Fast Faceted Search in XML
Using XQuery and Indexes in eXist

Anne Schuth & Maarten Marx
University of Amsterdam
Motivation
Motivation

• Lots of data
Motivation

- Lots of data
- In XML
Motivation

- Lots of data
- In XML
- With metadata
Explore Data with Facets
Explore Data with Facets
Explore Data with Facets
In XQuery
In XQuery

- `let $hits := collection("/db/xmark")//item[facet:query(., "gold")]]`
In XQuery

- let $hits := collection("/db/xmark/")//item[facet:query(., "gold")]
- let $counts := facet:counts($hits)
In XQuery

- `let $hits := collection("/db/xmark/\node{item}[facet:query(., "gold")])`
- `let $counts := facet:counts($hits)`

```xml
<facets>
  <facet name="location">
    <value name="United States">1103</value>
    <value name="Barbados">5</value>
    <value name="Gabon">4</value>
    <value name="Gambia">4</value>
    <value name="Palau">4</value>
    <value name="Cape Verde">3</value>
    <value name="Grenada">3</value>
    <value name="Myanmar">3</value>
    <value name="Namibia">3</value>
    <value name="New Zealand">3</value>
  </facet>
  <facet name="quantity">
    <value name="1">1355</value>
    <value name="2">116</value>
    <value name="3">5</value>
    <value name="4">1</value>
  </facet>
  <facet name="featured">
    <value name="yes">159</value>
  </facet>
</facets>
```
In XQuery

• let $hits := collection("/db/xmark")/item[facet:query(., "gold")]

• let $counts := facet:counts($hits)
In XQuery

- \( \text{let } $\text{hits} := \text{collection("/db/xmark/")//item[facet:query(., "gold")]} \)
- \( \text{let } $\text{counts} := \text{facet:counts($\text{hits})} \)
- \( \text{let } $\text{filtered} := $\text{hits[facet:filter(., ("location", "barbados"))]} \)
In XQuery

- let $hits := collection("/db/xmark")//item[facet:query(., "gold")]
- let $counts := facet:counts($hits)
- let $filtered := $hits[facet:filter(., ("location", "barbados"))]
- let $added := facet:add($filtered, "user", "gold-barbados")
<collection xmlns="http://exist-db.org/collection-config/1.0">
  <index>
    <fulltext default="none" attributes="no"/>
    <bobo>
      <text qname="item">
        <!-- contents here -->
      </text>
    </bobo>
    <create path="//location" type="xs:string"/>
    <create path="//quantity" type="xs:string"/>
    <create path="//@featured" type="xs:string"/>
  </index>
</collection>
<collection xmlns="http://exist-db.org/collection-config/1.0">
  <index>
    <fulltext default="none" attributes="no"/>
    <bobo>
      <text qname="item">
        <facet name="location" select="location"/>
        <facet name="quantity" select="quantity"/>
        <facet name="featured" select="@featured"/>
        <facet name="category" select="incategory/@category" type="multi"/>
        <facet name="date" select="//metadata/date"/>
      </text>
    </bobo>
    <create path="//location" type="xs:string"/>
    <create path="//quantity" type="xs:string"/>
    <create path="//@featured" type="xs:string"/>
  </index>
</collection>
Indexing
Indexing

• Just like Lucene Indexer
Indexing

- Just like Lucene Indexer
- Supporting root selects: “//metadata”
Indexing

• Just like Lucene Indexer
• Supporting root selects: “//metadata”
• Facet values are dictated by data
Indexing

• Just like Lucene Indexer
• Supporting root selects: “//metadata”
• Facet values are dictated by data
• Normalization (none as of yet)
Using Bobo-Browse (1)

- eXist
- Lucene
Using Bobo-Browse (1)
Using Bobo-Browse (2)

• A Faceted Search Implementation, an extension of Apache Lucene
Using Bobo-Browse (2)

- A Faceted Search Implementation, an extension of Apache Lucene
- Developed by and powering LinkedIn People Search
Using Bobo-Browse (2)

- A Faceted Search Implementation, an extension of Apache Lucene
- Developed by and powering LinkedIn People Search
  - No need for cache warm-up for the system to perform.
Using Bobo-Browse (2)

• A Faceted Search Implementation, an extension of Apache Lucene

• Developed by and powering LinkedIn People Search
  • No need for cache warm-up for the system to perform.
  • Multi value sort - sort documents on fields that have multiple values per doc, e.g tokenized fields
Using Bobo-Browse (2)

• A Faceted Search Implementation, an extension of Apache Lucene

• Developed by and powering LinkedIn People Search
  • No need for cache warm-up for the system to perform.
  • Multi value sort - sort documents on fields that have multiple values per doc, e.g. tokenized fields
  • Fast field value retrieval - over 30x faster than `IndexReader.document(int docid)`
Using Bobo-Browse (2)

- A Faceted Search Implementation, an extension of Apache Lucene
- Developed by and powering LinkedIn People Search
  - No need for cache warm-up for the system to perform.
  - Multi value sort - sort documents on fields that have multiple values per doc, e.g tokenized fields
  - Fast field value retrieval - over 30x faster than `IndexReader.document(int docid)`
  - Facet count distribution analysis
Using Bobo-Browse (2)

- A Faceted Search Implementation, an extension of Apache Lucene
- Developed by and powering LinkedIn People Search

- No need for cache warm-up for the system to perform.
- Multi value sort - sort documents on fields that have multiple values per doc, e.g. tokenized fields
- Fast field value retrieval - over 30x faster than `IndexReader.document(int docid)`
- Facet count distribution analysis
- Stable and small memory footprint
Using Bobo-Browse (2)

- A Faceted Search Implementation, an extension of Apache Lucene
- Developed by and powering LinkedIn People Search
  - No need for cache warm-up for the system to perform.
  - Multi value sort - sort documents on fields that have multiple values per doc, e.g. tokenized fields
  - Fast field value retrieval - over 30x faster than `IndexReader.document(int docid)`
  - Facet count distribution analysis
  - Stable and small memory footprint
  - Support for runtime faceting
Using Bobo-Browse (2)

• A Faceted Search Implementation, an extension of Apache Lucene

• Developed by and powering LinkedIn People Search

  • No need for cache warm-up for the system to perform.
  • Multi value sort - sort documents on fields that have multiple values per doc, e.g tokenized fields
  • Fast field value retrieval - over 30x faster than `IndexReader.document(int docid)`
  • Facet count distribution analysis
  • Stable and small memory footprint
  • Support for runtime faceting
  • Result merge library for distributed facet search
Using Bobo-Browse (2)

- A Faceted Search Implementation, an extension of Apache Lucene

- Developed by and powering LinkedIn People Search

  - No need for cache warm-up for the system to perform.
  - Multi value sort - sort documents on fields that have multiple values per doc, e.g tokenized fields
  - Fast field value retrieval - over 30x faster than `IndexReader.document(int docid)`
  - Facet count distribution analysis
  - Stable and small memory footprint
  - Support for runtime faceting
  - Result merge library for distributed facet search
  - ...
Fast Faceted Search

<table>
<thead>
<tr>
<th>Doc</th>
<th>location</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc 1</td>
<td>&lt;location&gt;Prague&lt;/location&gt;</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>Doc 2</td>
<td>&lt;location&gt;Amsterdam&lt;/location&gt;</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>Doc 3</td>
<td>&lt;location&gt;Prague&lt;/location&gt;</td>
<td>1 0 0 1</td>
</tr>
</tbody>
</table>
Fast Faceted Search

- Bit vectors

<table>
<thead>
<tr>
<th>Doc</th>
<th>Location</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc 1</td>
<td>&lt;location&gt;Prague&lt;/location&gt;</td>
<td>1</td>
</tr>
<tr>
<td>Doc 2</td>
<td>&lt;location&gt;Amsterdam&lt;/location&gt;</td>
<td>0</td>
</tr>
<tr>
<td>Doc 3</td>
<td>&lt;location&gt;Prague&lt;/location&gt;</td>
<td>1</td>
</tr>
</tbody>
</table>
Fast Faceted Search

- Bit vectors
- Document order

<table>
<thead>
<tr>
<th></th>
<th>location</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc 1</td>
<td>&lt;location&gt;Prague&lt;/location&gt; &lt;quantity&gt;1&lt;/quantity&gt;</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>Doc 2</td>
<td>&lt;location&gt;Amsterdam&lt;/location&gt; &lt;quantity&gt;2&lt;/quantity&gt;</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>Doc 3</td>
<td>&lt;location&gt;Prague&lt;/location&gt; &lt;quantity&gt;2&lt;/quantity&gt;</td>
<td>1 0 0 1</td>
</tr>
</tbody>
</table>
Fast Faceted Search

- Bit vectors
  - Document order
  - Filter

<table>
<thead>
<tr>
<th></th>
<th>location</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc 1</td>
<td>&lt;location&gt;Prague&lt;/location&gt; &lt;quantity&gt;1&lt;/quantity&gt;</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>Doc 2</td>
<td>&lt;location&gt;Amsterdam&lt;/location&gt; &lt;quantity&gt;2&lt;/quantity&gt;</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>Doc 3</td>
<td>&lt;location&gt;Prague&lt;/location&gt; &lt;quantity&gt;2&lt;/quantity&gt;</td>
<td>1 0 0 1</td>
</tr>
</tbody>
</table>
Fast Faceted Search

• Bit vectors
  • Document order
  • Filter
  • Re-index on updates

<table>
<thead>
<tr>
<th></th>
<th>location</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc 1</td>
<td>&lt;location&gt;Prague&lt;/location&gt;&lt;quantity&gt;1&lt;/quantity&gt;</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>Doc 2</td>
<td>&lt;location&gt;Amsterdam&lt;/location&gt;&lt;quantity&gt;2&lt;/quantity&gt;</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>Doc 3</td>
<td>&lt;location&gt;Prague&lt;/location&gt;&lt;quantity&gt;2&lt;/quantity&gt;</td>
<td>1 0 0 1</td>
</tr>
</tbody>
</table>
Fast Faceted Search

• Bit vectors
  • Document order
  • Filter
  • Re-index on updates
Fast Faceted Search

• Bit vectors
  • Document order
  • Filter
  • Re-index on updates

• Boolean AND operation
Fast Faceted Search

• Bit vectors
  • Document order
  • Filter
  • Re-index on updates

• Boolean AND operation
  • Fast implementations
Fast Faceted Search

- Bit vectors
  - Document order
  - Filter
  - Re-index on updates
- Boolean AND operation
  - Fast implementations
- Cardinality calculations
Fast Faceted Search

- Bit vectors
  - Document order
  - Filter
  - Re-index on updates
- Boolean AND operation
  - Fast implementations
- Cardinality calculations
  - Fast implementations
Benchmark (I)

- XMark


<table>
<thead>
<tr>
<th>XMark scaling</th>
<th>Size (MB)</th>
<th># Files</th>
<th># items</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>12</td>
<td>69</td>
<td>2175</td>
</tr>
<tr>
<td>0.2</td>
<td>24</td>
<td>136</td>
<td>4350</td>
</tr>
<tr>
<td>0.4</td>
<td>46</td>
<td>272</td>
<td>8700</td>
</tr>
<tr>
<td>0.8</td>
<td>92</td>
<td>542</td>
<td>17400</td>
</tr>
<tr>
<td>1.6</td>
<td>184</td>
<td>1084</td>
<td>34800</td>
</tr>
<tr>
<td>3.2</td>
<td>368</td>
<td>2166</td>
<td>69600</td>
</tr>
<tr>
<td>6.4</td>
<td>735</td>
<td>4330</td>
<td>139200</td>
</tr>
<tr>
<td>12.8</td>
<td>1500</td>
<td>8660</td>
<td>278400</td>
</tr>
</tbody>
</table>
Benchmark (2)
Benchmark (2)

- Three Facets
  - Location  up to 232 values
  - Quantity  up to 7 values
  - Featured  2 values
Benchmark (2)

• Three Facets
  • Location  up to 232 values
  • Quantity  up to 7 values
  • Featured  2 values

• Two Scenarios
Benchmark (2)

- **Three Facets**
  - Location up to 232 values
  - Quantity up to 7 values
  - Featured 2 values

- **Two Scenarios**
Compared To
Compared To

- Naive Method
Compared To

• Naive Method

declare function local:counts($hits){
    let $facets := ("$hits/location", "$hits/quantity", "$hits/@featured")
    let $labels := ("locations", "quantities", "featured")
    return <facets >
    {for $facet at $p in $facets
        let $facetvalues := util:eval($facet)
        return <facet name="{$labels[$p]}">
            {for $a in distinct-values($facetvalues)
                return <value name="{$a}">
                    {count($facetvalues. eq $a)}
                </value>}
        </facet>
    }
</facets>
Compared To

- **Naive Method**

```javascript
declare function local:counts($hits){
  let $facets := ("$hits/location", "$hits/quantity", "$hits/@featured")
  let $labels := ("locations", "quantities", "featured")
  return <facets >
    {for $facet at $p in $facets
      let $facetvalues := util:eval($facet)
      return <facet name="{$labels[$p]}">
        {for $a in distinct-values($facetvalues)
          return <value name="{$a}">
            {count($facetvalues[. eq $a])}
          </value>
        </facet>
    </facets>}
};
```

- **Ron van den Branden Method**
Compared To

• **Naive Method**

```xml
declare function local:counts($hits){
  let $facets := ("$hits/location", "$hits/quantity", "$hits/@featured")
  let $labels := ("locations", "quantities", "featured")
  return
  <facets >
    { for $facet at $p in $facets
      let $facetvalues := util:eval($facet)
      return
        <facet name="{$labels[$p]}">
          { for $a in distinct-values($facetvalues)
            return
              <value name="{$a}">
                { count($facetvalues. eq $a)}
              </value>
            }
        </facet>
      }
    </facets>
};
```

• **Ron van den Branden Method**

```xml
declare function local:cb($term, $data){
  <value name="{$term}">
    {$data[1]}
  </value>
};
```

```xml
declare function local:counts($hits){
  let $cb := util:function(xs:QName("local:cb"), 2)
  let $facets := ("$hits/location", "$hits/quantity", "$hits/@featured")
  let $labels := ("locations", "quantities", "featured")
  return
  <facets >
    { for $facet at $p in $facets
      let $vals := util:eval($facet)
      return
        <facet name="{$labels[$p]}">
          { util:index-keys($vals, "", $cb, 10000)}
        </facet>
      }
    </facets>
};
```
Large Scenario

Figure 4: Processing time in seconds per step of Scenario A, for different datasizes, see Table 1 (p. 6). Figures are for the Naive Method, Rvdb Method and the Indexed Method. Note the log scale of the vertical axis.

The development of the faceted search micro-benchmark based on XMark and its application using XCheck proved useful. The plots give quick insights in the scalability of the algorithms, possibilities for improvements and anomalies in the code. A future version of the Benchmark could include a way to measure scalability over the number of facet-value pairs.

6.1 Future Work

There are two obvious starting points for improvements: find the reason for the outlier in our experiment, as mentioned in Section 5.2 (p. 10). Secondly, as mentioned in Section 4.3.4 (p. 10), we should implement a solution like Wang (2009) for a mapping from position-situation vector to identifier, probably by integrating Zoie.

Besides improvements in our implementation of the Indexed Method, we would like to extend our experiments. It would be insightful to investigate the behavior of the three methods when we vary the number of facet-values. Also, a more precise measurement of where exactly in which method time is spent would give a clear indication of where we could still gain something. And lastly, we did not mention the behavior of the methods under (heavy) updates of the dataset; we assumed a static dataset where documents are never removed or added. It is very likely that for such a scenario another method should be preferred.

Acknowledgements

Maarten Marx acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh
Tiny Scenario

Figure 5: Processing time in seconds per step of Scenario B, for different datasizes, see Table 1 (p. 6). Figures are for the Naive Method, RvdB Method and the Indexed Method. Note the log scale of the vertical axis.
Combined Results

Figure 3: The average processing time over all steps (from both scenarios) per method for a growing dataset. Note the log scale of the vertical axis.

Looking at Figure 3, we see that the Indexed Method performs worse than both other methods but remains near constant while the time taken by both the Naive Method and Rvdb Method is polynomial. For datasets larger than about 400MB it is preferable to use the Indexed Method. It should also be noted that an average processing time of about 1 second per query is generally not acceptable in a user interface.

Both Figure 4 (following page) and 5 (p. 13) illustrate how the first drill-down step—step 2 in each scenario—for the Indexed Method consequently takes a lot of time. While all other timings for that method stay below or around 0.2 seconds, step 2 goes up to 8 seconds. We are not able to pinpoint the cause of this behavior yet. We do, however, point out that when we solve this issue the method will become within a very acceptable range with respect to processing time.

If we leave our Indexed Method aside, it is interesting to see that the Rvdb Method outperforms the Naive Method method for the earlier steps and vice versa for the later steps.

Conclusions

We conclude that the bit-vector implementation is the only one which scales with the datasize; the processing time stays near constant. Such behavior is a very desirable one in many applications. Our average running time of around 1 second for all queries on all datasets (going up to almost 300K nodes or 1.5GB) is promising but not yet good enough.
Wrap up

- Expressive XQuery interface
- Extension to eXist database
- Intuitive Benchmark
- Reasonable results
Open Issues
Open Issues

- Normalization
Open Issues

- Normalization
- Caching
Open Issues

• Normalization

• Caching

• Zoie: realtime indexing and search system