Scalable Approximate Query Processing With The DBO Engine

C. Jermaine et al.
Presented by: Yi Zhang
Jan 29, 2008

Problem

- Need for online estimation of query answer
  - Updates as query processing progresses
  - Statistically meaningful bounds
  - Accurate final answer of course
  - Application:
    * Interactive data exploration over large datasets
- Query style
  - SELECT FROM WHERE GROUP BY

Nested Loop Join

- Long delay between successive updates
- Limited contribution to shortening confidence interval

Ripple Join

Estimation & Statistics

- An example
  - SELECT SUM(expression) FROM R, S WHERE p
  - Estimated by \( \sum_{(r,s) \in \text{Run}} p \times \text{expression} \)
- Estimator unbiased, consistent
- Capable of giving tight confidence intervals
- Variance can also be calculated
- But...
  - When samples go out of memory...
  - Significant disk I/Os

SMS Join

- Generalization of ripple join
  - Identical to ripple join when all fits in memory
- Continues to provide confidence bounds when operating on disk
- Phases
  - Sort: read input relations in parallel, hash ripple join, and sort on join attributes
  - Interleave reads and writes across runs
  - Merge: similar to SMJ, parallel with Shrink
  - Shrink: update estimator after removing merged data space
Scalability!

- Two phase model of sort-merge does not scale well

```
SELECT SUM(T.C)
FROM R,S,T
WHERE R.A=S.A AND S.B=T.B
```

- Fixes
  - Two separate joins: Can we do better?
  - Pipelining
    - Bad randomness
    - Hard to estimate e.g. intermediate relation size

How DBO Does It

- We need both scalability and progressive accurate estimation
- Design differences from traditional databases
  - Information sharing across relational operations
  - Processing abstracted into levelwise steps
  - Each levelwise step maintains and updates an online estimator
  - Results used as input for next levelwise step
  - Levelwise estimators combined with all lower level estimators to produce a single estimator for final answer

A Quick Overview

Levelwise Step

- Scan & Merge
- Scan phase
  - Concurrent for input relations in a levelwise step
  - Records of a relation divided into runs
  - In-memory join for unbiased guess
  - Randomized sort order
    - Sort on H(id, R.key)
    - Round-robin processing of runs

Example
Merge Phase

Estimation

- First levelwise step
  - Sum of aggregate of tuples discovered so far * size ratio

- What about subsequent levels?
  - Intermediate results grouped by join key – not completely random
  - Cardinality of intermediate results unknown due to pipelining

Solution to Semi-Randomness

- “Clumps”
  - View group of tuples w/ same join key from a merge phase as a single, indivisible output tuple
  - Estimate using clumps

Solution to Unknown Relation Size

- We don’t need to estimate the unknown size!
- Partition H’s output space into p equi-sized ranges
  - Size ratio: \( \frac{1}{p+1} \)
  - Round-robin scan searches \( 1+(p-1)n \) combinations of runs

Putting All Estimators Together

- d+1 unbiased estimators for d levels
- Take weighted average

- To choose weights:
  - Lagrangian multiplier: \( f(m_1, m_2, \ldots, m_d) = \alpha \cdot \sum_{i=1}^{d} m_i + 1 \)
  - Variance minimized by

Other Cases

- Other aggregates
  - GROUP BY: separate estimators for each group
  - So far only SUM aggregate considered
    - COUNT: also a simple version of SUM
    - AVG: function of SUM and COUNT
    - STD_DEV: function of AVG and COUNT
- Non-bushy query plan trees

Different hash seeds for different levels, so estimators independent
Experiments

• Five queries over TPC-H schema
• 10GB generated data

Running Time and Accuracy

<table>
<thead>
<tr>
<th>Query Execution Time</th>
<th>Query</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Problems and Discussion

• Be organized or random?
  – Randomness provides unbiased online estimation
  – Indexing provides fast access
  – Can we make use of index? Can we access random data fast?
• Current system limits itself in many aspects
  – Sort-merge style join
  – Equality predicates
  – Are we losing the opportunity of optimization by this specific parallel way of query evaluation?