Insights from Sketch-based Relational Query Optimization

Florin Rusu
Department of Computer Science & Engineering
Database Research @ UC Merced

- Faculty
  - Florin Rusu

- Ph.D. students
  - Asoke Datta
  - Yesdaulet Izenov
  - Brian Tsan
Sketch-based Relational Query Optimization

- COMPASS query optimizer [SIGMOD 2021]
  - https://github.com/yizenov/compass_query_optimizer
  - Fully-integrated in the MapD GPU database (HEAVY.AI)
  - Funded by NSF III Core Small award 2008815

```
SELECT MIN(k.keyword), MIN(n.name), MIN(t.title)
FROM cast_info AS ci,
    keyword AS k,
    movie_keyword AS mk,
    name AS n,
    title AS t
WHERE k.keyword = 'marvel-cinematic-universe' AND n.name LIKE '%Downey%Robert%' AND t.production_year > 2010 AND k.id = mk.keyword_id AND t.id = mk.movie_id AND t.id = ci.movie_id AND ci.movie_id = mk.movie_id AND n.id = ci.person_id
```
Sketches for Join Size Estimation

**Push-Down Selection**

<table>
<thead>
<tr>
<th>k.id</th>
<th>k.keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁</td>
<td>x₁</td>
</tr>
<tr>
<td>x₂</td>
<td>x₂</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>xₖ</td>
<td>xₖ</td>
</tr>
</tbody>
</table>

σ(k. keyword) → k.id

<table>
<thead>
<tr>
<th>mk.keyword_id</th>
<th>mk.movie_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>y₁</td>
<td>z₁</td>
</tr>
<tr>
<td>y₂</td>
<td>z₂</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>yₘ</td>
<td>zₘ</td>
</tr>
</tbody>
</table>

**Fast-AGMS Sketch**

```
\left( \begin{array}{c}
  h_{x1} \\
  h_{x2} \\
  \vdots \\
  h_{xk}
\end{array} \right)_{rx1} \rightarrow
\left( \begin{array}{c}
  \xi_1 \\
  \xi_2 \\
  \vdots \\
  \xi_k
\end{array} \right)_{rx1} \oplus
\left( \begin{array}{c}
  a_{1,1} \\
  a_{1,2} \\
  \vdots \\
  a_{1,b}
\end{array} \right)_{rx1}
```

```
\left( \begin{array}{c}
  \sum_{i=1}^{b} \text{sk}^{k.id} (\text{sk}^{y_i,k.id} \odot \text{sk}^{mk.movie_id}) \cdot \text{sk}_i \\
  \sum_{i=1}^{b} \text{sk}^{k.id} (\text{sk}^{y_i,k.id} \odot \text{sk}^{mk.movie_id}) \cdot \text{sk}_i \\
  \vdots
\end{array} \right)_{r \times b} \oplus
\left( \begin{array}{c}
  a_{1,1} \\
  a_{1,2} \\
  \vdots \\
  a_{1,b}
\end{array} \right)_{r \times b}
```

```
\left( \begin{array}{c}
  \sum_{i=1}^{b} \text{sk}^{mk.movie_id} (\text{sk}^{y_i,\text{mk.movie_id}} \odot \text{sk}^{mk.movie_id}) \cdot \text{sk}_i \\
  \sum_{i=1}^{b} \text{sk}^{mk.movie_id} (\text{sk}^{y_i,\text{mk.movie_id}} \odot \text{sk}^{mk.movie_id}) \cdot \text{sk}_i \\
  \vdots
\end{array} \right)_{r \times b} \oplus
\left( \begin{array}{c}
  a_{1,1} \\
  a_{1,2} \\
  \vdots \\
  a_{1,b}
\end{array} \right)_{r \times b}
```

```
\left( \begin{array}{c}
  a_{2,1} \\
  a_{2,2} \\
  \vdots \\
  a_{2,b}
\end{array} \right)_{r \times b}
```

```
\left( \begin{array}{c}
  a_{1,1} \\
  a_{1,2} \\
  \vdots \\
  a_{1,b}
\end{array} \right)_{r \times b}
```

```
\left( \begin{array}{c}
  a_{2,1} \\
  a_{2,2} \\
  \vdots \\
  a_{2,b}
\end{array} \right)_{r \times b}
```

σ(t. year) → t.id
Conditioned Sketches from Masked Models

```
SELECT id FROM title WHERE production_year = 2016
```

- Learn sketch for `id` without pushing down selection on `production_year`
  - Probability of `id` conditioned on `production_year`
  - Ranges are decomposed into dyadic intervals
- Mask model
  - Transformer on attribute embeddings
  - Train on table data
  - No workload required
**Problem**

Fast-AGMS and Count-Min are hash-based sketches that summarize statistics about their data. However, it can only be applied to predefined selections, e.g., a sketch on the entire dataset can be repurposed for the selection of $A x$ where $A y = 2$.

We propose a method for training transformer models to dynamically generate the sketches of arbitrary selections.

**Goals**

- Generate sketches for any arbitrary selection.
- No prior knowledge of database schema required.
- One model per table instead of one model for all tables.
- Good enough cardinality estimation for query optimization.

**Count-Min**

<table>
<thead>
<tr>
<th>id</th>
<th>year</th>
<th>$H_d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2006</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>2012</td>
<td>4</td>
</tr>
<tr>
<td>35</td>
<td>2013</td>
<td>4</td>
</tr>
<tr>
<td>85</td>
<td>1984</td>
<td>2</td>
</tr>
<tr>
<td>95</td>
<td>2004</td>
<td>3</td>
</tr>
</tbody>
</table>

**Approximate Count-Min**

Our work is driven by the following observation:

$$sk_i(id \mid year = y) = \left[ p(h_i(id) = 1 \mid year = y) \right] \times |T|$$

Where $sk_i(id \mid year = y)$, the sketch of the selection defined by year $= y$, is equal to a $d$-dimensional vector of probabilities times the cardinality of the selection in $T$.

**Plan Cost Ratio**

<table>
<thead>
<tr>
<th>Method</th>
<th>Runtime (Mean)</th>
<th>P-error Percentiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count-Min (Approx)</td>
<td>1.50h</td>
<td>2.87 0.00 1.25 2.14 61.49</td>
</tr>
<tr>
<td>Fast-AGMS (Approx)</td>
<td>1.48h</td>
<td>2.80 0.00 1.33 2.10 61.49</td>
</tr>
<tr>
<td>Fast-AGMS</td>
<td>1.50h</td>
<td>2.78 0.00 1.50 2.88 61.49</td>
</tr>
<tr>
<td>Count-Min (Upper)</td>
<td>1.53h</td>
<td>2.72 0.01 1.36 2.14 61.49</td>
</tr>
<tr>
<td>Count-Min</td>
<td>1.50h</td>
<td>2.65 0.00 1.04 1.79 61.49</td>
</tr>
<tr>
<td>NeuroCard</td>
<td>1.51h</td>
<td>2.53 1.00 1.01 1.48 61.49</td>
</tr>
<tr>
<td>Fast-AGMS (Upper)</td>
<td>1.54h</td>
<td>1.30 1.03 1.36 1.84 4.31</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>1.68h</td>
<td>1.13 1.00 1.04 1.75 2.70</td>
</tr>
</tbody>
</table>

**Upper Bound Approximation**

For selections with multiple predicates, generating sketches requires inferring multiple conditional probabilities.

$$sk_i(X \mid y, z) = p(X \mid y, z) \times \frac{p(y, z)}{|T|}$$

Where $p(X \mid y, z)$ and $p(y \mid z)$ are to be inferred. We reduce the inferrences by approximating an upper bound of the sketch.

**Conclusion**

By dynamically generating sketches, we can compute sketches for arbitrary selections. This alleviates hash-based sketches of requiring predefined selections. The generated sketches are nearly as effective as their original, for query optimization in PostgreSQL.
Sub-optimal Join Order Classification

Q-error
- Cardinality estimation error
- Poor indicator for plan cost

L1-error
- Permutation error
- Swap weight
- Position weight

<table>
<thead>
<tr>
<th>2-way joins (CARDINALITY only)</th>
<th>$\rho$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{\rho}$</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Swap weight</td>
<td>1.0</td>
<td>107.84</td>
<td>62.36</td>
<td>1.73</td>
<td>7.71</td>
<td></td>
</tr>
<tr>
<td>Increasing weight $\mu$</td>
<td>1.0</td>
<td>108.84</td>
<td>171.2</td>
<td>172.93</td>
<td>180.64</td>
<td></td>
</tr>
<tr>
<td>Position weight $\delta$</td>
<td>107.84</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>
Sub-optimal Join Order Classification by L1-error
Yesdaulet Izenov, Asoke Datta, Brian Tsan and Florin Rusu
University of California, Merced
yizenov, adatta2, btsan, frusu@ucmerced.edu

Q-error Limitations

Goal: classification of sub-optimal join orders
Q-error is a poor indicator of sub-optimal join orders:
• too loose as a bound
• does not consider enumeration algorithms
• cardinality estimation errors matter equally

Q-error Bound (JOB queries)

Simple Cost Function (query 2c)

Cardinality Accuracy (JOB queries)

Join Size Importance (JOB queries)

Query Plans (query 2c)

2-way joins (COST + CARDINALITY)

3-way joins (COST + CARDINALITY)

4-way joins (COST + CARDINALITY)

5-way joins (COST + CARDINALITY)

L1-error

Sub-query Relative Order: estimation errors do not matter while sub-query relative order is preserved based on cardinality estimates
Sub-query Weights: the larger the weight, the greater the difference between two sub-queries
Small Cardinalities: sub-query with smaller cardinality is likely to be selected by the plan search algorithm (e.g., greedy algorithm)
Join Size Importance: early-stage multi-way joins are more critical and tend to be more accurate than joins at later stages

Dataset and Workload

IMDB Benchmark Dataset: real-world dataset containing correlations and non-uniform data distributions
Join Order Benchmark: challenging realistic workload — 113 queries
• 45 Simple queries with 4-9 join predicates
• 53 Moderate queries with 10-19 join predicates
• 15 Complex queries with 20-28 join predicates

Classification Results (JOB queries)

Yesdaulet Izenov, Asoke Datta, Brian Tsan and Florin Rusu (UC Merced)
Cardinality Estimation Impact on Query Plans

1. sort FK increasing: [mi_idx, mi, ci]
2. remove duplicates and sort FK_C[f] increasing: 
   - mi_idx : it2, t, mi, ci
   - mi : it1, mi_idx, t, ci
   - ci : mi_idx, t, n, mi

3. one-to-many join partitions:
   - mi_idx \times it2 \times t
   - mi \times it1
   - ci \times n

4. final join order:
   - mi_idx \times it2 \times t \times (mi \times it1) \times (ci \times n)

Graphs:
- a) non-indexed runtime in seconds
- b) indexed runtime in seconds
Cardinality Estimation in Query Optimization

Cardinality estimation is the primary input for query optimizers, enabling them to:
- Enumerate over search spaces and cost join orders.
- Choose the most cost effective join order.
- Identify optimal physical operators for execution.
- Reduce resource usage and minimize response times.
These factors contribute to improved database performance and efficient query processing.

Motivation

Leis et al. made an observation in their paper, “How Good Are Query Optimizers, Really?”

Setting
Indexes -> Primary Key Tables
Nested Loop Joins -> Disabled
Hash Join -> Enabled

Outliers
Foreign/ Foreign key joins
to avoid

Most queries performance is similar to optimal
Cardinality estimation need. Bad estimates are good enough

Experiments were conducted on PostgreSQL 9.4, a version lacking support for parallel query processing. With recent PostgreSQL releases incorporating parallel processing capabilities, it is worthwhile to revisit and delve deeper into the implications of this observation.

10K random plans cost - QuickPick

1. Comparison of the JOB workload performance in non-indexed and indexed settings. cost (Figures a, b) and runtime (Figures b, c).
2. Cost and runtime analysis 113 JOB queries - non-indexed settings.
3. Cost and runtime analysis 113 JOB queries - indexed settings.
4. Impact of suboptimal physical optimizer choices on optimal join-order.

Methodology

- Ubuntu 20.04, Intel Xeon E5-2660 v4(2.00GHz), 28 CPU, 256 GB RAM and HDD storage.
- PostgreSQL version 14.2 (Capable of inject Cardinalities for subqueries).
- shared buffers, work mem, effective cache size, max wal size → 128 GB.
- Indexed(Hash Join and Nestedloop Join) and non-indexed(Hash Join only) configurations.
- Dataset - IMDB and Workload - Join Order Benchmark.
Multidimensional Array Data Management

By Florin Rusu

Multidimensional arrays are one of the fundamental computing abstractions to represent data across virtually all areas of science and engineering, and beyond. Due to their ubiquity, multidimensional arrays have been studied extensively across many areas of computer science.

This survey provides a comprehensive guide for past, present, and future research in array data management from a database perspective. Unlike previous surveys that are limited to raster processing in the context of scientific data, this survey considers all types of arrays: rasters, data cubes, and tensors. The author’s goal is to identify and analyze the most important research ideas on arrays and to serve two objectives: first, to summarize the most relevant work on multidimensional array data management by identifying the major research problems; and second, to organize this material to provide an accurate perspective on the state-of-the-art and future directions in array processing.

Multidimensional Array Data Management covers all data management aspects, from array algebras and query languages to storage strategies, execution techniques, and operator implementations. Moreover, the author discusses which research ideas are adopted in real systems and how they are integrated in complete data processing pipelines. Finally, the author compares arrays with the relational data model. The result is a thorough survey on array data management that is an excellent resource for anyone interested in this topic, independent of experience level.

Invitation

- Come to see our posters
- Come to visit UC Merced. It's on us!!!