Efficient Evaluation of Forward XPath Axes over XML Streams

Abdul Nizar M.
P. Sreenivasa Kumar
Indian Institute of Technology Madras
INDIA
Querying Streamed XML

- Limited time of interest
- data is not stored persistently
- Content- and structure-based matching
- Issues – Efficiency and Scalability
- Applications
  - Subscribing to real-time news
  - Monitoring Stock Market data
XML

- Modelled as a node labelled tree
  - Internal (Element) nodes
  - Leaf (Data) nodes
  - Textual Order ≡ Pre-order ≡ Document Order

```
<AddrBook>
  <person>
    <name>
      <fname>fred</fname>
      <lname>john</lname>
    </name>
    <age>30</age>
  </person>
</AddrBook>
```
**XPath**

- Core language to express queries on XML data
- Path expressions specifying navigations through the XML tree
  - **Axis Steps** – direction of navigation on the XML Tree
  - **Predicates** – boolean /existential condition

```
/descendant::person /child::fname
```

```
/descendant::name[/child::fname = “fred”] /child::lname
```
Twigs

- Model XPath queries with \textit{child} and \textit{descendant} axes

\[
/\text{descendant::person} /\text{child::fname}
\]

\[
/\text{descendant::name}[/\text{child::fname} = \text{“fred”}]
/\text{child::lname}
\]
Ordered vs Unordered Axes

- **Unordered Axes**
  - child, descendant – forward axes
  - parent, ancestor - backward axes

- **Ordered Axes**
  - following-sibling, following – forward axes
  - preceding-sibling, preceding – backward axes
  - a/following::b – b-nodes that appear after an a-node but *not descendant* of a
  - a/preceding::b – b-nodes that appear before an a-node but *not ancestor* of a
Queries using Ordered Axes

- “The actions taken after it was found that there is a lesion in the bowel”
  
  /desc::observation[.=“lesion in the bowel”]/following::action

- figures that appear in the paper after the section titled “Motivation”
  
  /desc::section[name= “Motivation”]/following::figure

- sections that appear before the section titled “Datasets” excluding sections in which the “Datasets” section is nested in.
  
  /desc::section[name = “Datasets”]/preceding::section
Existing XML Stream querying algorithms

- Process XPATH with *child* and *descendant* axes
  - XSQ (Peng, S. Chavathe, ACMToDS 2005)
  - AFILTER (Candan, Chen, Agarwal VLDB 2006)
  - TwigM (Chen, Susan, Zheng ICDE 2006)
  - Branch Sequences (Aneesh and Sreenivasa Kumar ICDE 2007)
  - LazyQ and EagerQ (Gou and Chirkova SIGMOD 2007)
Existing XML Stream querying algorithms

- Turbo-XPath (Josifovsky et al., VLDBJ 2005)
  - Processes **ancestor** and **parent** axes

- SPEX (Olteanu, TKDE 2007)
  - Processes **ordered** axes
  - Uses transducer-based approach
Twig-based stream querying Systems are good

- The twig structure closely matches the tree representation of XML
- Effective use of stacks
  - Quite suitable for recursive documents
Twigs lack order semantics

- Twig-based systems can not be directly extended to handle ordered axes
  - The twig structure does not carry any order information
Twigs lack order semantics

- Twig-based systems can not be directly extended to handle ordered axes
  - The twig structure does not carry any order information
Twigs lack order semantics

- Twig-based systems cannot be directly extended to handle ordered axes
  - The twig structure does not carry any order information
Twigs lack order semantics

- Twig-based systems cannot be directly extended to handle ordered axes
  - The twig structure does not carry any order information
Twigs lack order semantics

- Twig-based systems cannot be directly extended to handle ordered axes
  - The twig structure does not carry any order information
Twigs lack order semantics

- Twig-based systems cannot be directly extended to handle ordered axes
  - The twig structure does not carry any order information

**Solution**

*Order-aware Twigs (OaTs)*
OaTs - Adding order semantics to Twigs

- Order Constraints
  - SLR Ordering – *following-sibling* axis
  - LR Ordering – *following* axis
- Can represent *preceding* and *preceding-sibling* axes also
OaTs - Adding order semantics to Twigs

- Order Constraints
  - SLR Ordering – *following-sibling* axis
  - LR Ordering – *following* axis
- Can represent *preceding* and *preceding-sibling* axes also

```
/descendant ::b/child::c/following-sibling::d
```
OaTs - Adding order semantics to Twigs

- Order Constraints
  - SLR Ordering – following-sibling axis
  - LR Ordering – following axis
- Can represent preceding and preceding-sibling axes also

More examples

/child::a/descendant ::b/child::c/following::d
OaTs - Adding Order Semantics to Twigs

- Closure Edges
  - Effectively represent *following-sibling* axis that appears *immediately* after *descendant* axis

\[
\text{a/descendant ::b/following-sibling::d}
\]
OaTs - Adding order semantics to Twigs

- Closure Edges
  - Effectively represent following-sibling axis that appears immediately after descendant axis

\[
\text{a/descendant ::b/following-sibling::d}
\]
Transforming path expressions to OaTs

/desc::a/desc::b/child::c/fs::d/fl::f/desc:g
Solution for forward axes

- child
- descendant
- following
- following-sibling
Solution for forward axes

- Node types

\[ /\text{desc::a/}\text{desc::b/}\text{child::c/}\text{fs::d/}\text{fl::f/}\text{desc::g} \]

- Boolean node
- Result node
- Stem node
Solution for forward axes
## Stack Frame Structure

<table>
<thead>
<tr>
<th>levelNo</th>
<th>sPreNo</th>
<th>linkInfo</th>
<th>PNVECT</th>
</tr>
</thead>
</table>

- **levelNo (sPreNo):** level number (pre-order number) of a node in the stream represented by the frame.

- **PNVECT:** Vector of pre-order numbers
  - Vector Size = fan-out of the query node the stack belongs to

- **linkInfo:** pointer to the ‘PARENT’ Frame
Stack Frame Structure

Query Nodes

Parent frame for $F_1$ & $F_2$
Parent stack for $S_d$ & $S_c$
Query Processing

- The OaT is processed branch by branch in left-to-right order
  - A branch is *not* processed *until* the constraints on branch to the left of it are met
  - *Every branch except the right-most* branch is processed *only once*
- Responds to *open-tag* and *close-tag* events from a stream parser
Query Processing

- Open-Tag Handler
  - For a node $e$ with open-tag $<t1>$, push a frame to stack $S_{t1}$ if,
    1. The relationships between query nodes the path from OaT’s root to query node labelled $t1$ are satisfied by a sequence of frames in the corresponding stacks.
    2. The relationships have already been satisfied by the root-to-leaf paths appeared before this path.
Query Processing

- **Close-Tag Handler**
  - For a node $e$ with close-tag $<t1>$ and depth $l$, pop the top frame $F$ of stack $S_{tl}$ if $F.\text{levelNo} = l$

  - If $F.\text{PNVECT}$ is filled completely, update the parent frame in the parent stack.
Query Processing

/desc::a/desc::b/child::c/fs::d/fl::f/desc::g
Query Processing
Query Processing
Query Processing
Query Processing
Query Processing
Query Processing
Query Processing
Query Processing
Query Processing
Query Processing
Closure Node Handling

![Diagram of Closure Node Handling]
Closure Node Handling
Closure Node Handling

Self Frame

$S_f$  $[-,-]$ $S_g$  $S_h$
Closure Node Handling

Self Frame

$S_f$

$S_g$

$S_h$
Closure Node Handling

![Diagram of closure node handling with nodes labeled f, g, h, etc., and edges indicating connections and transitions between states S_f, S_g, S_h.]
Closure Node Handling
Closure Node Handling
Closure Node Handling
Closure Node Handling
Experiments

- The algorithm was implemented in Java under Linux OS on an Athlon 64 2GHz machine with 2 GB RAM
- Performance Tests on various datasets
  - SWISSPROT
  - TREEBANK
  - XMARK
- Compared with SPEX implementation available at spex.sourceforge.net
Experiments

- **Experiment 1: Time Performance**
  - Three Queries on each dataset
    - No wild cards No predicates (SQ1, TQ1, XQ1)
    - Wild cards and value predicates (SQ2, TQ2, XQ2)
    - Closure Edges (SQ3, TQ3, XQ3)
Experiments

- Scalability
  - Dataset size \((d)\): 10MB, 20MB, \ldots, 80MB
  - Query Size (number of axis steps - \(q\)): 5, 10, \ldots, 25
  - Document Size \((d)\) vs Time
    - Query sizes: 15, 25
  - Query Size \((q)\) vs Time
    - Document sizes: 40, 80
Experiments

- **Effect of following axis**
  - # following axes ($f$): 0, 2, 4, …, 10
    - Start by $//NP//NP$ (FL0)
    - Extended by adding $f\text{::NP}//NP$ in steps of 2 (FL2, FL4, …, FL10)

- **Effect of following-sibling axis**
  - # following-sibling axes ($s$): 1, 3, 5, …, 9
    - Start by adding $//NP//f\text{::}* $(FS1)
    - Extended by adding $//NP//f\text{::}* $ in steps of 2 (FS3, FS5, …, FS9)
Conclusions

- Forward axis processing based on OaT was found to be quite simple, elegant, efficient and scalable
  - Effective encoding ordering constraints
  - Effective use of the ordering constraints
    - Delayed branch processing
    - Non-redundant branch processing
    - Early flushing of unwanted frames
References


THANK YOU
Future directions

- Message broker system for shared evaluation of multiple queries
- Handling of Complex predicates
- Handling of reverse ordered axes
Adding Order Semantics to Twigs

- Order Constraints
  - SLR Ordering – following-sibling axis
  - LR Ordering – following axis
- Can represent preceding and preceding-sibling axes also

/child::a/descendant::b/child::c/following::d/
Adding Order Semantics to Twigs

- Order Constraints
  - SLR Ordering – *following-sibling* axis
  - LR Ordering – *following* axis
- Can represent *preceding* and *preceding-sibling* axes also

/child::a/descendant ::b[/child::c]/following::d
Results in Document Order

\[ n_1 n_2 n_3 n_4 \]
\[ n_1 n_2 n_3 \]
\[ n_1 n_3 \]
\[ n_1 n_3 n_5 n_6 \]
Transforming path expressions to OaTs

/desc::a/desc::b/child::c/fs::d[/desc::d]/fl::f/desc:g[/child::h]
Solution for forward axes

- XPath expressions with forward axes
  - child, descendant, following, following-sibling
  - Support for predicates with child and descendant, appearing immediately before following or following-sibling axes
Solution for forward axes

- Node types

![Diagram with nodes a through h, indicating node types: Boolean node, Result node, Stem node]
Solution for forward axes
Query Processing

Diagram showing a network of nodes and edges with labels and numbers.
Query Processing
Query Processing
Query Processing

Diagram showing a graph with nodes labeled "a" through "r" and arrows indicating relationships between nodes. The diagram includes a subgraph with nodes "c" to "e" and "g" to "h".

Nodes are connected by edges with numbers indicating the step or iteration in the query processing. The diagram illustrates the flow of data or queries through the nodes, with some nodes marked with specific values or constraints.

The overall structure suggests a process of query evaluation or data aggregation, with the diagram possibly representing a query processing algorithm or model.
Query Processing
Query Processing
Query Processing
Query Processing
Query Processing
Query Processing
Query Processing
Query Processing