Relational Query Optimization

Yanlei Diao
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

**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):
  - \((\text{day}<8/9/94 \text{ OR } \text{bid}=5 \text{ OR } \text{sid}=3) \text{ AND } (\text{rname}=\text{‘Paul’} \text{ OR } \text{sid}=3)\)

- Interleave FROM and WHERE into an operator tree for optimization.
  - Query optimization largely works for Conjunctive Queries (only).

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

- Query plans and equivalences

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

- Handling nested queries

- Multi-objective optimization in Cloud Computing
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):

\[ \pi_{\text{sname}} \left( \sigma_{\text{bid}=100 \land \text{rating}>5} \left( \text{Reserves} \bowtie \text{Sailors} \right) \right) \]
Query Evaluation Plan

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

- What are the missed opportunities?
  - Selections could have been `pushed’ earlier.
  - Use of indexes.
  - More efficient joins.
Relational Algebra Equivalences

- **Selections:**
  \[ \sigma_{c_1 \land \ldots \land c_n}(R) \equiv \sigma_{c_1}(\ldots \sigma_{c_n}(R)) \]  
  (Cascade)
  \[ \sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R)) \]  
  (Commute)

- **Projections:**
  \[ \pi_{a_1}(R) \equiv \pi_{a_1}(\ldots(\pi_{a_1,\ldots,a_n}(R))) \]  
  (Cascade)

- **Joins:**
  \[ (R \bowtie S) \equiv (S \bowtie R) \]  
  (Commute)
  \[ R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \]  
  (Associative)

☞ **Show that:**
\[ R \bowtie (S \bowtie T) \equiv (T \bowtie R) \bowtie S \]
More Equivalences

\[ \sigma_c (R \times S) \equiv R \bowtie c S \]

\[ \sigma_c (R \bowtie S) \equiv \sigma_c (R) \bowtie S, \text{ if } c \text{ is only applied to } R \]

\[ \pi_a (\sigma_c (R)) \equiv \sigma_c (\pi_a (R)) \text{ holds if } \sigma \text{ only uses attributes retained by } \pi \]

\[ \text{For } \pi_b (R \bowtie a S), \text{ we can ‘push’ } \pi \text{ before } \bowtie \text{ by retaining both the } a \text{ attribute and the } b \text{ attribute (if existent)} \]

\[ \text{But, aggregates do not commute with other operators.} \]
Schema for Examples

Sailors (\textit{sid}: integer, \textit{sname}: string, \textit{rating}: integer, \textit{age}: real)
Reserves (\textit{sid}: integer, \textit{bid}: integer, \textit{day}: dates, \textit{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.
Query Plan 1 (Selection Pushed Down)

- Push selections below the join.

- **Materialization vs. Pipelining:**
  - **Materialization**: Materialize a temporary relation T, if the next operator needs to scan T multiple times.
  - **Pipelining**: the opposite.

- **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.
**Indexes**

- A **tree** index *matches* (a conjunction of) terms if the attributes in the terms form a *prefix* of the search key.
  - Tree index on \(<a, b, c>\)
  - \(a=5\) AND \(b=3\) ?
  - \(a=5\) AND \(b>6\) ?
  - \(b=3\) ?
Query Plan 3 (Using Indexes)

- **Selection using index**: clustered index on *bid* of Reserves.
  - Retrieve 100,000/100 = 1000 tuples
  - Clustering: read 1000/100 = 10 pages.

- **Indexed NLJ**: *pipeline* the outer and *index lookup* on *sid* of Sailors.
  - The outer: no need to materialize.
  - The inner: *sid* is a key; at most one match tuple, unclustered index OK.

- **Cost**:
  - Selection of Reserves tuples (~10 I/Os).
  - For each tuple, get matching Sailor tuple (1000*(2~3)).
  - Total = 2010~3010 I/Os.
Outline

- Query plans and equivalences

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

- Handling nested queries

- Multi-objective optimization in Cloud Computing
An SQL query is parsed into a collection of **query blocks**, and these are optimized one block at a time.

Nested blocks are usually treated as calls to a subroutine, made once per outer tuple. (Optimization is advanced material...)

```sql
SELECT S.sname
FROM   Sailors S
WHERE  S.age IN
       (SELECT MAX (S2.age)
        FROM   Sailors S2
        GROUP BY S2.rating)
```
Three Main Issues in Optimization

Given a query block, three main optimization issues:

- **Plan cost**: what is the cost of a given plan?
- **Plan space**: which plans are considered?
- **Search algorithm**: how do we search the plan space for the cheapest estimated plan?
- We will learn the design of *System R Optimizer*
(1) **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 factor (RF) / selectivity* of each term, which reflects the impact of the *term* in reducing result size.

```sql
SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk
```
Statistics in System Catalog

- Statistics about each relation (R) and index (I):
  - **Relation cardinality**: # tuples (NTuples) in R
  - **Relation size**: # pages (NPages) in R
  - **Index cardinality**: # distinct values (NKeys) in I
  - **Index size**: # leaf pages (INPages) in I
  - **Index height**: # nonleaf levels (IHeight) of I
  - **Index range**: low/high key values (Low/High) in I
  - **Number of distinct values** in an attribute (NKeys)
  - **Histogram** for an attribute
Cost Estimates for Single-Relation Plans

- **Index I on primary key** matches selection:
  - Cost of lookup = \( \text{Height}(I)+1 \) for a B+ tree, \( \approx 1.2 \) for hash index
  - Cost of record retrieval = 1

- **Clustered index** I matching one or more selections:
  - Cost of lookup + \( \text{product of RF 's of matching terms (RF-terms)} \times (\text{INPages}(I)+\text{NPages}(R)) \)

- **Non-clustered index** I matching one or more selections:
  - Cost of lookup + RF-terms * \( \text{INPages}(I) \) + \( \min(\text{RF-terms} \times \text{NTuples}(R), \text{NPages}(R)) \)

- **Sequential scan** of file: \( \text{NPages}(R) \)

- May add extra costs for GROUP BY, sorting, and duplicate elimination (if a query says DISTINCT)
Reduction Factors

- Reduction factor (RF) or Selectivity of each term reflects the impact of the term in reducing result size.
  - Assumption 1: uniform distribution of the values!
  - Term \( \text{col} = \text{value} \): RF = \( 1/N\text{Keys(I)} \), if there is an index I on \( \text{col} \).
  - Term \( \text{col} > \text{value} \): RF = \( (\text{High}(I) - \text{value})/(\text{High}(I) - \text{Low}(I)) \)
  - Term \( \text{R.col1} = \text{S.col2} \):
    1) If \( \text{R.col1} \) is a foreign key, \( \text{S.col2} \) is a primary key, then RF = \( 1/\text{NTuples(S)} \)
    2) Otherwise, RF = \( 1/\text{MAX}(N\text{Keys(I1)}, N\text{Keys(I2)}) \)
       - If there are indexes I1 and I2 on \( \text{col 1} \) and \( \text{col 2} \), respectively
         - Each value from R, which is supposed to be in the smaller index I1, has a matching value in S with the larger index I2.
         - Values in S are evenly distributed.
         - So each R tuple has \( \text{NTuples(S)}/\text{NKeys(I2)} \) matches, a RF of \( 1/\text{NKeys(I2)} \).
         - If there is only one index I, \( 1/\text{NKeys(I)} \).

```sql
SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk
```
Illustration for $R.col1 = S.col2$