Relational Query Optimization

Chapter 15

Highlights of System R Optimizer

- **Impact:**
  - Most widely used currently; works well for < 10 joins.
- **Cost estimation:**
  - Approximate art at best.
  - Statistics, maintained in system catalogs, used to estimate cost of operations and result sizes.
  - Considers combination of CPU and I/O costs.
- **Plan Space:**
  - Too large, must be pruned.
  - Only the space of left-deep plans is considered.
  - Left-deep plans allow output of each operator to be pipelined into the next operator without storing it in a temporary relation.
  - Cartesian products avoided.

Overview of Query Optimization

- **Plan:** Tree of R.A. ops, with choice of alg for each op.
  - Each operator typically implemented using a ‘pull’ interface: when an operator is ‘pulled’ for the next output tuples, it ‘pulls’ on its inputs and computes them.
- **Two main issues:**
  - For a given query, what plans are considered?
  - Algorithm to search plan space for cheapest (estimated) plan.
  - How is the cost of a plan estimated?
- **Ideally:** Want to find best plan.
  - Practically: Avoid worst plans!
- We will study the System R approach.

Query Blocks: Units of Optimization

- An SQL query is parsed into a collection of query blocks, and these are optimized one at a time.
- Nested blocks are treated as calls to a subroutine, made once per outer tuple. (This is an over-simplification, but for now...)
- For each block, the plans considered are:
  - All available access methods, for each reln in FROM clause.
  - All left-deep join trees (i.e., all ways to join the relations one-at-a-time, with the inner reln in the FROM clause, considering all reln permutations and join methods.)

Schema for Examples

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

- Similar to old schema; rname added for variations.
- 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.

Relational Algebra Equivalences

- **Selections:** σc1...cn(R) = σc1(σc2(...(σcn(R)))) (Cascade)
- **Projections:** πc1...cn(R) = πc1(...(πcn(R))) (Cascade)
- **Joins:** R join (S join T) = (R join S) join T (Associative)
  - (R join S) = (S join R) (Commute)
- Show that: R join (S join T) = (T join R) join S
More Equivalences

- A projection commutes with a selection that only uses attributes retained by the projection.
- Selection between attributes of the two arguments of a cross-product converts cross-product to a join.
- A selection on just attributes of \( R \) commutes with \( R \) join \( S \). (i.e., \( \sigma(R \text{ join } S) = \sigma(R) \text{ join } S \))
- Similarly, if a projection follows a join \( R \) join \( S \), we can 'push' it by retaining only attributes of \( R \) (and \( S \)) that are needed for the join or are kept by the projection.

Enumeration of Alternative Plans

- There are two main cases:
  - Single-relation plans
  - Multiple-relation plans
- For queries over a single relation, queries consist of a combination of selects, projects, and aggregate ops:
  - Each available access path (file scan / index) is considered, and the one with the least estimated cost is chosen.
  - The different operations are essentially carried out together (e.g., if an index is used for a selection, projection is done for each retrieved tuple, and the resulting tuples are pipelined into the aggregate computation).

Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate cost of each operation in plan tree.
    - Depends on input cardinalities.
    - We've already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - Must also estimate size of result for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.

Cost Estimates for Single-Relation Plans

- Index I on primary key matches selection:
  - Cost is \( \text{Height}(I)+1 \) for a B+ tree, about 1.2 for hash index.
- Clustered index I matching one or more selects:
  - \( (\text{NPages}(I)+\text{NPages}(R)) \times \text{prod. of RF’s of matching selects} \)
- Unclustered index I matching one or more selects:
  - \( (\text{NPages}(I)+\text{NTuples}(R)) \times \text{prod. of RF’s of matching selects} \)
- Sequential scan of file:
  - \( \text{NPages}(R) \).

  **Note:** Typically, no duplicate elimination on projections! (Exception: Done on answers if user says `DISTINCT`.)

Example

- If we have an index on `rating`:
  - \( (1/\text{NKeys}(I)) \times \text{Ntuples}(R) = (1/10) \times 40000 \) tuples retrieved.
  - Clustered index: \( (1/\text{NKeys}(I)) \times (\text{NPages}(I)+\text{NPages}(R)) = (1/10) \times (50+500) \) pages are retrieved. (The cost.)
  - Unclustered index: \( (1/\text{NKeys}(I)) \times (\text{NPages}(I)+\text{NTuples}(R)) = (1/10) \times (50+40000) \) pages are retrieved.
- If we have an index on `sid`:
  - Would have to retrieve all tuples/pages. With a clustered index, the cost is 50+500, with unclustered index, 50+40000.
- Doing a file scan:
  - We retrieve all file pages (500).

Queries Over Multiple Relations

- Fundamental decision in System R: only left-deep join trees are considered.
  - As the number of joins increases, the number of alternative plans grows rapidly; we need to restrict the search space.
  - Left-deep trees allow us to generate all fully pipelined plans.
    - Intermediate results not written to temporary files.
    - Not all left-deep trees are fully pipelined (e.g., SM join).
Multirelation Plans

- Left-deep plans differ only in the order of relations, the access method for each relation, and the join method for each join.
- Enumerated using N passes (if N relations joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. (All 2-relation plans.)
  - Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N’th relation. (All N-relation plans.)
- For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each interesting order of the tuples.

Cost Estimation for Multirelation Plans

- Consider a query block:
- Maximum # tuples in result is the product of the cardinalities of relations in the FROM clause.
- Reduction factor (RF) associated with each term reflects the impact of the term in reducing result size. Result cardinality = Max # tuples * product of all RF’s.
- Multirelation plans are built up by joining one new relation at a time.
  - Cost of join method, plus estimation of join cardinality gives us both cost estimate and result size estimate.

Example

- Pass 1:
  - Sailors: B+ tree on rating, Hash on sid
  - Reserves: B+ tree on bid
- Pass 2:
  - We consider each plan retained from Pass 1 as the outer, and consider how to join it with the (only) other relation.
    - e.g., Reserves as outer - Hash index can be used to get Sailors tuples that satisfy sid = outer tuple’s sid value.

Summary

- Query optimization is an important task in a relational DBMS.
- Must understand optimization in order to understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).
- Two parts to optimizing a query:
  - Consider a set of alternative plans:
    - Must prune search space; typically, left-deep plans only.
    - Must estimate cost of each plan that is considered.
    - Must estimate size of result and cost for each plan node.
  - Key issue: Statistics, indexes, operator implementations.
Summary (Cont.)

- Single-relation queries:
  - All access paths considered, cheapest is chosen.
  - Issues: Selections that match index, whether index key has all needed fields and/or provides tuples in a desired order.

- Multiple-relation queries:
  - All single-relation plans are first enumerated.
  - Selections/projections considered as early as possible.
  - Next, for each 1-relation plan, all ways of joining another relation (as inner) are considered.
  - Next, for each 2-relation plan that is 'retained', all ways of joining another relation (as inner) are considered, etc.
  - At each level, for each subset of relations, only best plan for each interesting order of tuples is 'retained'.