Generating Efficient Execution Plans for Vertically Partitioned XML Databases

Patrick Kling, M. Tamer Özsu, and Khuzaima Daudjee

University of Waterloo
David R. Cheriton School of Computer Science

VLDB 2011

The Problem

- Centralized query evaluation techniques for XML well understood
- These techniques do not scale to large collection sizes and heavy workloads
- Goal: use distribution to improve scalability
- Focus on end-to-end cost of query evaluation

Distributed XML Query Evaluation: Two Scenarios

- Integrating multiple data sources
  - Fragmentation is determined by existing data sources
  - Need flexible fragmentation model to express this
- Distribution for performance
  - Choose fragmentation to suit workload
  - Can use more constrained fragmentation model
  - Fragmentation specification allows for distributed query optimization

Distributed XML Query Evaluation: Two Scenarios

- Integrating multiple data sources
  - Fragmentation is determined by existing data sources
  - Need flexible fragmentation model to express this
- Distribution for performance
  - Choose fragmentation to suit workload
  - Can use more constrained fragmentation model
  - Fragmentation specification allows for distributed query optimization

Outline

Outline

Fragmenting XML Collections

- Ad-hoc fragmentation
- Structure-based fragmentation

Ad-hoc fragmentation

- Cut arbitrary edges in document tree
- Highly flexible (good for data integration)
- No explicit fragmentation specification
- Limited potential for exploiting fragmentation characteristics for query optimization
- Not a suitable choice for this work

Structure-based Fragmentation

- Fragmentation according to characteristics of data or schema
- Yields a fragmentation specification that can be exploited for query optimization
- Better choice when distributing for performance

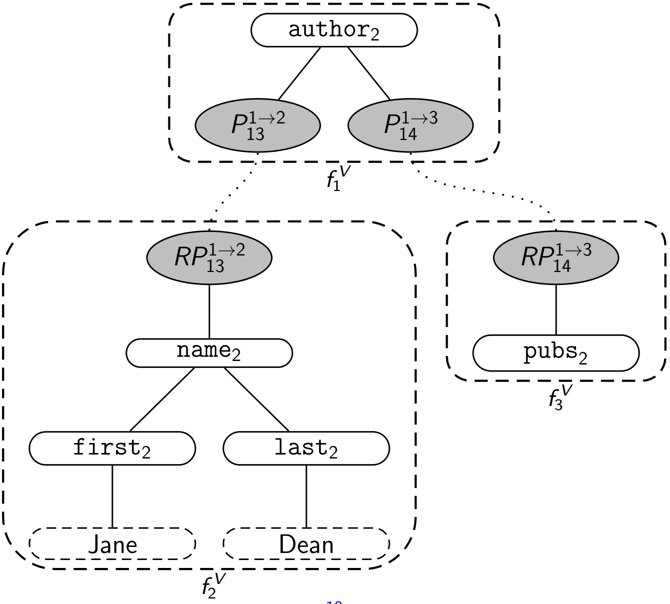
Our Fragmentation Model

- Focus on simplicity and precise fragmentation specification
- Focus on partitioning collection (replication is orthogonal)
- Follow semantics of relational fragmentation techniques
  - Horizontal fragmentation (based on predicates/selection)
  - Vertical fragmentation (based on partitioning of schema/projection)
  - Hybrid fragmentation (combination of horizontal and vertical steps)

Our Fragmentation Model

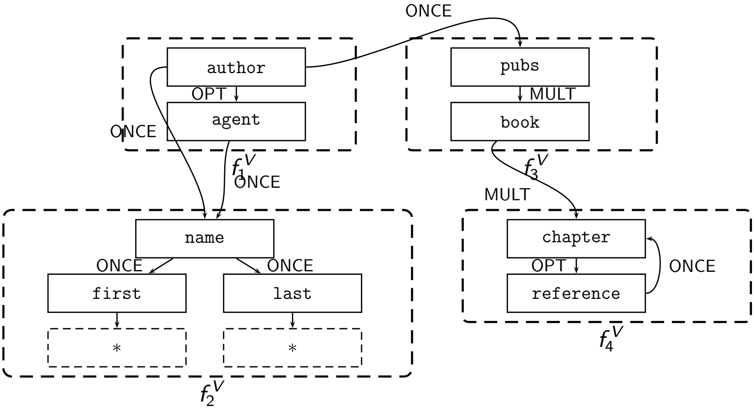
- Focus on simplicity and precise fragmentation specification
- Focus on partitioning collection (replication is orthogonal)
- Follow semantics of relational fragmentation techniques
  - Horizontal fragmentation (based on predicates/selection)
  - Vertical fragmentation (based on partitioning of schema/projection)
  - Hybrid fragmentation (combination of horizontal and vertical steps)

Vertical Fragmentation



Vertical Fragmentation Specification

Vertical fragmentation is specified by a *fragmentation schema*.



UNIVERSITY OF Waterloo

Outline

Query model

XQ, subset of XPath

- Nested paths with child and descendant steps
- Explicit node tests and wild cards
- Value constraints (numeric or textual)
- $Q := \sigma \mid * \mid Q//Q \mid Q/Q \mid Q[q]$

  $q := Q \mid . = / \neq \textit{ str} \mid . = / \neq / \leq / < / \geq / > \textit{ num}$

UNIVERSITY OF Waterloo

Query Example

"Find all references in publications written by authors whose first name is 'William' and whose last name is 'Shakespeare' "

Query Example

"Find all references in publications written by authors whose first name is 'William' and whose last name is 'Shakespeare' "

/author[./name[./first = "William" and
./last = "Shakespeare"]]//reference

Query Example

"Find all references in publications written by authors whose first name is 'William' and whose last name is 'Shakespeare' "

$/\text{author}[./\text{name}[./\text{first} = \text{“William”}\ \text{and}$
$\quad ./\text{last} = \text{“Shakespeare”}]]//\text{reference}$

- Node tests

Query Example

"Find all references in publications written by authors whose first name is 'William' and whose last name is 'Shakespeare' "

/author[./name[./first = "William" and ./last = "Shakespeare"]]//reference

- Node tests
- Value constraints

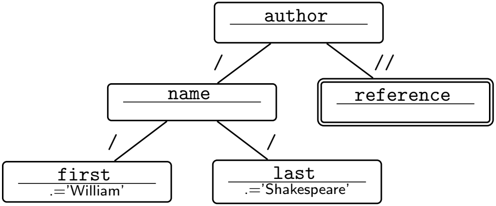
Query Example

"Find all references in publications written by authors whose first name is 'William' and whose last name is 'Shakespeare' "

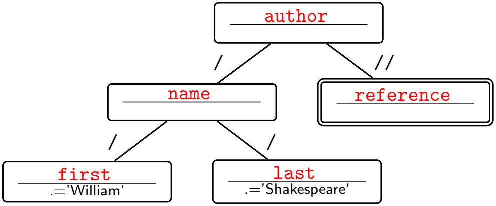
$/\text{author}[./\text{name}[./\text{first} = \text{“William”}\ \text{and}$
$\quad ./\text{last} = \text{“Shakespeare”}]]//\text{reference}$

- Node tests
- Value constraints
- Structural constraints

Tree Patterns

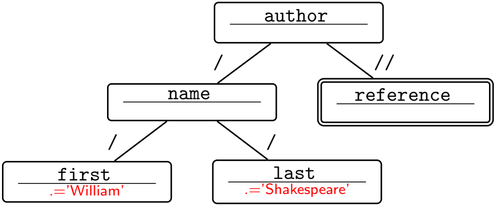


Tree Patterns



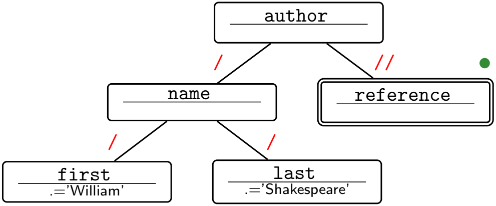
- Pattern nodes with node tests and value constraints

Tree Patterns



- Pattern nodes with node tests and value constraints

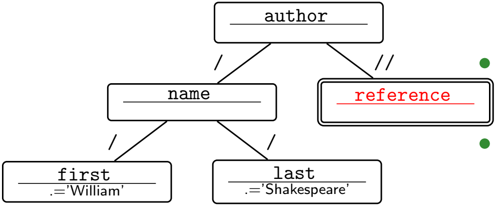
Tree Patterns



- Pattern nodes with node tests and value constraints
- Edges annotated with XPath axes

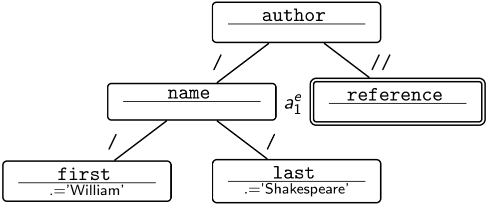
UNIVERSITY OF Waterloo

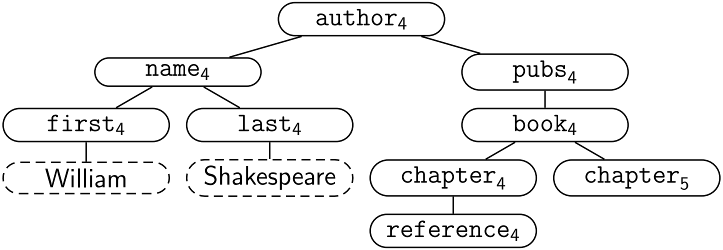
Tree Patterns



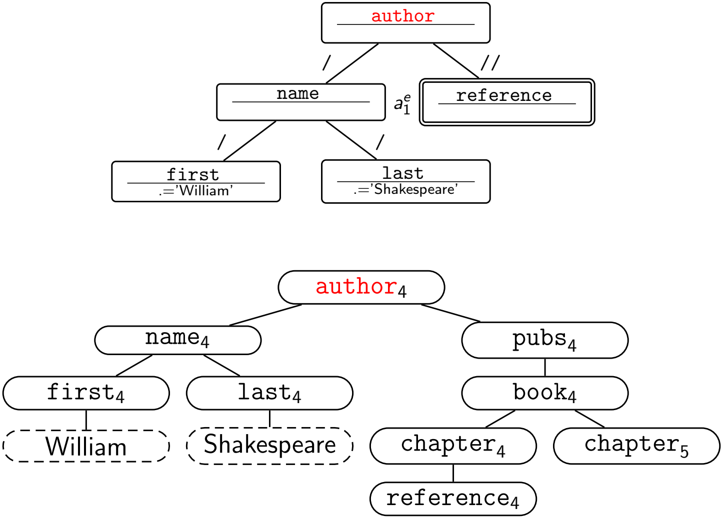
- Pattern nodes with node tests and value constraints
- Edges annotated with XPath axes
- Extraction point nodes

Evaluating Tree Pattern Queries

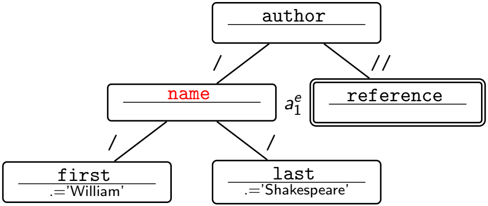


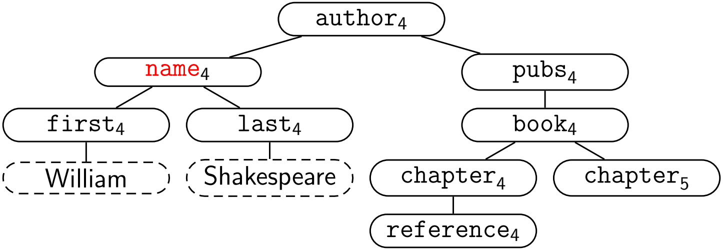


Evaluating Tree Pattern Queries

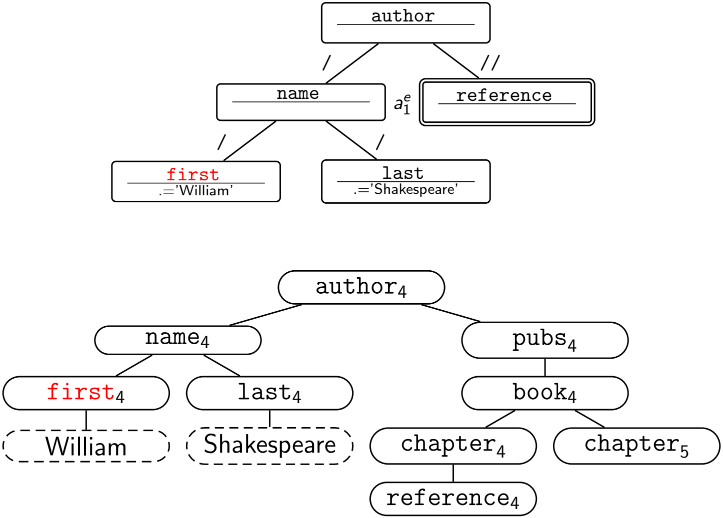


Evaluating Tree Pattern Queries

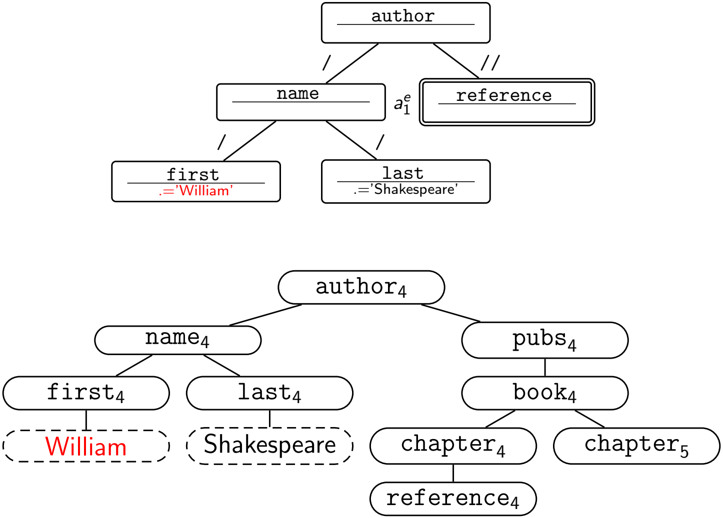




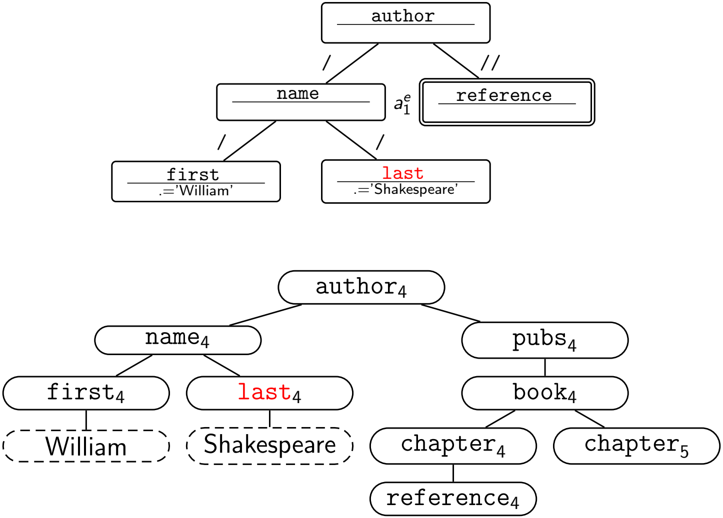
Evaluating Tree Pattern Queries



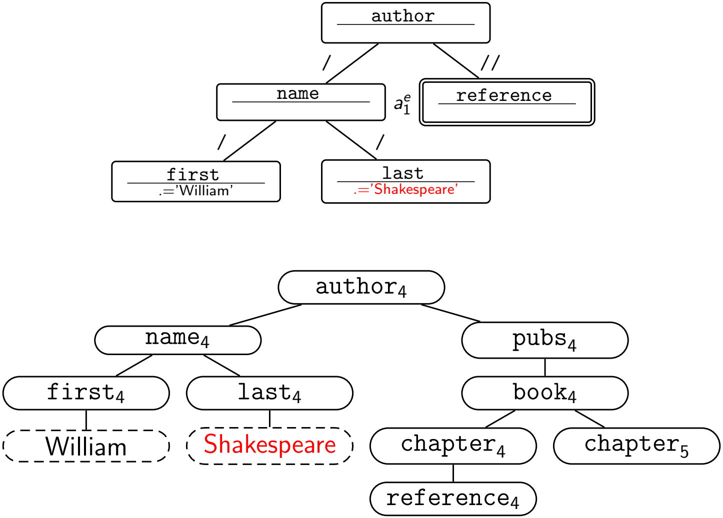
Evaluating Tree Pattern Queries



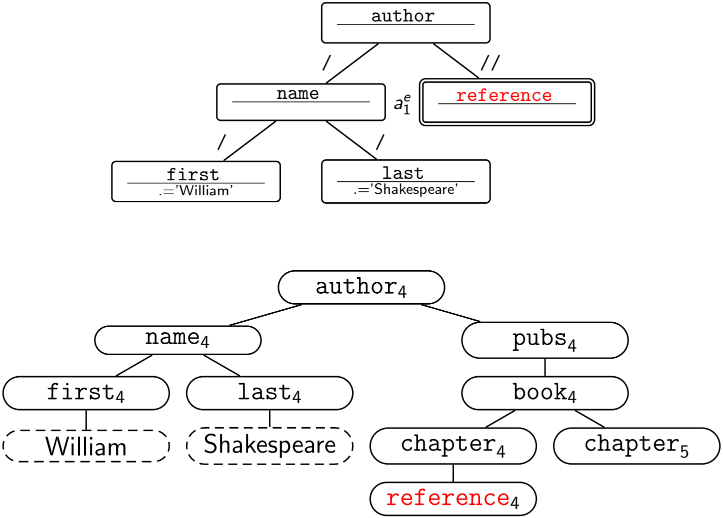
Evaluating Tree Pattern Queries



Evaluating Tree Pattern Queries

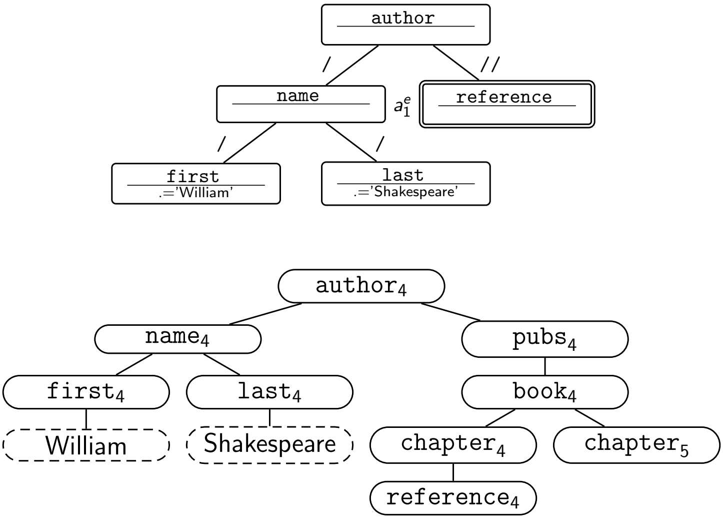


Evaluating Tree Pattern Queries



UNIVERSITY OF Waterloo

Evaluating Tree Pattern Queries



$[a_1^e = \texttt{reference}_4]$

Evaluating Tree Pattern Queries

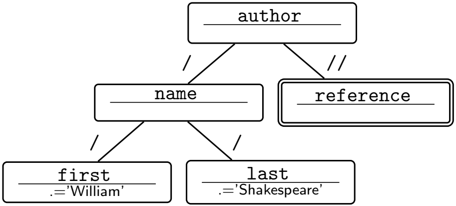
- Various centralized approaches exist
  - Navigating document trees
  - Structural joins
- We leverage these for distributed query evaluation

Querying Vertically Distributed XML Collections

- Input
  - Fragmentation-unaware tree pattern query
  - Fragmentation schema
- Tasks
  - Annotate tree pattern nodes with corresponding fragments
  - Decompose tree pattern into sub-patterns for individual fragments
  - Convert sub-patterns to local plans using existing techniques (each site is free to choose local strategy)
  - Generate distributed execution plan that specifies how results are combined

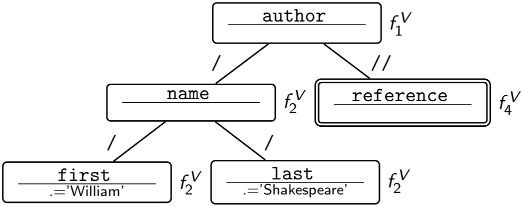
Querying Vertically Distributed XML Collections

- Annotate tree pattern nodes
- Decompose tree pattern
- Convert sub-patterns into local plans
- Generate distributed execution plan



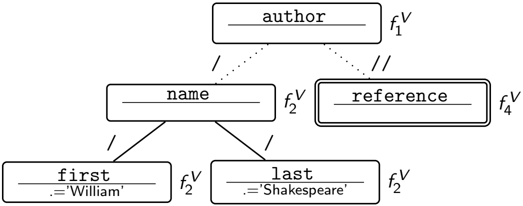
Querying Vertically Distributed XML Collections

- Annotate tree pattern nodes
- Decompose tree pattern
- Convert sub-patterns into local plans
- Generate distributed execution plan



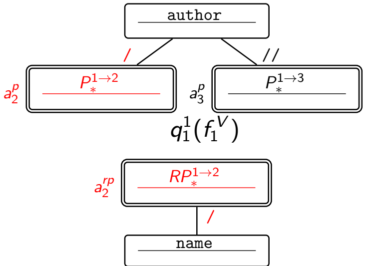
Querying Vertically Distributed XML Collections

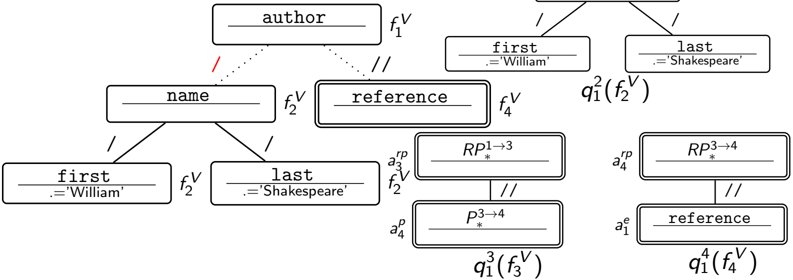
- Annotate tree pattern nodes
- Decompose tree pattern
- Convert sub-patterns into local plans
- Generate distributed execution plan



Querying Vertically Distributed XML Collections

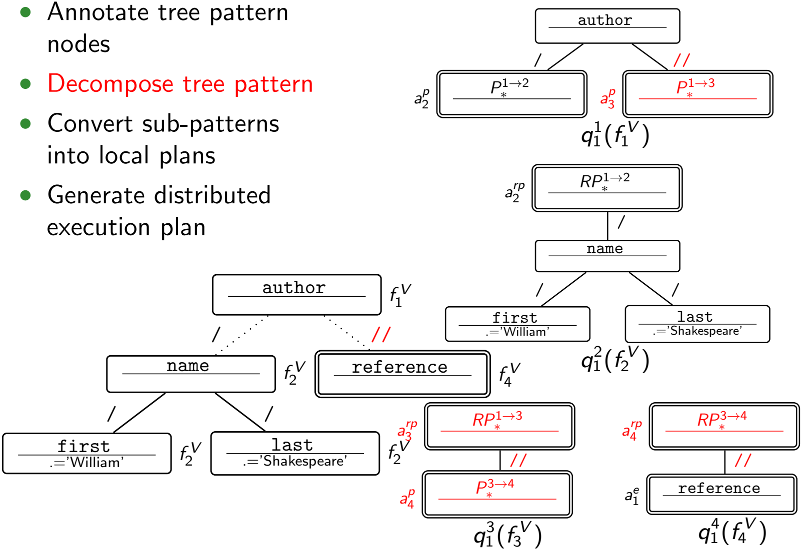
- Annotate tree pattern nodes
- Decompose tree pattern
- Convert sub-patterns into local plans
- Generate distributed execution plan





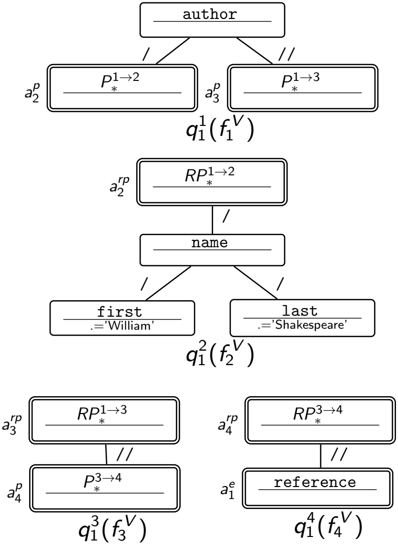
Querying Vertically Distributed XML Collections

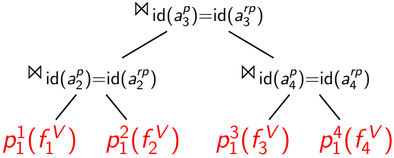
- Annotate tree pattern nodes
- Decompose tree pattern
- Convert sub-patterns into local plans
- Generate distributed execution plan



Querying Vertically Distributed XML Collections

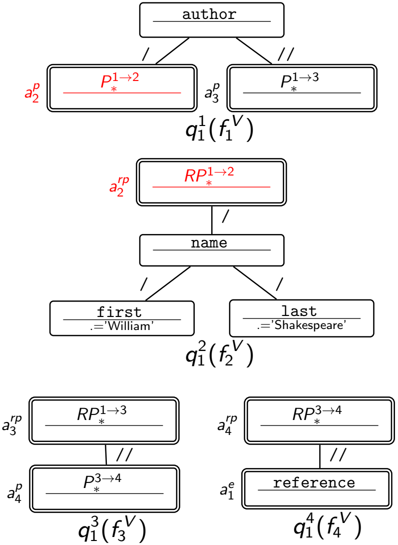
- Annotate tree pattern nodes
- Decompose tree pattern
- Convert sub-patterns into local plans
- Generate distributed execution plan

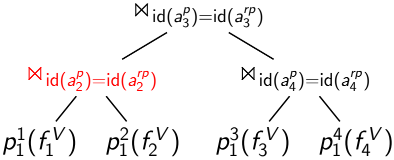




Querying Vertically Distributed XML Collections

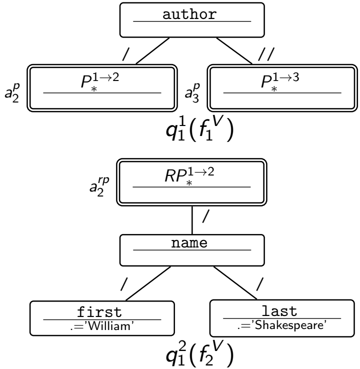
- Annotate tree pattern nodes
- Decompose tree pattern
- Convert sub-patterns into local plans
- Generate distributed execution plan

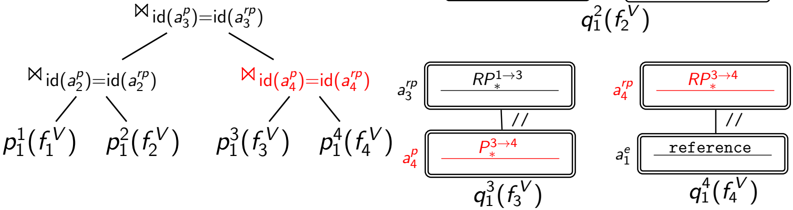




Querying Vertically Distributed XML Collections

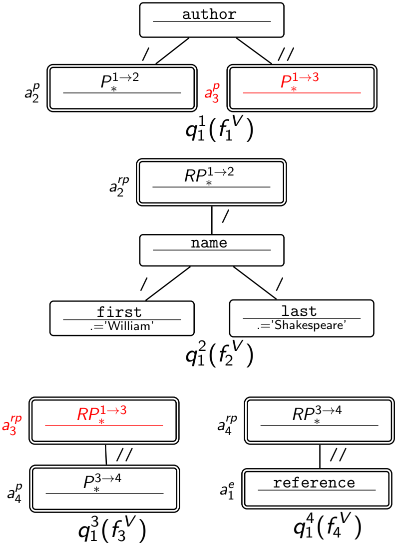
- Annotate tree pattern nodes
- Decompose tree pattern
- Convert sub-patterns into local plans
- Generate distributed execution plan

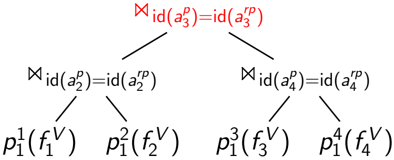




Querying Vertically Distributed XML Collections

- Annotate tree pattern nodes
- Decompose tree pattern
- Convert sub-patterns into local plans
- Generate distributed execution plan





Improving Distributed Execution Plans

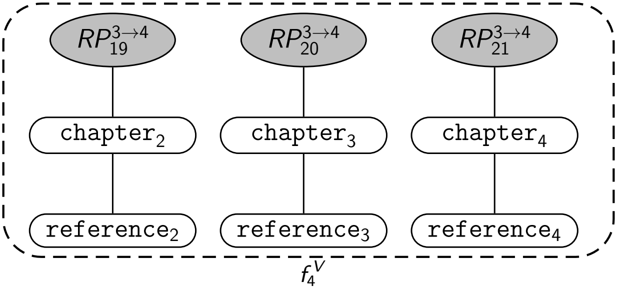
- Pruning irrelevant fragments
- Join order
- Push cross-fragment joins into local plans

Improving Distributed Execution Plans

- Pruning irrelevant fragments
- Join order
- Push cross-fragment joins into local plans

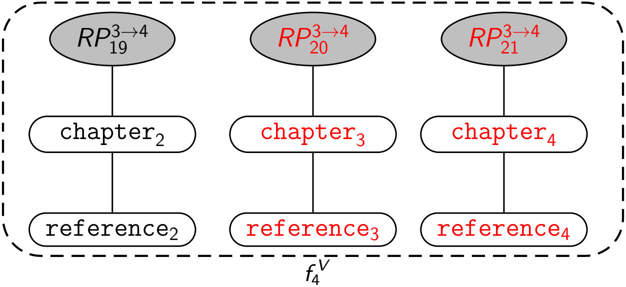
Pushing Cross-Fragment Joins

Large fraction of local results are discarded by cross-fragment join



Pushing Cross-Fragment Joins

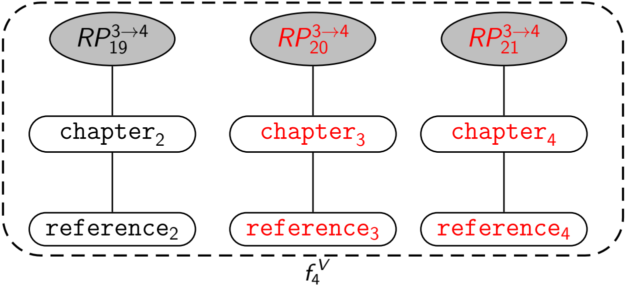
Large fraction of local results are discarded by cross-fragment join



UNIVERSITY OF Waterloo

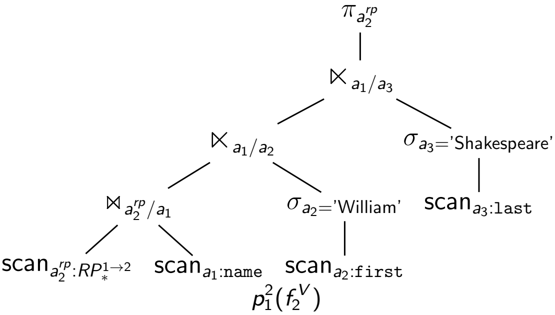
Pushing Cross-Fragment Joins

Large fraction of local results are discarded by cross-fragment join

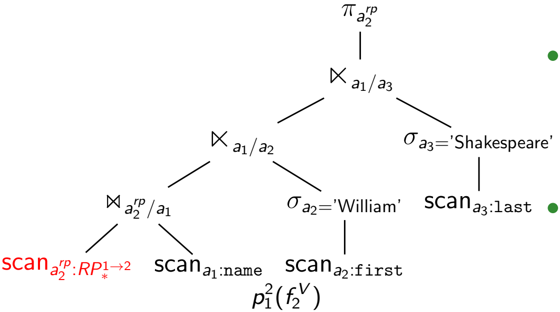


- Idea: only access relevant sub-trees in fragment
- Avoid computing irrelevant local results
- Use pipelining to push cross-fragment join into local plan

A Local Query Plan

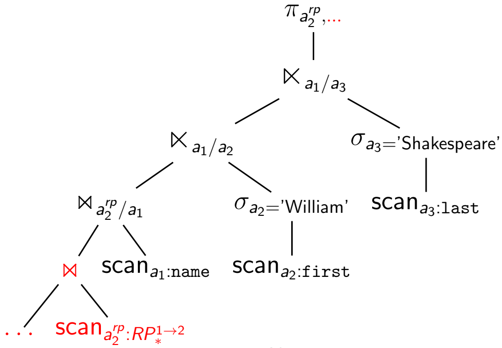


A Local Query Plan



- Plan scans root proxy nodes in fragment
- Idea: filter these root proxy nodes before evaluating remainder of plan
- Works for navigating plans and plans based on structural joins (shown here)

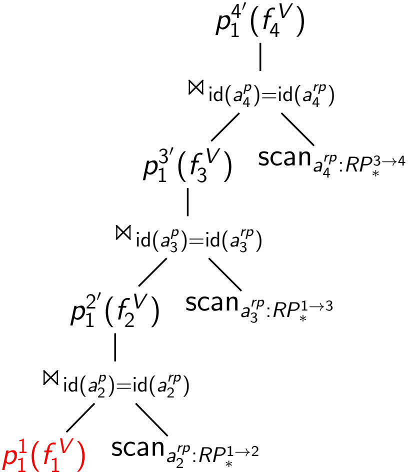
A Local Query Plan



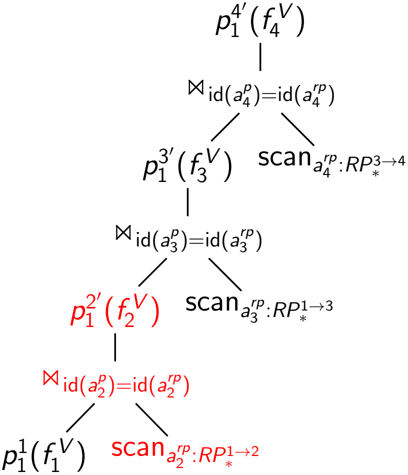
$p_1^2(f_2^V)$

- Plan scans root proxy nodes in fragment
- Idea: filter these root proxy nodes before evaluating remainder of plan
- Works for navigating plans and plans based on structural joins (shown here)

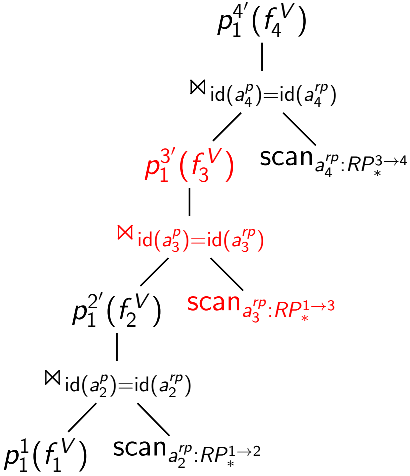
Pushing Cross-Fragment Joins



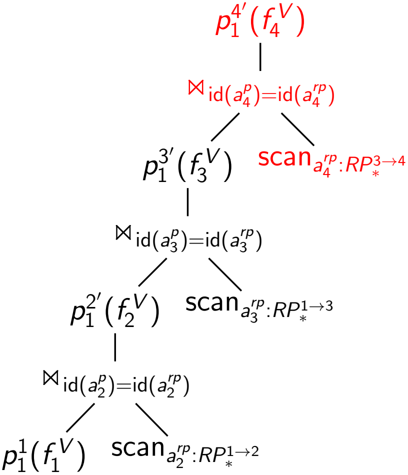
Pushing Cross-Fragment Joins



Pushing Cross-Fragment Joins



Pushing Cross-Fragment Joins



Pushing Cross Fragment Joins: Implementation

- Can use full pipelining if both inputs to join are ordered
- Alternatively, can use index on root proxy nodes
- Full parallelism after first tuple received by local plan

Pushing Cross Fragment Joins

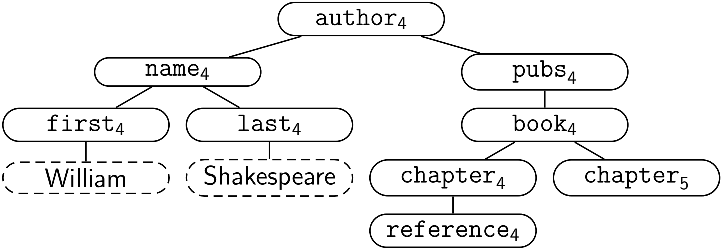
- Avoids accessing large portion of sub-trees within a fragment
- Can only be fully used in left-deep plans
- Decreases flexibility (e.g., where joins are performed)

Label Path Filtering

- Cross-fragment join pushing works well but decreases flexibility
- Goal: find a solution that can obtain partial benefit for scenarios where join pushing cannot be applied
- Idea: use selection instead of join to filter out some root proxy nodes

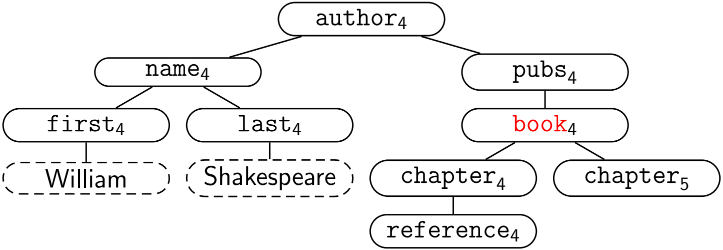
Label Path Filtering

- Assign to each proxy node the label path from the document root
- Filter for label paths that are compatible with the query



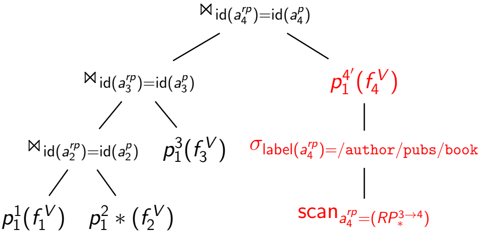
Label Path Filtering

- Assign to each proxy node the label path from the document root
- Filter for label paths that are compatible with the query



/author/pubs/book

Label Path Filtering



- Assume there are two types of publications: `book` and `article`
- Can use selection to filter chapters based on publication type

UNIVERSITY OF Waterloo

Label Path Filtering

- Can be used in more cases
- Retains higher degree of flexibility
- Benefit is more limited (does not filter all irrelevant root proxy nodes)

Determining the Best Distributed Execution Plan

- Join pushing and label path filtering are not always advantageous
- Determine best execution plan using cost model

Outline

Performance Evaluation

- Implemented techniques within Natix
- 12 GB XMark collection (auction data)
- 1 Amazon EC2 instance for each of each of 10 vertical fragments

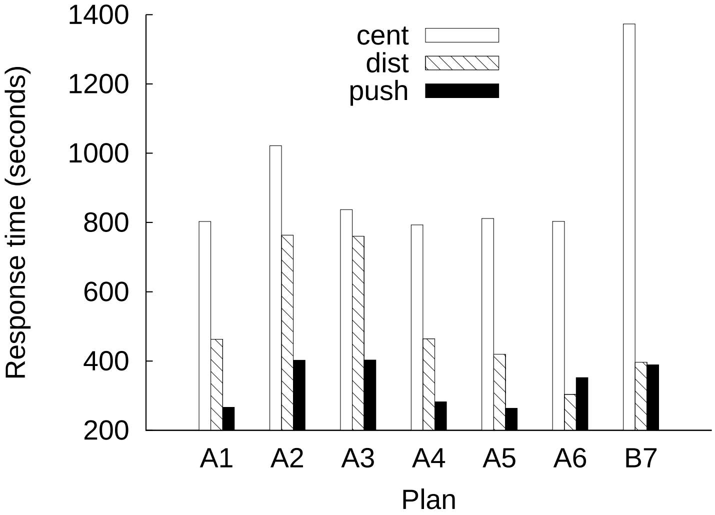
Performance Evaluation

- XPathMark queries (with few filtering value constraints)
- Modified, more selective XPathMark queries

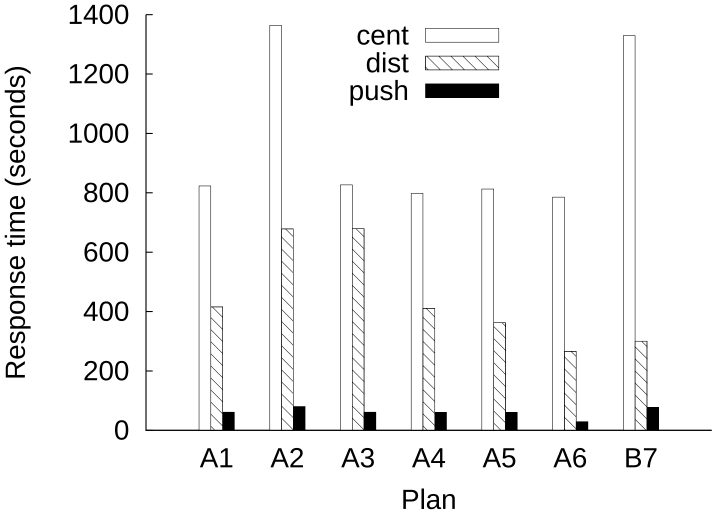
A1	/site/closed_auctions/closed_auction/annotation/description/text/keyword
A2	//closed_auction//keyword
A3	/site/closed_auctions/closed_auction//keyword
A4	/site/closed_auctions/closed_auction[annotation/description/text/keyword]/date
A5	/site/closed_auctions/closed_auction[descendant::keyword]/date
A6	/site/people/person[profile/gender and profile/age]/name
B7	//person[profile/@income]/name
A1S	/site/closed_auctions/closed_auction[price > 600]/annotation/description/text/keyword
A2S	//closed_auction[price > 600]//keyword
A3S	/site/closed_auctions/closed_auction[price > 600]//keyword
A4S	/site/closed_auctions/closed_auction[price > 600][annotation/description/text/keyword]/date
A5S	/site/closed_auctions/closed_auction[price > 600][descendant::keyword]/date
A6S	/site/people/person[starts-with(name, 'Ry')][profile/gender and profile/age]/name
B7S	//person[starts-with(name, 'Ry')][profile/@income]/name

University of Waterloo

Performance Evaluation: XPathMark



Performance Evaluation: Selective XPathMark



UNIVERSITY OF Waterloo

Conclusions

- Distribution can make XML query evaluation more scalable
- Join pushing can significantly improve query performance
- A cost model is essential for finding the optimal technique for a given query

UNIVERSITY OF
Waterloo

References

[1] Patrick Kling, M. Tamer Özsu, Khuzaima Daudjee: Generating Efficient Execution Plans for Vertically Partitioned XML Databases, PVLDB 2010.

[2] Patrick Kling, M. Tamer Özsu, Khuzaima Daudjee: Scaling XML Query Processing: Distribution, Localization and Pruning, DAPD 2011.