A Brief Overview of Query Optimization

Wentao Wu
Microsoft Research
2/9/2018
Outline

• Architecture of Query Optimizer

• Cost Modeling

• Dynamic Query Optimization
Architecture of Query Optimizer
Conceptual View

Query (SQL) → Parse → Abstract Syntax Tree (AST) → Analyze → (Relational) Algebraic Tree → Logical Optimization → Physical Optimization → (Best) Execution Plan

Search (Iterating)
Logical Optimization

• **Goal:** Produce logically equivalent (relational) algebraic trees.

• Common techniques
  • Push down selections/projections/aggregations.
  • Reorder joins (inner/outer/semi/anti joins).
  • Rewrite nested subqueries.

• References
  [1] U. Dayal. Of nests and trees: A unified approach to processing queries that contain nested subqueries, aggregates, and quantifiers. (VLDB’87)
  [2] Weipeng P. Yan, Per-Åke Larson: Eager Aggregation and Lazy Aggregation. (VLDB’95)
Physical Optimization

• **Goal:** Replace logical operators in the algebraic tree with physical operators.
  - E.g., join => hash/merge/nested-loop join
  - E.g., aggregation => sort-based/hash-based aggregation

• Common techniques
  - Rule-based: E.g., pattern matching in SparkSQL.
  - Cost-based: Use a cost model to estimate execution cost of a physical plan.

• References
    (ACM Trans. Database Syst., 1986)
Search Framework

• **Goal:** Search for the “best” execution plan (i.e., the plan with the lowest cost).

• Common techniques
  • Bottom-up: Dynamic programming (System R). Used by Oracle, IBM DB2, PostgreSQL.
  • Top-down: Volcano => Cascades. Used by Microsoft SQL Server, Greenplum (Pivotal).

• References
Other References

• Surveys

• Search frameworks
  [4] Immanuel Trummer, Christoph Koch: Solving the Join Ordering Problem via Mixed Integer Linear Programming. (SIGMOD’17)
Is Query Optimization a “Solved” Problem?

• Well, it has been 40 years since the 1979 System-R paper ...

What are the “right” problems that remain unsolved?
Cost Modeling: The Pain
Query Optimizer Needs Good Cost Models

• Unlike the other components in a query optimizer, cost modeling lacks a standard procedure and is case-by-case.
  • It is based on the “knowledge” from the database system developers about the relative execution overheads of different operators.

• Common techniques
  • Analytical modeling: Used by most (if not all) major database systems.
  • Machine learning: One of the hot research areas in recent years.
Analytic Modeling

- Basically develop cost formulas for different operators.
  - E.g. Cost = CPU cost + I/O cost + Network communication cost

- Cost models need validation and calibration.

References

[1] Lothar F. Mackert, Guy M. Lohman: R* Optimizer Validation and Performance Evaluation for Local Queries. (SIGMOD’86)
Machine Learning

• Don’t trust the cost formulas made by optimizer developers.
  • Learn cost functions based on query execution data.

• References

  ([3] and [4] combine analytic modeling with machine learning.)
Cardinality Estimation: The Hardest Part

• No matter you use analytic modeling or machine learning, you need cardinality information (i.e., sizes of intermediate results produced by operators in query execution plans).

• Recent work shows that cardinality estimation may be the most (and often the only) important thing in cost modeling.


• Common techniques
  • Use histograms: equi-width, equi-depth, multi-dimensional.
  • Use samples, sketches, statistical models, execution feedback, etc.
Single-column Histograms

• Histograms for a single column
  • Equi-width, equi-depth, V-optimal, ...
  • **Attribute-Value-Independence (AVI) assumption**: Assume the independence between histograms (i.e., distributions) when estimate selectivity/cardinality for predicates involving more than one columns (e.g., $X > 3$ and $Y < 8$).
  • The estimation error can be exponential under AVI.

• References
  [3] V. Poosala et al., Improved histograms for selectivity estimation of range predicates. (SIGMOD’96)
Multi-column Histograms

• Motivation: Overcome the AVI assumption.
  • Use histograms to capture the joint distribution across multiple columns.

• Drawback: The size increases exponentially w.r.t. the # of columns.
  • Workload-driven approaches: Only construct multi-column histograms for columns that appear in workload queries.

• References
Other Approaches

• Sampling/Sketches (References)
  [4] Phillip B. Gibbons: Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports. (VLDB’01)

Theoretical results:
  [3] M. Riondato et al., The VC-Dimension of SQL Queries and Selectivity Estimation through Sampling. (ECML/PKDD’11)

• Statistical models/Feedback (References)
Dynamic Query Optimization
Motivation

• So far, we have been talking about “static query optimization”.
  • We assume that a query plan is ready and won’t be changed during execution.
  • The performance of the query plan is subject to cost modeling, which depends on the accuracy of cardinality estimation, an inherently hard problem.

• However, why should we stick with one single query plan?
  • We shouldn’t!

• Dynamic query optimization (a.k.a., interleave query optimization with query execution/processing)
  • Let’s prepare multiple plans and decide at runtime which one(s) to use.
Two Key Problems

• How to generate multiple query plans?

• When to switch to a different query plan?

• Common techniques
  • Parametric query optimization
  • Adaptive/robust query processing
  • Mid-query re-optimization
Parametric Query Optimization

• Mainly used for stored procedures (query templates).
  • Rather than use one plan for all parameter values, use different plans for different parameter values.
  • Multiple plans are generate during query compilation/optimization.
  • Pick one plan before execution depending on the parameter value observed.

• References
  [1] Y. E. Ioannidis et al., Parametric Query Optimization. (VLDB’92)
Adaptive/Robust Query Processing

• Generate multiple plans during query compilation.
  • Similar to parametric query optimization (PQO).

• Dynamically switch plans based on feedback from execution.
  • This is different from PQO, which does not switch after execution starts.

References

Mid-Query Re-Optimization

• Start from the plan generated by the optimizer.
  • So there is only one plan after query compilation/optimization stage. (This is different from PQO and adaptive/robust query optimization.)

• At runtime, keep monitor feedback from query execution.
  • If there is evidence that the current plan is sub-optimal (e.g., significant cardinality estimation error), stop execution and ask the optimizer to re-optimize the remaining part of the query based on execution feedback.

• References
  [2] V. Markl et al., Robust Query Processing through Progressive Optimization. (SIGMOD’04)
  [4] W. Wu et al., Sampling-based query re-optimization. (SIGMOD’16)
Thank you!
The Optimizer of Microsoft SQL Server

• Reference for the example below:

[1] Florian Waas, César A. Galindo-Legaria: Counting, Enumerating, and Sampling of Execution Plans in a Cost-Based Query Optimizer. (SIGMOD’00)

Figure 1: Copying the initial plan into the MEMO structure.
The Optimizer of Microsoft SQL Server (Cont.)

Figure 2: MEMO structure representing alternative solutions.

Figure 3: MEMO Structure with materialized links between operators and children, for possible plans rooted in operator 7.7.