Relational Query Optimization

Yanlei Diao
UMass Amherst
Overview of Query Evaluation

- **Query evaluation plan**: tree of relational algebra operators, with choice of algorithm for each operator.

- **Query optimization**: given a query, many plans are possible
  - Ideally, find the most efficient plan.
  - In practice, avoid worst plans in practice.
SQL Refresher

SELECT {DISTINCT} <list of columns>
FROM <list of relations>
{WHERE <list of "Boolean Factors">}
{GROUP BY <list of columns>}
{HAVING <list of Boolean Factors>}
{ORDER BY <list of columns>};

- Query Semantics:
  1. Take Cartesian product (a.k.a. cross-product) of relns in FROM, projecting only to those columns that appear in other clauses
  2. If a WHERE clause exists, apply all filters in it
  3. If a GROUP BY clause exists, form groups on the result
  4. If a HAVING clause exists, filter groups with it
  5. If an ORDER BY clause exists, make sure output is in right order
  6. If there is a DISTINCT modifier, remove duplicates
Basics of Query Optimization

- Convert selection conditions to **conjunctive normal form (CNF)**:
  - \((day < 8/9/94 \text{ OR } bid = 5 \text{ OR } sid = 3) \text{ AND } (rname = 'Paul' \text{ OR } sid = 3)\)

- Interleave FROM and WHERE into an **operator tree for optimization**.

- Apply GROUP BY, HAVING, DISTINCT and ORDER BY at the end, pretty much in that order.
Outline

- Query plans and equivalences

- Query optimization issues
  - Plan space
  - Cost estimation
  - Plan search

- Handling nested queries
Schema for Examples

Sailors (sid: integer, sname: string, rating: integer, age: real)
Reserves (sid: integer, bid: integer, day: dates, rname: string)

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
    R.bid=100 AND S.rating>5

Expression in Relational Algebra (RA):
\[ \pi_{sname} (\alpha_{bid=100 \land rating>5} (Reserves \bowtie_{sid=sid} Sailors)) \]
Query Evaluation Plan

- **Query evaluation plan** extends an RA tree with:
  - *access method* for each relation;
  - *implementation method* for each other operator.

- What are the missed opportunities?
  - Selections could have been `pushed` earlier.
  - Use of indexes.
  - More efficient joins.
Query Plan 1 (Selection Pushed Down)

- **Push selections below the join.**
- **Materialization vs. Pipelining:**
  - Store a temporary relation T, if the subsequent join needs to scan T multiple times.
  - The opposite is pipelining.

- With 5 buffer pages, cost of plan:
  - Scan Reserves (1000) + write temp T1 (10 pages, if we have 100 boats, uniform distribution).
  - Scan Sailors (500) + write temp T2 (250 pages, if we have 10 ratings).
  - **Sort-Merge join**: Sort T1 (2*2*10), sort T2 (2*4*250), merge (10+250).
  - Total = 4060 page I/Os.
Query Plan 2 (Different Join Method)

- Change the join method to block nested loops join.

- With 5 buffer pages, cost of plan:
  - Scan Reserves (1000) + write temp T1 (10 pages).
  - Scan Sailors (500) + write temp T2 (250 pages).
  - BNL join: join cost = 10 + 4*250.
  - Total cost = 2770.
Outline

- Query plans and equivalences

- Query optimization issues
  - Plan space
  - Cost estimation
  - Plan search

- Handling nested queries
Three Main Issues in Optimization

Given a query block, three main optimization issues:

- **Plan space**: what plans are considered?
- **Plan cost**: what is the cost of each plan?
- **Search algorithm**: how do we search the plan space for the cheapest estimated plan?
Highlights of System R Optimizer

- **Impact:** most widely used; works well for < 10 joins.

- **Plan Space:** too large, must be pruned.
  - Only considers the space of *left-deep plans*.
  - Avoids *cartesian products*!

- **Cost of a plan:** approximate art at best.
  - Uses statistics to estimate *costs of operations* and *result sizes*.
  - (Considers combination of CPU and I/O costs.)

- **Plan Search:** dynamic programming
  - Prunes useless subplans.
(1) Plan Space

- For each query block, the plans considered are:
  - All *access methods*, for each reln in the FROM clause.
  - All *left-deep join trees*: all the ways to join the relns one-at-a-time, with the inner reln in the FROM clause.
  - All permutations of N relns: N factorial!
Plan Space

- For each block, the plans considered are:
  - All *access methods*, for each reln in FROM clause.
  - All *left-deep join trees*: all the ways to join the relns one-at-a-time, with the inner reln in the FROM clause.
    - All permutations of N relns: *N factorial*!
    - But avoid *Cartesian products*!
      Join R, S, T w. R.a = S.a and S.b = T.b, how many left-deep trees?
  - All *join methods*, for each join in the tree.
  - Appropriate *places for selections and projections*. 
(2) Cost Estimation

- For each plan considered, must estimate its cost.

- Estimate cost of each operation in a plan tree:
  - Depends on input cardinalities.
  - Depends on the method (sequential scan, index scan, join...)

- Estimate size of result for each operation in tree:
  - Use statistics about input relations.
  - Estimate the reduction factors of predicates.
(3) Plan Search

- As the number of joins increases, the number of alternative plans grows rapidly.

- System R: (1) use *only left-deep join trees*, (2) avoid *Cartesian products*.
  - Allow *pipelined* plans; intermediate results not written to temporary files.
  - Not all left-deep trees are fully pipelined!
    - Sort-Merge join: at least sorting phase
    - Two-phase hash join: partitioning phase
Search Algorithm

- Left-deep join plans:
  - Differ in the order of relations, access method for each relation, join method for each join.
  - But may share common prefixes, so don’t enumerate all. Instead use Dynamic Programming

- Dynamic Programming
  - “a method for solving problems that exhibit the properties of overlapping subproblems and optimal substructures”
  - Find the best plans to access A, B, C, D individually
  - Find the best plans for joining A-B, A-C, A-D, B-A, B-C, B-D, C-A, C-B, C-D, D-A, D-B, D-C; store the best for (A-B), (A-C), (A-D), (B-C), (B-D), (C-D)
  - Repeat this for 3 relation sets…
  - Repeat this for 4 relation sets…
  - This procedure is revised if given specific join predicates of A,B,C,D (i.e., left deep trees but avoiding Cartesian products).
**An Example Star Schema**

- **Dynamic Programming**
  “a method for solving problems that exhibit the properties of *overlapping subproblems* and *optimal substructures*”
  - Find the best plans to access A, B, C, D individually
  - Repeat this for 3 relation sets…
  - Repeat this for 4 relation sets…

![Diagram of a star schema with nodes A, B, C, D connected in a star pattern.]