Query Optimization
Part III

CPS 216
Advanced Database Systems

Announcements

- Reading assignment this week
  - "Randomized Algorithms for Optimizing Large Join Queries," by Ioannidis & Kang. SIGMOD 1990
  - "Online Aggregation," by Hellerstein et al. SIGMOD 1997
- Homework #3 due in 2 days (Wednesday, April 9)
- Homework #4 out in 2 days (Wednesday, April 9)
- Project milestone #2 due in 7 days (Monday, April 14)

Review of the bigger picture

Query optimization

- Consider a space of possible plans
- Estimate costs of plans in the search space
- Search through the space for the "best" plan (today)

- Focus on select-project-join query blocks
  - Join ordering is the most important subproblem
Search space

- "Bushy" plan example:

- Search space is huge: 30240 bushy plans for a six-table join
- More if we consider:
  - Multiway joins
  - Different join methods
  - Placement of selection and projection operators

Left-deep plans

- Heuristic: consider only "left-deep" plans, in which only the left child can be a join
- How many left-deep plans are there for $R_1 \bowtie \cdots \bowtie R_n$?

A greedy algorithm

- $S_1, \ldots, S_n$
  - Say selections have been pushed down; i.e., $S_j = \sigma_{R_i}$
  - Start with the pair $S_j$, $S_j$ with the smallest estimated size for $S_j \bowtie S_j$
  - Repeat until no table is left:
    - Pick $S_j$ from the remaining tables such that the join of $S_j$ and the current result yields an intermediate result of the smallest size
    - Pick most efficient join method
    - Minimize expected size
    - Current subplan
    - Remaining tables to be joined

Complexity?
Query optimization in System R

- A.k.a. Selinger-style query optimization
  - The classic paper on query optimization (Selinger et al., SIGMOD 1979)

- Basic ideas
  - Left-deep trees only
  - Bottom-up generation of plans using dynamic programming
  - "Interesting orders"

Bottom-up plan generation

- Observation 1: Once we have joined $k$ tables together, the method of joining this result further with another table is independent of the previous join methods
- Observation 2: Any subplan of an optimal plan must also be optimal (otherwise we could replace the subplan to get a better overall plan)
  - Not exactly accurate (next slide)

- Bottom-up generation of optimal left-deep plans
  - Compute the optimal plans for joining $k$ tables together
    - Suboptimal plans are pruned
  - From these plans, derive optimal plans for joining $k + 1$ tables

The need for “interesting order”

- Example: $R(A, B) \bowtie S(A, C) \bowtie T(A, D)$
- Best plan for $R \bowtie S$: nested-loop join (beats sort-merge)
- Best overall plan: sort-merge join $R$ and $S$, and then sort-merge join with $T$
  - Subplan of the optimal plan is not optimal!
- Why?
  - The result of the sort-merge join of $R$ and $S$ is sorted on $A$
  - This is an interesting order that can be exploited by later processing (e.g., join, duplicate elimination, GROUP BY, ORDER BY, etc.).
### Dealing with interesting orders

- When picking the best plan
  - Comparing their costs is not enough
  - Plans are not totally ordered by cost anymore
  - Comparing interesting orders is also needed
    - Plans are now partially ordered
    - Plan X is better than plan Y if
      - Cost of X is lower than Y
      - Interesting orders produced by X subsume those produced by Y
- Need to keep a set of optimal plans for joining every combination of k tables
  - At most one for each interesting order

### System-R algorithm

- Pass 1: Find the best single-table plans
- Pass 2: Find the best two-table plans by considering each single-table plan (from Pass 1) as the outer input and every other table as the inner input
- ...
- Pass k: Find the best k-table plans by considering each (k-1)-table plan (from Pass k-1) as the outer input and every other table as the inner input
- ...
- Heuristics
  - Push selections and projections down
  - Process cross products at the end

### Reasoning about predicates

- `SELECT * FROM R, S, T
  WHERE R.A = S.A AND S.A = T.A;`
- Looks like a cross product between R and T
- But there is really a join between R and T
- A good optimizer should be able to detect this case and consider the possibility of joining R with T first
System-R algorithm example

- **SELECT SID, CID**
  FROM Student, Enroll, Course
  WHERE Student.age < 10
  AND Student.SID = Enroll.SID
  AND Enroll.CID = Course.CID
  AND Course.title LIKE '%data%';

- Primary keys/indexes
  - Student(SID), Enroll(CID, SID), Course(CID)

- Ordered, secondary indexes
  - Student(age), Course(title)

Example: pass 1

- Plans for {Student}
  - S1: Table scan, then filter (age < 10); cost 100; result ordered by SID
  - S2: Index scan using condition (age < 10); cost 5; result ordered by age

- Plans for {Enroll}
  - E1: Table scan; cost 1000; result ordered by CID, SID ← interesting order

- Plans for {Course}
  - C1: Table scan, then filter (title LIKE '%data%'); cost 40; result ordered by CID ← interesting order
  - C2: Index scan, then filter (title LIKE '%data%'); cost 60; result ordered by title ← not an interesting order

Example: pass 2

- Plans for {Student, Enroll}
  - Extending best plans for {Student}
    - From S1 (table scan, then filter (age < 10))
      - Block-based nested loop join with Enroll; cost 1100
      - Sort Enroll by SID, and merge join; cost 5100; ordered by SID ← no longer an interesting order
    - From S2 (index scan using condition (age < 10))
      - Block-based nested loop join with Enroll; cost 1005
  - Extending best plans for {Enroll} ...
Example: pass 2 continued

- Plans for \{Student, Course\}
  - Ignore; it is a cross product
- Plans for \{Enroll, Course\}
  - Extending best plans for \{Course\}
    - From C1 (table scan, then filter (title LIKE '%data%'))
      - Merge join; cost 1040
  - Extending best plans for \{Enroll\}

- Plans for \{Student, Enroll\}

- Plans for \{Student, Course\}
  - None!
- Plans for \{Enroll, Course\}
  - (FILTER(Course) SMJ Enroll) NLJ (INDEX-SCAN(Student));
    - cost ...

- Plans for \{Student, Enroll, Course\}

Example: pass 3

Finally, plans for \{Student, Enroll, Course\}

- Extending best plans for \{Student, Enroll\}
  - (INDEX-SCAN(Student) NIJ Enroll) NIJ FILTER(Course);
    - cost ...
  - ...
- Extending best plans for \{Student, Course\}
  - None!
- Extending best plans for \{Enroll, Course\}
  - (FILTER(Course) SMJ Enroll) NLJ (INDEX-SCAN(Student));
    - cost ...
  - ...

Considering bushy plans

- Store all optimal 1-table, 2-table, ..., and \(k\)-table plans
- To find the optimal plan for \(k+1\) tables
  - For every possible partition of these tables into two groups, find the best ways of joining the optimal plans for the two groups
  - Store the overall optimal plans
Optimizer “blow-up”

- A 20-way join will easily choke an optimizer using the System-R algorithm

- Solutions
  - Heuristics-based query optimization
  - Randomized query optimization (Ioannidis & Kang, SIGMOD 1990)
  - Genetic programming (PostgreSQL)

Search space revisited

![Diagram](image)

Transformations

Relational algebra equivalences (or query rewrite rules in general):

- Join method choice: \( R \bowtie_{\text{method}_1} S \rightarrow R \bowtie_{\text{method}_2} S \)
- Join commutativity: \( R \bowtie S \rightarrow S \bowtie R \)
- Join associativity: \( (R \bowtie S) \bowtie T \rightarrow R \bowtie (S \bowtie T) \)
- Left join exchange: \( (R \bowtie S) \bowtie T \rightarrow R \bowtie (T \bowtie S) \)
- Right join exchange: \( R \bowtie (S \bowtie T) \rightarrow S \bowtie (R \bowtie T) \)

- Why the last two redundant rules?
  - “Shortcuts” to avoid using the join commutativity rule, which does not change the cost of certain joins (example?)—creating plateaus in the plan space
Iterative improvement

- Repeat until some stopping condition (e.g., time runs out):
  - Start with a random plan
  - Repeatedly go downhill (i.e., pick a neighbor with a lower cost randomly) to get to a local optimum
- Return the smallest local optimum found

Simulated annealing

- Start with a plan and an initial temperature
- Repeat until temperature is 0:
  - Repeat until some equilibrium (e.g., a fixed number of iterations):
    - Move to a random neighbor of the plan (an uphill move is allowed with probability \( e^{-\frac{\Delta \text{cost}}{\text{temperature}}} \))
      - Larger \( \rightarrow \) smaller probability
      - Lower temperature \( \rightarrow \) smaller probability
    - Reduce temperature
- Return the plan visited with the lowest cost

Two-phase optimization

- Phase I: run iterative improvement for a while to find a good local optimum
- Phase II: run simulated annealing with a low initial temperature to get more improvements

Why does this heuristic tend to work better than both iterative improvement and simulated annealing?
Shape of the cost function

- An average local optimum has a much lower cost than an average plan
- The average distance between a random state and a local optimum is long
- There are lots of local optima
- Many local optima are connected together through low-cost plans within short distances

Comparison of randomized algorithms

- Iterative improvement
- Simulated annealing
- Two-phase
  - Phase I uses iterative improvement to get to the cup bottom quickly
  - Phase II uses simulated annealing to explore the cup bottom further