Pushing XPath Accelerator to its Limits

Christian Grün, Marc Kramis
Alexander Holupirek, Marc H. Scholl, Marcel Waldvogel

Department of Computer and Information Science
University of Konstanz
Overview

...processing large XML documents

...our two prototypes

...a benchmark framework

...performance results

...what we will do next
Motivation

Observation

• sizes of XML instances are continuously growing:
  ▪ Library Data, U Konstanz: 2 GB
  ▪ DBLP: 300 MB
  ▪ Wikipedia: 5 GB up to 500 GB
  ▪ Log files > 10 GB ...

Fact

• XML processors needed to handle these documents
• current XML processors usually fail:
  ▪ by design (on-the-fly parsing, 2GB limit, indexing overhead, ...)
  ▪ by technical limits (main memory barrier, swapping, ...)
Motivation

MonetDB/XQuery

- based on the Pathfinder project, developed in Konstanz
- XPath Accelerator: relational XML encoding
- StairCase Join: very efficient path traversal
- Loop-Lifting: linear execution of nested loops

Identified Bottlenecks (Challenges...)

- main memory limitation
- no content/value indexes
Motivation

Two Approaches

BaseX
shrink main memory representation
→ pure main memory processing
→ compressed representation
  of XPath Accelerator encoding
→ introduction of an inherent value index

Idex
optimize disk layout
persistent native XML storage ←
constant scalability ←
logarithmic updateability ←
# BaseX – Memory Architecture

## Node Table Representation

<table>
<thead>
<tr>
<th>Pre</th>
<th>Par</th>
<th>Tag</th>
<th>Content</th>
<th>Kind</th>
<th>AttName</th>
<th>AttVal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>db</td>
<td></td>
<td>elem</td>
<td>id</td>
<td>add0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>address</td>
<td></td>
<td>elem</td>
<td>title</td>
<td>Prof.</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>name</td>
<td>Hack Hacklinson</td>
<td>text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>street</td>
<td>Alley Road 43</td>
<td>text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>city</td>
<td>Chicago, IL 60611</td>
<td>text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>address</td>
<td>Jack Johnson</td>
<td>text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>name</td>
<td>Jack Johnson</td>
<td>text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>street</td>
<td>Pick St. 43</td>
<td>text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>city</td>
<td>Phoenix, AZ 85043</td>
<td>text</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parent (32 bit)</th>
<th>Kind/Token (1/31 bit)</th>
<th>Attributes (10/22 bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...0000</td>
<td>0....0000</td>
<td>nil</td>
</tr>
<tr>
<td>...0001</td>
<td>0....0001</td>
<td>0000...0000</td>
</tr>
<tr>
<td>...0010</td>
<td>0....0010</td>
<td>0001...0001</td>
</tr>
<tr>
<td>...0011</td>
<td>1....0000</td>
<td>nil</td>
</tr>
<tr>
<td>...0010</td>
<td>0....0100</td>
<td>nil</td>
</tr>
<tr>
<td>...0011</td>
<td>1....0010</td>
<td>nil</td>
</tr>
<tr>
<td>...0001</td>
<td>0....0001</td>
<td>0000...0010</td>
</tr>
<tr>
<td>...0010</td>
<td>0....0010</td>
<td>nil</td>
</tr>
</tbody>
</table>

- **index storage**
- **numeric references**
BaseX – Querying

Value Indexing

- Text and AttributeValue indexes are extended by references to Pre values (⇒ inverted index)
- Small memory overhead (12 – 18%)

Query Optimization:

- Predicates are evaluated first (selection pushdown)
- Internal index axis and cs() kind test are added for predicate evaluation
- Queries are inverted & rewritten

Example:

/\text{db/address[@id = "add0"]}/\text{name}
⇒ index::node()[@id = "add0"]/parent::\text{address}[parent::db/parent::cs()]/child::name
Idefix – Data Structures

Concept

Shredding

Block Storage
Perfidix — Java Benchmarking Framework

Task
• automate tedious manual benchmarking tasks
  ▪ generic
  ▪ à la JUnit
  ▪ integration (Eclipse, Ant, ...)

Output
• console or XML per benchmark ($n$ runs)
  ▪ minimum, maximum, average, standard deviation, confidence interval 95

Discussion
• Java memory management
• benchmark history
Perfidix – Java Benchmarking Framework (cont.)

**Code Example**

```java
public class DemoBench extends Benchmarkable {
    public DemoBench() {...} // one-time initialization
    public setUp() {...} // per-run & method preparation
    public tearDown() {...} // per-run & method cleanup
    public benchFoo() {...} // method Foo to bench
    public benchBar() {...} // method Bar to bench
}
```

**Output Example**

<table>
<thead>
<tr>
<th></th>
<th>unit</th>
<th>sum</th>
<th>min</th>
<th>max</th>
<th>avg</th>
<th>stddev</th>
<th>conf95</th>
<th>runs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ns</td>
<td>8023000</td>
<td>19000</td>
<td>3822000</td>
<td>80230.00</td>
<td>376167.54</td>
<td>[6501.16, 153958.84]</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>ns</td>
<td>3951000</td>
<td>15000</td>
<td>778000</td>
<td>39510.00</td>
<td>74585.05</td>
<td>[24891.33, 54128.67]</td>
<td>100</td>
</tr>
<tr>
<td>TOTAL</td>
<td>ns</td>
<td>11974000</td>
<td>3951000</td>
<td>8023000</td>
<td>5987000.00</td>
<td>2036000.00</td>
<td>[3165247.96, 8808752.04]</td>
<td></td>
</tr>
</tbody>
</table>

---
Evaluation

Systems

- MonetDB & BaseX
  - main memory based processing
  - similar data structures
- X-Hive & Idefix
  - persistent disk storage
  - comparable scalability

Benchmark Queries

- XMark, 110 KB – 22 GB
- six value-oriented DBLP Queries, 300 MB
Evaluation – Scalability

XMark queries (x-axis ⇒ number of query, y-axis ⇒ execution time in sec.)
Evaluation – XMark

XMark queries (x-axis ⇒ number of query, y-axis ⇒ execution time in sec.)

- MonetDB
- BaseX
- X-Hive
- Idefix
Evaluation – DBLP

DBLP queries (x-axis: number of query, y-axis: execution time in sec.)

contains() function:
[1] /dblp/*[contains(title, 'XPath')]

range query:
[2] /dblp/*[year/text() < 1940]/title

exact predicate match:
[3] /dblp//inproceedings[contains(@key, '/edbt/')]
  [year/text() = 2004]
[6] //article[author/text() = 'Donald D. Chamberlin']
  [contains(title, 'XQuery')]
Lessons Learned

- hard-coded queries might blur evaluation results
- comparison troublesome with different systems
  - granularity of measurements (shredding, compilation, serialization, ...)
  - impact of different system components (storage, query)
  - availability of different features (updates, complete query implementation)
- handling of serialization output
- assure correctness of large results
- many factors to measure:
  - CPU load
  - memory I/O
  - disk I/O
  - memory consumption
Future Work

Merge BaseX & Idefix

- comprehensive support for value-based queries
- full text queries, including scoring algorithms (like SRA/INEX)
- optimize XML table compression
- optimize disk layout (hybrid, networked, and holographic storage)
- write Pathfinder plugin to support XQuery
- complete update implementation

Benchmarking

- use of virtual machines for benchmark reproducibility
- specify benchmark for XML updates
- application benchmark for eMail storage