Introduction to Query Optimization

Chapter 13

Overview of Query Optimization

- Plan: Tree of R.A. ops, with choice of alg for each op.
- Each operator typically implemented as 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.

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 held in a temporary relation.
  - Cartesian products are avoided.

Schema for Examples

Sailors (id integer, name string, rating integer, age real) Reserve (id integer, bid integer, day date, name string)

Similar to old schema, name added for variations.

Reserves:
- Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.

Sailors:
- Each tuple is 30 bytes long, 80 tuples per page, 500 pages.

Motivating Example

```
SELECT S.name
FROM Reserves R, Sailors S
WHERE R.id = S.id AND R.bid = 100 AND S.rating > 5
```

- Cost: 300-500 hours/Os
- By no means the worst plan!
  - Plan: \( \pi_{name} \) (On-the-fly)
  - Includes several opportunities:
    - "pushed" earlier, no use of any available indexes, etc.
  - Goal of optimization: To find more efficient plans that compute the same answer.

Alternative Plans 1 (No Indexes)

```
\begin{array}{c}
\text{Reserves} \\
\text{Sailors}
\end{array}
```

- **Main difference:** push selects
- With 5 buffers, cost of plan:
  - Scan Reserves (1000) + write temp T1 (10 pages, if we have 100 boats, uniform distribution).
  - Scan Sailors (300) + write temp T2 (250 pages, if we have 30 ratings).
  - Sort T1 (2^30), sort T2 (2^320), merge (10-250).
  - Total: 3560 page I/Os.
- If we used BNL join cost = 10 + 4*250, total cost = 2700.
- If we push projections, T1 has only sid, T2 only sid and name;
  - T1 fits in 3 pages, cost of BNL drops to under 250 pages, total < 2000.
Alternative Plans 2
With Indexes

- With clustered index on sid of Reserve, we get 100,000/100 = 1,000 tuples on 1,000/100 = 10 pages.
- InL with pipelining (outer is not materialized).
  - Projecting out unnecessary fields from outer doesn’t help.
  - Join columns ‘sid’ is a key for Sailors.
  - At most one matching tuple, clustered index on sid/OK.
  - Decision not to push on index before the join is based on availability of sid index on Sailors.
- Cost: Selection of Reserve tuples (101/Os); for each
  must get matching Sailors tuple (100/1.2); total 1,210/Os.

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, join, etc.)
  - Must estimate size of result for each operation in tree.
  - Use information about the input relations.
  - For selections and joins, assume independence of predicates.
- We’ll discuss the System R cost estimation approach:
  - Very inexact, but works ok in practice.
  - More sophisticated techniques known now.

Statistics and Catalogs

- Need information about the relations and indexes
  involved. **Catalogs** typically contain at least:
  - # tuples (Ntuples) and # pages (Npages) for each relation.
  - # distinct key values (NKeys) and NPages for each index.
  - Index height: low/high key values (Low/High) for each tree index.
- Catalogs updated periodically.
  - Updating whenever data changes is too expensive lots of approximations anyway, so slight inconsistency ok.
  - More detailed information (e.g., histograms of the values in some field) are sometimes stored.

Size Estimation and Reduction Factors

- Consider a query block:
  
  \[
  \text{SELECT attribute list FROM relation list WHERE terml AND ... AND termn}
  \]
- 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 = \( \text{Max # tuples} \times \text{product of all RFs} \).
  - Implicit assumption that terms are independent!
  - Term (const = value) has RF 1 (NKeysI), given index on col
  - Term (null = null) has RF \( \frac{1}{\text{Max}(NKeysI), NKeysI^2)} \)
  - Term (not) has RF \( \frac{9}{(HighH-value)/HighH-LowH)} \)

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 preserve each space typically, left-deep plans only.
    - Must estimate cost of each plan that is considered.
  - Key issues: Statistics, indexes, operator implementations.