Analyzing Plan Diagrams of Data Query Optimizers
Paper

• Analyzing Plan Diagrams of Database Query Optimizers
• written in 2005
• Naveen Reddy, Jayant R. Haritsa
• Database Systems Lab, SERC/CSA
• Indian Institute of Science, Bangalore 560012, INDIA
Relational Database Management System

- Where is a Query Optimizer located?
Query Optimizer

- How does a Query Optimizer work?

Parsed Query (from Parser) → Query Transformer → Transformed Query → Estimator → Query and estimates → Plan Generator → Query Plan (to Row Source Generator) → Dictionary

subquery unnesting, query rewrite with materialized views, ...
Query Optimizer

- Identifies the most efficient strategy to execute SQL queries
- Returns query plan
- Part of the RDBMS
- Based on estimated number of rows of relevant relations
- Relevant join conditions are given in the query
Query Plan

- A strategy to execute SQL queries
- Query Plan does not affect the result
- Example: (id is FK of Aid)
- select o,p from A inner join B on (A.id=B.Aid) where B.p=0;
- Plan P1: AB
  - A=3, B=6/3=2 => cost(AB)=6
- Plan P2: BA
  - B=3 (using where), A=1 (primary key) => cost(BA)=3
How to analyze a query optimizer?

1. Send SQL statement `EXPLAIN SELECT ... FROM ... WHERE ...`

2. Optimizer returns query plan

3. This can be repeated by varying the where condition(s) systematically until the selectivity space is covered

4. Corresponding plan and cost for each set of conditions can be visualized

Orders

Customers

Selectivity Space

4711

13

Orders
Query plan costs

test=# explain select o,p
    from A inner join B on (A.id=B.Aid)
    where B.p=0;

QUERY PLAN

Hash Join  (cost=1.07..2.18 rows=3 width=6)
    Hash Cond: (b.aid = a.id)
    ->  Seq Scan on b  (cost=0.00..1.07 rows=3 width=8)
        Filter: (p = 0)
    ->  Hash  (cost=1.03..1.03 rows=3 width=6)
        ->  Seq Scan on a  (cost=0.00..1.03 rows=3 width=6)

<table>
<thead>
<tr>
<th>A</th>
<th>id</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>id</th>
<th>Aid</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
TPC-H

- Decision support benchmark
  - database schema, data
  - queries for testing (Q1 through Q22)
- We use same data set as the authors of the paper:

<table>
<thead>
<tr>
<th>relation</th>
<th>#rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>part.tbl</td>
<td>200000</td>
</tr>
<tr>
<td>nation.tbl</td>
<td>25</td>
</tr>
<tr>
<td>customer.tbl</td>
<td>150000</td>
</tr>
<tr>
<td>region.tbl</td>
<td>5</td>
</tr>
<tr>
<td>partsupp.tbl</td>
<td>800000</td>
</tr>
<tr>
<td>supplier.tbl</td>
<td>10000</td>
</tr>
<tr>
<td>orders.tbl</td>
<td>1500000</td>
</tr>
<tr>
<td>lineitem.tbl</td>
<td>6001215</td>
</tr>
</tbody>
</table>
Picasso

• Picasso
  • Sends explain queries against the database
  • Stores and visualizes query plan and costs

• Picasso Server
  • Connects to Database (SQL-Server, Oracle, Sybase, postgresql, …)
  • Multiuser

• Picasso Client
  • Connects to PICASSO Server
  • GUI for PICASSO Server
Run test with Picasso
Run test with Picasso

- Picasso varies WHERE conditions according to plot resolution:
  - and o_totalprice :varies
  - and c_acctbal :varies
- 2 varying conditions ⇒ 2-dimensional selectivity space (2D)
- Picasso runs EXPLAIN-Statements to determine the query plan chosen by the optimizer.
- 2D and Resolution=100 ⇒ $100^2 = 10,000$ EXPLAIN-Statements
Diagram creation process

Create „EXPLAIN SELECT ...“ SQL-Statement with condition(s)

Optimizer

SQL

RDBMS

Add new point to the Plan Diagram

Query Plan

Output Diagram

explain select name, price from customer, order where [...] and o_totalprice <= 98679 and c_acctbal <= 5809; result: P4
Plan Diagram and Cost Diagram

Plan Diagram: 2D-visualization of execution plans

Cost Diagram: 3D-visualization of estimated costs. Estimation is done by optimizer.
Cost domination Principle

Monotonic cost behaviour:
Increasing selectivity space increases estimated costs

Non-monotonic cost behaviour:
Increasing selectivity space does not always increase estimated costs

Cost domination principle:
When increasing selectivity space, estimated costs must increase monotonically.
## Estimated costs vs. execution costs

<table>
<thead>
<tr>
<th></th>
<th>Estimated costs</th>
<th>Execution costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Statement for each point in diagram</td>
<td>explain select ...</td>
<td>select ...</td>
</tr>
<tr>
<td>Measurement</td>
<td>#rows, ...</td>
<td>execution time</td>
</tr>
<tr>
<td>Creation time for a 10x10 diagram</td>
<td>10s</td>
<td>4h 10min</td>
</tr>
<tr>
<td>Monotonicity</td>
<td>Estimated cost diagram should always be monotonic</td>
<td>Execution cost diagram might not be monotonic, e.g. if estimations were not precise</td>
</tr>
</tbody>
</table>

=> Even if we have Monotonic estimated cost behaviour, we want to reduce #Plans and #Segments.
Reduced Plan Diagram

Plan Diagram

Reduced Plan Diagram: Maximum allowed Cost Increase: 5% 10%

Reduced Plan Diagram:
Try to reduce #Plans without increasing estimated costs more than X% in each point
(X% is called Costgreedy Reduction or plan optimality tolerance threshold.)
TPC-H Query 7

Plan Diagram
- 4 Plans
- P4 covers 1.78% only
- Many small segments
- Not smooth

Cost Diagram
- monotonic

Reduced Plan Diagram
- 2 Plans only
- Max Increase ≤ 6.73%
- Avg Increase ≤ 0.72%
TPC-H Query 8

**Plan Diagram**
- 9 Plans
- 80% covered by 3 Plans
- Many small segments
- Not smooth

**Cost Diagram**
- monotonic

**Reduced Plan Diagram**
- 4 Plans only
- Max Increase ≤ 9.8%
- Avg Increase ≤ 0.41%
TPC-H Query 9

Plan Diagram
• 22 Plans
• 80% covered by 3 Plans
• Many small segments
• Not smooth
• Highly instable

Cost Diagram
• monotonic

Reduced Plan Diagram
• 10 Plans
• Max Increase ≤ 9.15%
• Avg Increase ≤ 0.15%
Clarifications

- #Plans does not necessarily correspond with execution time of one query. We do not rate performance!
- But: We consider the **stability** of an optimizer.
Hypothesis

- Modern query optimizers make extremely fine-grained plan choices
- smaller plans occupy less than 1% of selectivity space
- Optimizers are over-sophisticated
- Processing overheads with query optimization could be lowered
Parametric query optimization (PQO)

- **PQO**
  - Apriori identify the optimal set of plans at compile time
  - At run time, use plan corresponding to selectivity parameter(s).

- **PQO Assumptions:**
  - **Plan Convexity**
    - If Plan P is optimal at point A and at point B, it is also optimal at all points joining the two points
  - **Plan Uniqueness**
    - An optimal plan P appears at only one contiguous region in the entire space
  - **Plan Homogenity**
    - An optimal plan P is optimal within the entire region enclosed by its plan boundaries.

But: today's optimizers can look like that:

Better: reduce #plans for PQO
Conclusion

- Often highly intricate diagrams
- Typically 80% of space covered by 20% of plans (postgresql: 37%)
- Cardinality of plan diagram can be reduced, without materially affecting the query cost
- Assumptions of parametric query optimization literature do not hold in practice. (Reduction needed)
- Needed:
  - Mechanism for pruning the plan search space
  - Directly reduce #plans by optimizer
Discussion

Plan Diagram

Cost Diagram

Reduced Plan Diagram

Do you have any questions?
Comparison of Database Analyzers

<table>
<thead>
<tr>
<th>TPC-H Query</th>
<th>#Plans</th>
<th>%plans for 80%</th>
<th>Gini Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>100,00%</td>
<td>0.47</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>37,50%</td>
<td>0.65</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>50,00%</td>
<td>0.57</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>33,33%</td>
<td>0.68</td>
</tr>
<tr>
<td>9</td>
<td>22</td>
<td>13,64%</td>
<td>0.76</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>22,22%</td>
<td>0.48</td>
</tr>
<tr>
<td>18</td>
<td>7</td>
<td>14,29%</td>
<td>0.24</td>
</tr>
<tr>
<td>21</td>
<td>4</td>
<td>25,00%</td>
<td>0.28</td>
</tr>
<tr>
<td>postgresql</td>
<td>8.13</td>
<td>37,00%</td>
<td>0.52</td>
</tr>
</tbody>
</table>

- OptX: Commercial DBs tested by IIS
  - postgresql: Same dataset, similar reference system
- postgresql uses smaller #Plans to cover space.
- postgresql needs higher percentage of plans to cover 80% of selectivity space.
- postgresql's Gini Index is close to 0.5 which is better than 0.7 or 0.8.
Generation of Reduced Plan Diagram

- P1, P4 are upper bounds of P2
- We move point to P1 which has lowest cost (90)
- If all Points of a plan are swit
Query plan costs

- The order of the operations in a query plan is important:
- \( \text{cost}(AB) \neq \text{cost}(BA) \)
- \( \text{cost}(ABC) \neq \text{cost}(A) \times \text{cost}(BC) \)
- \( \text{cost}(ABC) \neq \text{cost}(AB) \times \text{cost}(C) \)
Complex patterns

- Rapidly alternating choices
- not created on postgresql