Overview of Query Optimization in Relational Systems

Presenter: Albert Wong
Discussion: Stephen Ingram

Introduction

• 2 key components of query evaluation component of a SQL database system
  – Query optimizer
  – Query execution engine

Query Execution Engine

• Implements a set of physical operators
  • A physical operator takes as input one or more data streams and produces an output data stream
  – Ex. (external) sort, sequential scan, index scan, nested loop join, sort-merge join
  – pieces of code that are used as building blocks to make possible the execution of SQL queries
  – responsible for the execution of the operator tree (execution plan) that results in generating answers to the query

Example Operator Tree

Query Optimizer

• Responsible for generating the input for the execution engine
  • Takes a parsed representation of a SQL query as input
  • Responsible for generating an efficient execution plan for the given SQL query from the space of possible execution plans
The Key Idea: Query Optimization as a Search Problem

- To solve problem, we need to provide:
  - Search space
  - Cost estimation technique so that a cost may be assigned to each plan in the search space
  - Enumeration algorithm that can search through the execution space

Goals of an Optimizer

- Search space includes plans that have low cost
- Costing technique is accurate
- Enumeration algorithm is efficient

Search Space

- Depends on the set of algebraic transformations that preserve equivalence and the set of physical operators supported in an optimizer
- Transformations do not necessarily reduce cost and therefore must be applied in a cost-based manner by the enumeration algorithm to ensure a positive benefit

Commuting Between Operators

- Generalized Join Sequencing
- Outer Join and Join
  - Join(R, S LOJ T) = Join(R, S) LOJ T
- Group-By and Join

Linear and Bushy Joins

- Does the formulation of a query affect the execution of that query? Can users optimize their queries’ execution through better syntax?
- Bushy Joins: Is it naive to just leave them out of the search? Why do we always only consider linear joins? When would this cause problems? How could we incorporate bushy joins into our search?
Multi-Block Query to Single-Block

- **Merging Views**
  
  \[
  \text{Q} = \text{Join}(R, V)
  \]

  \[
  \text{View } V = \text{Join}(S, T)
  \]

  \[
  Q = \text{Join}(R, \text{Join}(S, T))
  \]

- **Merging Nested Subqueries**

Statistics and Cost Estimation

- Cost estimation must be accurate because optimization is only as good as its cost estimates.
- Must be efficient as it is repeatedly invoked by the optimizer.
- Basic estimation framework:
  - collect statistical summaries of data stored
  - given an operator and statistical summaries of its input streams, determine
    - statistical summary of output data stream
    - estimated cost of executing the operation

Statistical Summaries of Data

- Example: number of tuples in table, number of physical pages used by table, statistical information on columns such as histograms.
- Can use sampling to determine histograms that are accurate for a large class of queries.
- Estimating distinct values is provably error prone.
- Statistics must be propagated from base data to be useful.
  - Can be difficult as assumptions must be made when propagating statistical summaries.

A Statistical Discussion

- Some estimated statistics are provably erroneous. Is it then worth estimating? If so, what sort of strategy should we adopt when using estimates with known problems?

Cost Computation

- CPU, I/O, (parallel or distributed systems) communication costs are all factors in cost estimation.
- Difficult to determine best cost estimator.
- Statistical summary propagation and accurate cost estimation are difficult open issues in query optimization.

Enumeration Architectures

- Enumeration algorithm must pick an inexpensive execution plan for a given query by exploring the search space.
- Enumerators tend to concentrate on linear join sequences rather than bushy join sequences due to the size of the search space including bushy join sequences.
- Want to build enumerator that can adapt to changes in search space:
  - New transformations
  - Addition of new physical operators
  - Changes in cost estimation techniques.
Extensible Optimizers

- Use of generalized cost functions and physical properties with operator nodes
- Use of a rule engine that allows transformations to modify the query expression or the operator trees
- Exposed “knobs” to tune behavior of system
- Ex. Starburst and Volcano/Cascades

Materialized Views

- Views cached by database system
- Query can take advantage of materialized views to reduce the cost of executing the query
- Problems
  - Reformulating query to take advantage of materialized views (general problem is undecidable)
  - Determining effective sufficient conditions is nontrivial

Summary of Chaudhuri’s Paper

- Query optimization considered a search problem whose solution requires a search space, cost estimation technique, and an enumeration algorithm
- Query optimization can be considered an art
- No one knows what the best execution plan for a given query is

Ending Discussions

- Most of us have decided that the Relational Model is the way to go. These papers, however, show that under the hood are some scary problems and black magic. Is this surprising? Why (not)?
- This paper is from 98, System R paper is from ~ 20 years earlier. How has query optimization changed in that span of time? Is the amount/direction of progress surprising? Do you expect much change in the years since the printing of this paper?
- What other areas of Computer Science is query optimization like? Could it benefit from ideas from other areas? How?