XML and Databases

Lecture 8
Streaming Evaluation: how much memory do you need?

Sebastian Maneth
NICTA and UNSW

CSE@UNSW -- Semester 1, 2009

Small XPath Quiz

Can you give an expression that returns the last/first occurrence of each distinct price element?

```xml
<price>3</price>
<price>1</price>
<price>3</price>
<price>1</price>
<price>3</price>
<price>4</price>
<price>1</price>
<price>7</price>
</b>
```

Should return

```
<price>3</price>
<price>4</price>
<price>1</price>
<price>7</price>
```

Should return

```
<price>3</price>
<price>1</price>
<price>4</price>
<price>7</price>
```

What if we mean number-distinctness (not strings)?

Outline

1. Automaton Approach
2. Parallel Evaluation of Multiple Queries
3. Sizes of Automata
4. How to deal with Filters
5. Existing Systems for Streaming XPath Evaluation

Recall: Top-Down Evaluation of Simple Paths

- evaluate in one single pre-order traversal (using a stack)

```
<price>3.0</price>
<price>1</price>
<price>3.00</price>
<price>1</price>
<price>3</price>
<price>4</price>
<price>1.000</price>
<price>7</price>
</b>
```

Streaming Algorithm!

- No need to store the document!
- Can evaluate on SAX event stream.

BUT

Need output buffers, if subtrees of match nodes should be printed!
Recall: Top-Down Evaluation of Simple Paths

→ evaluate in one single pre-order traversal (using a stack)

If we print node-IDs, then no output buffers are needed!

True Streaming, with memory need proportional to height.

Streaming Algorithm!

→ No need to store the document!!

Can evaluate on SAX event stream.

Simple Path: //a_1/a_2/a_3/.../a_n

TIME: one pass through document tree.

SPACE: stack of query positions.

height is bounded by depth of document tree.

BUT

Need output buffers, if subtrees of match nodes should be printed!

Arbitrary Slash+Slashslash

→ evaluate in one single pre-order traversal (using a stack)

Arbitrary queries with //a//b//c

multiple: //

query match position: p = 3

[StartElement( a )] push(3)
[EndElement( a )] p = pop() = 3

no match stay in p=3!

[StartElement( a )] push(3)
[StartElement( c )] push(3)
[EndElement( c )] p = pop() = 3
[StartElement( b )] push(3)
[StartElement( c )] push(3)
...

no match stay in p=3!

Result node!
Mark it, and stay in p=3.

Output Node-ID Start copying to Output Buffer
1. Automaton Approach

Recall
Deterministic Automaton runs in
\rightarrow linear time and
\rightarrow constant space

(plus stack of states, if we run on paths of a tree)

\textbf{Same as before}

Jump back within /-sequence.
AT MOST to the beginning of the last / .
Use KMP within /-sequence.
For *'s: build several KMP-tables.
1. Automaton Approach

Recall
Deterministic Automaton runs in
- linear time and
- constant space
(plus stack of states, if we run
on paths of a tree)

Problem
If it is NOT an e here, then what to do??

E.g.,
ab cd e fg h

We should be in state X!!

Æ =?
Which other letters need to be considered?

c d x y

Æ for x ≠ c, not important what x is
Æ only c/c ≠ x matters

Æ for x ≠ e, not important what x is
Æ only c/c ≠ e matters
Which other letters need to be considered?

\[ Q1 \equiv \neg a/b/c \]

Advantage of automata:
\( \Rightarrow \) can be combined to evaluate MANY queries “in parallel”.

Questions
1. Which transition is WRONG?

\[ Q1 \equiv \neg a/b/c \]

Advantage of automata:
\( \Rightarrow \) can be combined to evaluate MANY queries “in parallel”.

Questions
1. Which transition is WRONG?
2. How many transitions are missing?

\[ Q1 \equiv \neg a/b/c \]

Advantage of automata:
\( \Rightarrow \) can be combined to evaluate MANY queries “in parallel”.

Questions
1. Which transition is WRONG?
2. How many transitions are missing?
Advantage of automata: can be combined to evaluate MANY queries “in parallel”.

\[ Q_1 = \text{a/b/c} \quad \text{and} \quad Q_2 = \text{a/c} \]

**Questions**

1. Which transition is WRONG?
2. How many transitions are missing?

To evaluate MANY queries “in parallel”.

Combined automaton:
\[ \text{SIZE} \leq \text{SIZE}(A1) \times \text{SIZE}(A2) \]
3. The Size of the DFA

Theorem [GMOS’02] The number of states in the DFA for one linear XPath expression P is at most:

\[ k + |P| k s^m \]

- \( k \) = number of //
- \( s \) = size of the alphabet (number of tags)
- \( m \) = max number of * between two consecutive //

How to deal with filters?

1. Size of largest documents that can be streamed in this way depends on:
   - filters,
   - sizes of (pre) selected trees,
   - quality of (1), (2), etc..

2. Optimize:
   - Store potential match trees as DAGs
   - Release potential match trees as early as possible!
How to deal with filters?

Size of largest documents that can be streamed in this way depends on:
- filters,
- sizes of (pre) selected trees,
- quality of (1), (2), etc..

Release potential match trees as early as possible!

Find earliest point at which we know the filter is true.

Must be stored in memory

Size of largest documents that can be streamed in this way depends on:
- filters,
- sizes of (pre) selected trees,
- quality of (1), (2), etc..

Release potential match trees as early as possible!

Find earliest point at which we know the filter is true.

No need to store. Stream!

How to deal with filters?

Harder for Boolean combinations:
\[
\text{not}(./d/e) \text{ and } (./c/d \text{ or } //b/c)
\]

Question: where is the earliest point for this filter?

No need to store. Stream!

We can also construct automata for filter expressions!

Use a push-down for potential candidates.

Push-Down Automaton can probably be designed so that it pops/outputs candidates as early as possible.

Another idea

Use 2-pass algorithm: first (bottom-up) phase to mark subtrees with filter information.
Second (top-down) phase to determine match nodes.

Why is this interesting?

- Fast main memory evaluation
- Use disk as intermediate store (stream twice)
5. Streaming XPath Algorithms

- XFilter and YFilter [Altinel and Franklin 00] [Diao et al 02]
- X-scan [Ives, Levy, and Weld 00]
- XMLTK [Avila-Campillo et al 02]
- XTris [Chan et al 02]
- SPEX [Olteanu, Kiesling, and Bry 03]
- Lazy DFAs [Green et al 03]
- The XPush Machine [Gupta and Suciu 03]
- XSQ [Peng and Chawathe 03]
- TurboXPath [Josifovski, Fontoura, and Barta 04]
- ...

Some following slides are by T. Amagasa and M Onizuka (Japan)
See http://www.dasfaa07.ait.ac.th/DASFAA2007_tutorial3_1.pdf

Most of the following slides are by Dan Suciu (the above slides are actually also based on Suciu's slides ☺)
**XFilter (cont.)**  
NFA, view class: //tag

Decomposing XPath Query

![Diagram of XPath query decomposition](image1)

**YFilter**  
NFA, view class: XP{/../,*}

- Prefix sharing
- Predicates are processed by labels

![Diagram of XPath and NFA](image2)

**Shared data structure**

- Sharing identical structures among query trees
- What to share: node-test, simple path, branch, etc.

<table>
<thead>
<tr>
<th>What to share?</th>
<th>View class</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-test</td>
<td>(tag)</td>
<td>XPath (YFilter)</td>
</tr>
<tr>
<td>simple sub-path</td>
<td>(tag)/. .../tag</td>
<td>XTree (nodest)</td>
</tr>
<tr>
<td>simple path</td>
<td>XP(/.../*)</td>
<td>YFilter (nodest), Lazy DFA (nodeset), Prefix Filters (constant YFilter)</td>
</tr>
<tr>
<td>branch</td>
<td>XP([/*]/...)</td>
<td>XPath machine (structures)</td>
</tr>
</tbody>
</table>

**XPath Processing with FA**  
From XPath (XP{[/*].../*}) to NFA

![Diagram of XPath processing with FA](image3)

**NFA-based XPE Processing**

![Diagram of NFA-based XPE processing](image4)
Basic NFA Evaluation

Properties:
- Space = linear
- Throughput = decreases linearly

Systems:
- XFilter [Altinel&Franklin’99], YFilter.
- XTree [Chan et al.’02]

Basic DFA Evaluation

Properties:
- Throughput = constant!
- Space = GOOD QUESTION

System:
- XML Toolkit [University of Washington]
  http://xmltk.sourceforge.net

The Size of the DFA

Theorem [GMOS’02] The number of states in the DFA for one linear XPath expression P is at most:

\[ k + |P| \cdot k \cdot s^m \]

where:
- \( k \) is the number of //
- \( s \) is the size of the alphabet (number of tags)
- \( m \) is the maximum number of * between two consecutive //

Solution:
Compute the DFA Lazily

- Also used in text searching
- But will it work for 10^6 XPath expressions?
  - YES!
- For XPath it is provably effective, for two reasons:
  - XML data is not very deep
  - The nesting structure in XML data tends to be predictable

Size of DFA: Multiple Expressions

100 expressions

2^{100} states!!

There is a theorem here too, but it’s not useful…
Lazy DFA and “Simple” DTDs

- Document Type Definition (DTD)
  - Part of the XML standard
  - Will be replaced by XML Schema
- Example DTD:

```
<!ELEMENT document (section*)>
<!ELEMENT section ((section|abstract|table|figure)*)>
<!ELEMENT figure (table?,footnote*)>
```

Definition: A DTD is simple if all cycles are loops

**Lazy DFA and “Simple” DTDs**

- Simple DTD:
  - XPath expressions
  - Eager DFA “remembers” $2^d$ sets
  - Lazy DFA “remembers” only 4 sets

**Theorem** [GMOS’02] If the XML data has a “simple” DTD, then lazy DFA has at most:

$$1 + D(1+n)^d$$

states.

- n = max depths of XPath expressions
- D = size of the “unfolded” DTD
- d = max depths of self-loops in the DTD

**Fact of life:** “Data-like” XML has simple DTDs

**Lazy DFA and Data Guides**

- “Non-simple” DTDs are useless for the lazy DFA
- “Everything may contain everything”

**Definition** [Goldman&Widom’97]

The data guide for an XML data instance is the Trie of all its root-to-leaf paths

**Fact of life:** “Text”-like XML has non-simple DTDs
Fact of life: real XML data has “small” data guide [Liefke&S.’00]

**Theorem** [GMOS’02] If the XML data has a data guide with G nodes, then the number of states in the lazy DFA is at most:

\[ 1 + G \]

G = number of nodes in the data guide