Query Processing on Cubes Mapped from Ontologies to Dimension Hierarchies

Carlos Garcia-Alvarado  
*Greenplum EMC  
USA* 

Carlos Ordonez  
*University of Houston  
USA*
Scenario

- Computer Science
  - Parallel processors
    - Core Duo, Core 2 Quad, Itanium
  - Array and vector processors
    - PicoChip, IntellaSys MPPAs, ASOCS MPPAs
  - Distributed systems
    - Teradata, PWD, Greenplum

**Dimension** | **Measurements**
---|---
1 | $D_j$ | $A_1$
1 | Parallel Processors | 30
1 | Array and Vector Processors | 30
2 | Parallel Processors | 40
2 | Distributed databases | 40
3 | Array and Vector Processors | 50
3 | Distributed databases | 50
3 | Parallel Processors | 50
4 | Parallel Processors | 180
5 | Array and Vector Processors | 40
6 | Distributed databases | 310
7 | Distributed databases | 400
Dimension Hierarchies

Level 0
{Computer Science} 800

Level 1
{Distributed Systems, Multiprocessors} 90

{Distributed Systems} 340
{Multiprocessors} 800

Level 2
{Parallel processors, Array and vector processors, Distributed databases} 50

{Parallel processors, Array and vector processors} 80
{Parallel processors, Distributed databases} 90
{Array and vector processors, Distributed databases} 50

{Parallel processors} 300
{Array and vector processors} 120
{Distributed databases} 800
Problem

Efficient summarization of text corpora mapped from ontologies to dimension hierarchies.

OLAP Cube is an excellent candidate to represent concept hierarchies and perform efficient aggregations on multiple combinations.

Previous work required star schema or cubes on demand with all concepts [1][2][3].
Definitions

- Let a collection C, or corpus, of n documents.
- Each document has \{D_1, D_2, \ldots, D_k\} dimensions.
- Each document has a measurements \{A_1, A_2, \ldots, A_e\}
- Fact table is in vertical format: \(F(i, D_j, A_1, \ldots, A_e)\)
- An ontology O is mapped to a dimension hierarchy as a tree-like structure.
- A query Q is a subset of dimensions from F which builds an OLAP cube.

---

**Figure 4: Hierarchies.**
CUBO: CUBed Ontologies

- Take advantage of sparse frequency matrix.
- Perform single pass through the data.
- Store the result in a Hash-table.
- Load the Ontology in main memory for summarization by level.
Fact Table Computation

Algorithm 1: CUBO

Input: $O,F,Q,T,\{A_1,\ldots\}$
Output: $R$

/* Init CUBO struct */
1 $R \leftarrow \emptyset$;

/* Load ontology in main memory. */
2 $O \leftarrow \text{LoadOntologyFromOWL}();$

/* Filter $F$ to consider only those $D_j$ in $Q$ */
3 $\hat{F} \leftarrow \{t_i | t_i \in F \land \exists D_j \text{ s.t. } D_j \in t_i \land D_j \in Q\}$;

/* Single data set scan. */
4 $t \leftarrow \emptyset$;

5 while row in $\hat{F}$ do
6     if document changed then
7         /* Add document to CUBO */
8             /* $D_j \in$ row. */
9                 $t \leftarrow t \cup \{D_j\};$
10            end
11            /* Add last document to CUBO */
12            BuildCube($t,O,T,R,\{A_1,\ldots\}$);
13     end
14 end

BuildCube($t,O,T,R,\{A_1,\ldots\}$);
Build Cube per Document

Algorithm 2: BuildCube

Input: $t, O, T, R, \{A_1, \ldots\}$
Output: $R$

$s_h \leftarrow \text{Combos}()$;

/* Aggregate all the existing combos of the $h-1$ level. */

forall $combo$ do

$R \leftarrow R \cup \{1, combo, \{A_1, \ldots\}\}$;

end

/* Recursive function to extract all unique concepts by level $h-2$ to 0. */

$s_{0, \ldots, h-2} \leftarrow \text{CombosForOntologyLevel}(s, O, T)$;

/* Increments found combos by level. */

forall $l$ in $s_{0, \ldots, h-2}$ do

forall $combo$ in $s_l$ do

$R \leftarrow R \cup \{l, combo, \{A_1, \ldots\}\}$;

end

end

return $R$;
Example

Q={Parallel processors, Array and vector processors, Distributed databases}

...3, Array and Vector Processors, 50
3, Distributed databases, 50
3, Parallel Processors, 50

A₁ = 50

Hₜ₋₁
{Array and Vector Processors, Distributed databases}
{Distributed databases, Parallel Processors}
{Array and Vector Processors, Parallel Processors}

Hₜ₋₂
{Parallel Processors, Distributed Systems}

H₀
{Computer Science}
Time Complexity

- Traditional data cube computation $O(nh2^k)$
- The average number of $k$ and $h$ is small.
- Our algorithm has a worst time complexity of $O(n2^{kh})$, but on average performs less computations.
Experiments in a DBMS

• CUBO is a User-Defined Function in C#.

• Our experiments were run on:
  – Intel Xeon Dual Core @3.00 GHz
  – 1 TB Hard drive
  – 4 GB RAM
  – SQL SERVER 2005
Data Sets

Table 1: TPCH Corpora.

<table>
<thead>
<tr>
<th>n</th>
<th>Max $k_j$</th>
<th>Min $k_j$</th>
<th>Total $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>3</td>
<td>1</td>
<td>1038</td>
</tr>
<tr>
<td>10K</td>
<td>3</td>
<td>1</td>
<td>6589</td>
</tr>
<tr>
<td>100K</td>
<td>5</td>
<td>1</td>
<td>9702</td>
</tr>
<tr>
<td>1M</td>
<td>5</td>
<td>1</td>
<td>9702</td>
</tr>
<tr>
<td>10M</td>
<td>5</td>
<td>1</td>
<td>9702</td>
</tr>
</tbody>
</table>

Table 2: dbpedia Corpora.

<table>
<thead>
<tr>
<th>n</th>
<th>Max $k_j$</th>
<th>Min $k_j$</th>
<th>Total $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>9</td>
<td>1</td>
<td>156</td>
</tr>
<tr>
<td>10K</td>
<td>14</td>
<td>1</td>
<td>231</td>
</tr>
<tr>
<td>100K</td>
<td>16</td>
<td>1</td>
<td>263</td>
</tr>
<tr>
<td>1M</td>
<td>26</td>
<td>1</td>
<td>302</td>
</tr>
<tr>
<td>10M</td>
<td>46</td>
<td>1</td>
<td>308</td>
</tr>
</tbody>
</table>
Experiments

Table 3: Performance of Traditional Cube and CUBO (* unable to compute)

<table>
<thead>
<tr>
<th>d</th>
<th>Traditional Single Level</th>
<th>CUBO</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>36</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>36</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>37</td>
<td>9</td>
</tr>
<tr>
<td>16</td>
<td>*</td>
<td>15</td>
</tr>
<tr>
<td>32</td>
<td>*</td>
<td>44</td>
</tr>
<tr>
<td>64</td>
<td>*</td>
<td>96</td>
</tr>
</tbody>
</table>
Experiments

Figure 6: Varying Corpus Size.

Figure 7: Varying Number of Dimensions.

Figure 8: CUBO Size when Varying Number of Dimensions.

Table 5: Varying Ontology Levels in TPCH (time in seconds).

<table>
<thead>
<tr>
<th>n</th>
<th>ALL</th>
<th>MAX 2</th>
<th>MAX 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10K</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>100K</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1M</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10M</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
Conclusions

• CUBO is an efficient and single pass algorithm for summarizing hierarchical data.

• CUBO is faster than using a traditional OLAP algorithm.

• CUBO performs faster than the theoretical upper bound.

• CUBO not sensitive to the branching factor.
Future Work

• Support ontologies that do not fit in main memory.

• Improve scalability on h (more than 5 levels deep).

• Support unbalanced trees (ontologies) and ontologies with multiple parents.

• Support incremental computation of new dimensions.

• CUBO needs to be explored in MPP databases.
References

