Pathfinder/MonetDB: A High-Performance Relational Runtime for XQuery

Jan Rittinger
Database & Information Systems Group
University of Konstanz

03/16/2005
Pathfinder/MonetDB: A High-Performance Relational Runtime for XQuery

XQuery query → Pathfinder

Parser
Sem. Analysis
Core Translation
Typechecking
Simplification
Optimization
Core to MIL Translation

result

MonetDB
Pathfinder runtime module
MIL
Database
Pathfinder/MonetDB: A High-Performance Relational Runtime for XQuery

Jan Rittinger

Experimental Results

Join Recognition

Problems & Solutions

Introduction

Pathfinder/MonetDB (MIL and Output)

```
758 | iter_oidMap := nil_oid oid;
759 | iter := oid_oidMap.leftfetchjoin(outer001);
760 | oid_oidMap := nil_oid oid;
761 | # item := item;
762 | # kind := kind;
763 | } # end of mapBack ()
764 | # cleanUpLevel ()
765 | inner001 := nil_oid oid;
766 | outer001 := nil_oid oid;
767 | loop001 := nil oid oid;
768 | v_vid001 := nil oid oid;
769 | v_iter001 := nil oid oid;
770 | v_pos001 := nil oid oid;
771 | v_item001 := nil oid oid;
772 | v_kind001 := nil oid int;
773 | } # end of for-tranlation
774 | } # end of evaluateJoin
775 | { # translate fn:count (item*) as integer
776 | var iter_count := {count}{iter.reverse()}.loop001.reverse();
777 | iter_count := iter_count.reverse().mark(090).reverse();
778 | var ins_vals := iter_count.reverse().mark(nil).reverse();
779 | int_values := int_values.segbase(nil).insert(ins_vals).segbase(090);
780 | ins_vals := nil oid int;
781 | item := iter_count.leftjoin(int_values.reverse());
782 | item := item.reverse().mark(090).reverse();
783 | iter_count := nil oid int;
784 | iter := loop001.reverse().mark(090).reverse();
785 | kind := iter.project(INT);
786 | } # end of translate fn:count (item*) as integer
787 | print_result(wml, ws, item, kind, int_values, db1_values, dec_values, str_values);
```

<?xml version="1.0" encoding="utf-8"?>
<XQueryResult>
30
</XQueryResult>
13:49:15 rittinger@titan03:/local_tmp/rittinge/pathfinder/Linux>
**Conclusion** (Summer School GK - St. Vigil):

- almost every feature of XQuery is supported
- 18 of 20 Xmark queries are running with generated MIL code

but ...

- performance problems (mostly because of iterative translations)
  - solution: intelligent loop-lifted version
- a lot of work has still to be done (e.g. built-in functions, ...)

Challenges

• mapping makes extensive use of sort operators (mostly indirect by a row numbering operator)
• concepts have to be `loop lifted'
e.g. path steps, functions, …
(instead of simple values, sets of values for multiple iterations have to be managed)
• for-loop nesting implicitly creates cartesian products
Idea: sorting

- try to use only order preserving operations
  - some additional operators are needed
    (e.g., "merged union" instead of union)
- keep more order properties (secondary orderings)
  - only refine sorting
    - we can get rid of runtime order properties
example: How many people are living on each continent?

```xml
for $a in doc("earth.xml")//continent
    return count($a/country//*people)
```

<table>
<thead>
<tr>
<th>iter</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>n</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iter</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iter</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>m</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iter</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>p</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iter</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
</tr>
</tbody>
</table>

doc("earth.xml") //continent $a /country//*people //people
Idea: loop lifted path steps

- evaluate all iterations during one sequential scan over the document
- use stack with active iter values \(\Rightarrow\) multiple iterations
- maintain ideas of path step algorithms (staircase join - pruning, partitioning, skipping)
**Motivation: join recognition**

**Introduction**

**Pathfinder/MonetDB: A High-Performance Relational Runtime for XQuery**

Jan Rittinger

---

**Experimental Results**

**Join Recognition**

**Problems & Solutions**

---

- **Motivation: join recognition**

  independent for-loop execution

  \[
  \begin{align*}
  S_0 &\quad \text{for } a \text{ in } (1, 2, 3) \\
  &\quad \quad \text{for } b \text{ in } (30, 20, 10) \\
  S_1 &\quad \quad \text{where } 10 \times a = b \\
  S_2 &\quad \quad \text{return } 42
  \end{align*}
  \]

  \[
  \Rightarrow \text{for } a \text{ in } (1, 2, 3) \quad \text{for } b \text{ in } (30, 20, 10) \\
  \quad \text{return } 10 \times a \quad = \quad \text{return } b
  \]

  \[
  \Rightarrow \text{intermediate results stay linear}
  \]

  three steps:

  - join pattern
  - independent execution
  - join translation
Join pattern

for $v$ in $e_{in}$
  return if ($\rho(e_1,e_2)$) then $e_{return}$ else ()

for $a$ in (1,2,3)
  for $b$ in (30,20,10)
    where 10 * $a$ = $b$
    return 42

for $a$ in (1,2,3)
  for $b$ in (30,20,10)
    return if (=(10*$a,$b))
      then 42
      else ()

\[ e_{in}: (30,20,10) \]
\[ \rho: = \]
\[ e_1: 10*a \]
\[ e_2: b \]
\[ e_{return}: 42 \]
Conditions for independent execution

for $v$ in $e_{in}$
return if ($p(e_1, e_2)$) then $e_{return}$ else ()

three limitations:

• variable $v$ appears free in $e_2$ only

• variables occurring free in $e_2$ and $e_{in}$ are bound in any enclosing scope, except for the scope that directly encloses the pattern

• $p$ is a supported join predicate of the database

($w, ...$) scope$_0$
($x, ...$) scope$_1$
($z, ...$) $v$ $e_{in}$ scope$_n$

($e_1$ and $e_2$ may be arbitrarily swapped)
Test Set: Xmark Benchmark (11 MB to 11 GB)
Experimentation Platform: 1.6 GHz AMD Opteron 242 (1MB L2 cache), 8GB RAM

order awareness:
• speedup of factor 2
Experimental Results

Join Recognition Problems & Solutions Experimental Results

Pathfinder/MonetDB: A High-Performance Relational Runtime for XQuery

Jan Rittinger

loop lifting path steps:

• most queries are 3 to 10 times faster
• queries, where loop lifting is not necessary, pay for additional state keeping information (Q1, Q15)
join recognition:
- doesn't finish 100MB and above without join detection
- speedup shows that join recognition is required for reasonable execution times
Experimental results

Join Recognition

Problems & Solutions

Experimental Results

Pathfinder/MonetDB: A High-Performance Relational Runtime for XQuery

Jan Rittinger

Experimental results

scaling (normalized to 100 MB):

- system scales (almost) perfectly linear
- Q11, Q12 scale quadratically: intermediate results grows with the size of the Cartesian product ('<'-comparison)
## Experimental Results

### Pathfinder/MonetDB: A High-Performance Relational Runtime for XQuery

Jan Rittinger

<table>
<thead>
<tr>
<th>Operation</th>
<th>10 MB</th>
<th>100 MB</th>
<th>1 GB</th>
<th>10 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>selection</strong></td>
<td>Q</td>
<td>Galax</td>
<td>X-Hive</td>
<td>PF/M</td>
</tr>
<tr>
<td>1</td>
<td>0.07</td>
<td>0.37</td>
<td>0.05</td>
<td>0.72</td>
</tr>
<tr>
<td>2</td>
<td>0.03</td>
<td>0.45</td>
<td>0.07</td>
<td>0.31</td>
</tr>
<tr>
<td>3</td>
<td>0.14</td>
<td>0.65</td>
<td>0.29</td>
<td>1.76</td>
</tr>
<tr>
<td><strong>document order</strong></td>
<td>4</td>
<td>0.23</td>
<td>0.10</td>
<td>0.09</td>
</tr>
<tr>
<td><strong>casting</strong></td>
<td>5</td>
<td>0.05</td>
<td>0.13</td>
<td>0.05</td>
</tr>
<tr>
<td><strong>descendant steps, aggregation (count), +</strong></td>
<td>6</td>
<td>1.30</td>
<td>1.07</td>
<td>0.03</td>
</tr>
<tr>
<td><strong>join</strong></td>
<td>7</td>
<td>2.69</td>
<td>1.57</td>
<td>0.03</td>
</tr>
<tr>
<td><strong>nested joins</strong></td>
<td>8</td>
<td>0.17</td>
<td>0.85</td>
<td>0.14</td>
</tr>
<tr>
<td><strong>element construction</strong></td>
<td>9</td>
<td>113.24</td>
<td>32.25</td>
<td>0.20</td>
</tr>
<tr>
<td><strong>large (intermediate) results, join</strong></td>
<td>10</td>
<td>1.74</td>
<td>5.28</td>
<td>0.81</td>
</tr>
<tr>
<td><strong>reconstruction</strong></td>
<td>11</td>
<td>2.63</td>
<td>98.91</td>
<td>0.19</td>
</tr>
<tr>
<td><strong>full text search</strong></td>
<td>12</td>
<td>1.44</td>
<td>23.39</td>
<td>0.15</td>
</tr>
<tr>
<td><strong>long path (child steps)</strong></td>
<td>13</td>
<td>0.04</td>
<td>0.10</td>
<td>0.07</td>
</tr>
<tr>
<td><strong>empty</strong></td>
<td>14</td>
<td>1.93</td>
<td>0.72</td>
<td>0.26</td>
</tr>
<tr>
<td><strong>not, empty</strong></td>
<td>15</td>
<td>0.02</td>
<td>0.03</td>
<td>0.10</td>
</tr>
<tr>
<td><strong>empty</strong></td>
<td>16</td>
<td>0.03</td>
<td>0.03</td>
<td>0.11</td>
</tr>
<tr>
<td><strong>not, empty</strong></td>
<td>17</td>
<td>0.06</td>
<td>0.09</td>
<td>0.08</td>
</tr>
<tr>
<td><strong>UDF, function call</strong></td>
<td>18</td>
<td>0.08</td>
<td>0.08</td>
<td>0.05</td>
</tr>
<tr>
<td><strong>sort by</strong></td>
<td>19</td>
<td>1.17</td>
<td>0.67</td>
<td>0.12</td>
</tr>
<tr>
<td><strong>group by</strong></td>
<td>20</td>
<td>0.28</td>
<td>0.11</td>
<td>0.25</td>
</tr>
</tbody>
</table>
created a working system,
• which supports **arbitrary computations over XML input documents**
• which is **scalable** due to optimizations like order awareness, loop lifting, and join recognition

⇒ relational approach does work
⇒ Pathfinder/MonetDB is a promising alternative to other XQuery solutions

next steps:
• release in mid-april (under open source license)
• algebraic optimizations (e.g., join detection)