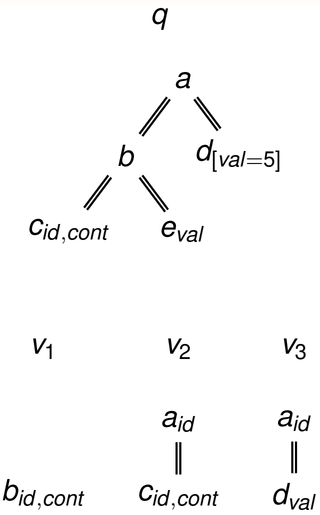


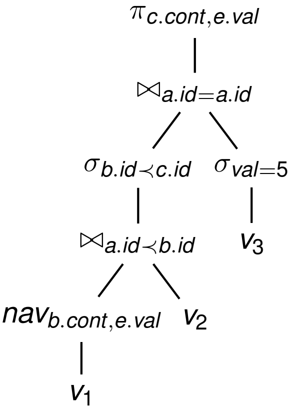
Bottom-up rewriting





Bottom-up rewriting





Rewriting algorithms trade-offs

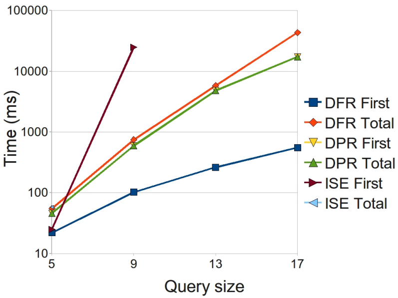
SE, ISE, DPR and DFR are correct and complete.
They produce all minimal canonical rewritings of q given $\mathcal{V}$.

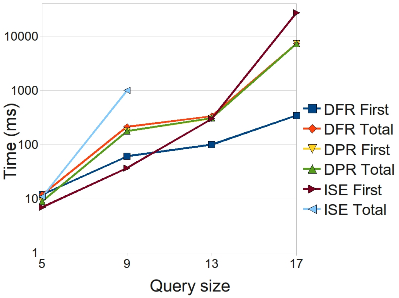
Heuristic for quality of rewriting: number of views

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

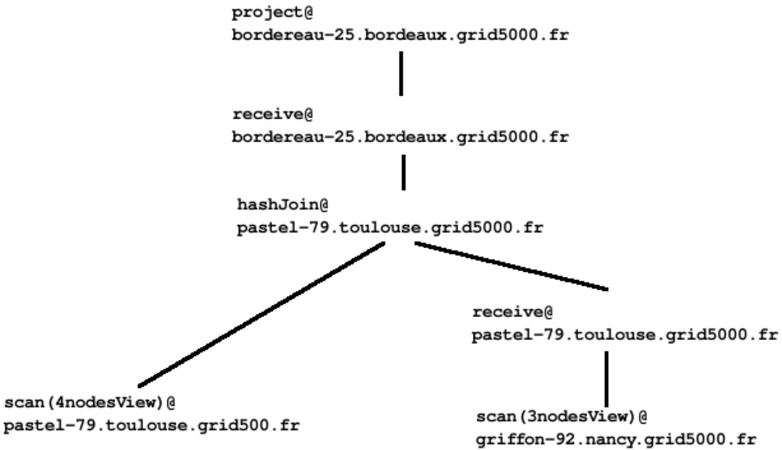
Could also consider closeness among views, query peers

Performance of rewriting algorithms

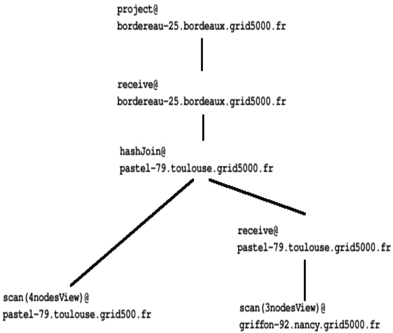


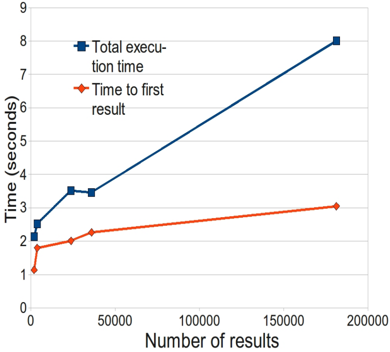


Query execution: sample plan



Query execution





Outline

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 databses

Thank you!

[AAC+08] **Serge Abiteboul, Tristan Allard, Philippe Chatalic, Georges Gardarin, A. Ghitescu, François Goasdoué, Ioana Manolescu, Benjamin Nguyen, M. Ouazara, A. Somani, Nicolas Travers, Gabriel Vasile, and Spyros Zoupanos.**
Webcontent: efficient p2p warehousing of web data.
PVLDB, 1(2):1428–1431, 2008.

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

[AMP+08] **Serge Abiteboul, Ioana Manolescu, Neoklis Polyzotis, Nicoleta Preda, and Chong Sun.**
XML processing in DHT networks.
In *ICDE*, pages 606–615, 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.

[KMK08] **Zoi Kaoudi, Iris Miliaraki, and Manolis Koubarakis.**
RDFS reasoning and query answering on top of DHTs.
In *International Semantic Web Conference*, pages 499–516, 2008.

[LHSH04] **Boon Thau Loo, Ryan Huebsch, Ion Stoica, and Joseph M. Hellerstein.**
The case for a hybrid P2P search infrastructure.
In *IPTPS*, pages 141–150, 2004.

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

In *International Semantic Web Conference*, pages 399–413, 2006.

[LP08] **Kostas Lillis and Evaggelia Pitoura.**
Cooperative XPath caching.
In *SIGMOD Conference*, pages 327–338, 2008.

[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.