More Cost Based Plan Selection and Joins

CS157B

Chris Pollett

Apr.4, 2005.

Outline

- Intro to Cost-Based Plan Selection
- Choosing Join Order

Introduction to Cost-Based Plan Selection

- Number of I/Os of the plan we choose influenced by:
  - The particular logical operators chosen to implement the query
  - The sizes of intermediate relations
  - The physical operators used to implement the logical operators
  - The ordering of similar operations such as joins.
  - The method of passing arguments from one physical operator to another.

Obtaining Estimates for Size Parameters

- DBMS stores values of V(R,A) in catalog.
- This is not always updated, but can be explicitly updated by the DBA.
- For example, one can use ANALYZE TABLE blah COMPUTE STATISTICS; in Oracle.
- DBMS may compute a histogram rather than just V(R,A). That is, a plot of a range of values for an attribute versus number of objects in that range.
- Above is equal-width, can also compute equal height statistics, and most frequent value statistics.
- Most statistics can be computed by a table scan

Incremental Computation of Statistics

- Once statistics have been computed, we want to be able to maintain them without having to recompute everything.
- For example, to maintain T(R), we can just add +1 on an insert and -1 on a delete.
- If there is a B-tree index on any attribute R, we could also estimate T(R) using the number of leaves in B-tree and assuming each is 3/4 full. (Don't have to change each insert.)
- If there in an index on a, then V(R,a), can also be calculated as we insert/delete from the index.
- In the particular case that a is key, then T(R)=V(R,a).
- In no index, might use a rudimentary structure that holds each value of a.
- Might also sample data and assume data is according to some distribution: uniform, zipfian, etc.

Heuristics for Reducing the Cost of Logical Query Plans

- Might try things like pushing selects down the tree.
- As the preferred logical query plan is being generated, consider plans generated by applying several such heuristics.
- Then compute cost before and after each transformation. If the transformation doesn't help we don't do it.

Approaches to Enumerating Physical Plans

- Two standard approaches:
  - **Top down:** work from the root down. For each possible implementation of the operation at the root, consider each possible way to evaluate its arguments. Compute the cost of each possibility and choose the least.
  - **Bottom up:** For each subexpression, compute the cost of all possible ways to compute that subexpression. Choose the one of least cost.
- We'll focus on bottom up approach.

More on Enumerating Physical plans

- **Heuristic selection** - try different heuristic to improve cost. Ex: Try using an index if doing a select. If index on join attribute, try using an index join. If one attribute of join sorted, try using a sort join. Group smallest relations first when doing set operations.
- **Branch and Bound Plan Enumeration** - Use heuristics to first find a good plan, say with cost C. Then enumerate plans for subexpressions. If any cost for a subexpression is above C, can discard all plans that involve this way to compute the subexpression. Hopefully, using this can speed up over exhaustive search.
- **Hill Climbing** - make small local changes to plan, if this improves cose switch. If no small change in plan yields an improvement above a given amount then output plan.
- **Dynamic Programming** - say more about in a minute, mainly for join orderings.
- **Selinger-Style Optimization** - modifies dynamic programming. Keeps non-low cost subexpressions which might make higher up in tree faster. For example, subexpression might computed in sorted order.

Join Order- Significance of Left and Right Join Arguments

- When ordering joins, we should remember our algorithms are often asymmetric in terms of cost on A join B versus B join A:
  - Nested-loop join. Here want smaller relation as outer loop.
  - Index-join - right argument assumed to be the one with the index.

Join Trees

- Consider (A join B join C). We could order this as:
  - ((A join B) join C) or as
  - (A join (B join C))
- Many ways to take join of n things.
- Would take too long if we had to enumerate all of them.

Left-Deep Join Trees

- An ordering like (((A1 join A2) join A3) …) is called a **left deep ordering**.
- Note (((A3 join A1) join A2)…) computes same join and is also left deep.
- One can define a **right deep ordering** in a similar fashion.
- Even if we allow non-join operations, we might still call it a left-deep ordering, provided binary operators are parenthesized in above way.
- In general, there is only one left-deep tree shape for n relations. However, one is able to put the relations into this shape in n! ways.
- Number of tree shapes given by T(1)=1, T(n) = Sum^{n-1}_{i=1}T(i)T(n-i)

Some Advantages of Left Deep Join Trees

- They tend to produce efficient plans because:
  - If one pass join algorithms are used, and the build relation is on the left, then the amount of memory needed at any one time tends to be smaller than if used a right -deep or bushy tree.
  - If nested-loop joins are used, implemented by iterators, then we avoid having to construct any intermediate relation more than once.