Joins, Completing Plans, System Failures

CS157B

Chris Pollett

Apr.6, 2005.

Outline

- Dynamic Programming for Join Order
- Completing the Physical-Query-Plan Selection
- Types System Failures

Dynamic Programming for Join Order

- Suppose we want to compute (R_1 join R_2 join … join R_n)
- We can build a table that contains:
  - Each table $R_{i,}$ as well as its size and number of values for the join attribute
  - For each subset of $R_{i\_1\ldots}$ R_{i_j} of the tables of size j an estimate of the total cost of performing the join of this subset.
- The second kinds of entries are built from the best joins of subsets j-1 tables by going through these entries and joining one more table. Then check if this j-subset is already in the table. If it is compare its currently listed cost with the new way of joining. If the new way is smaller than replace.

Completing the Physical-Query

- To finish up creating a query plan, we need to consider the following issues:
  - How to select algorithms to implement the operations of the query plan not done as part of some earlier step as in the dynamic programming for join order
  - Decide whether to materialize intermediate results or pipeline them
  - Notations for physical plan operators

Choosing a selection method

- What way should we use to evaluate $\sigma_C(R)$?
- Assuming there are no multidimensional attributes we can, if possible:
  - Choose an attribute in C, with an index, and which is compared to a constant
  - After using the index to find those things satisfying this part of C, apply the rest of the condition to complete the selection.
- If no such attribute exists need to do a table-scan
- In comparing costs want to choose most selective single indexed attribute

Choosing a Join Method

- Request a nested loop join, if one can hope that the outer table can fit totally in memory (so maybe don’t need to execute outer loop more than once).
- Use a sort-join when one or both tables is already sorted on the join attribute. Or, if there are two or more tables joined on the same attribute (as then sort step can be amortized).
- If there is an index on the join attribute and the results are expected to be small use an index join
- Fall back to nested loop join if none of the above apply.

Pipelining Versus Materialization

- Naïve way to evaluate queries is to write intermediate results back out to disk. This is called *materialization*.
- *Pipelining* is a technique whereby we pass the output of the subquery computed so far to the next stage in the query evaluation. It is typically implemented using iterators.

Notations for Physical Query Plans

- Each node in logical query plan needs to be replaced by a physical operator.
- Operators for leave:
  - TableScan(R)
  - SortScan(R,L) - sort according to list L
  - IndexScan(R,C) - look up index according to condition C
  - IndexScan(R,A) - entire relation is retrieved via lookup on attribute A.
- Operators for Selection:
  - Filter(C) -- filter child results according to C.
  - Some select might be an IndexScan followed by a Filter
- Operator for Sorting:
  - Sort(L) -- may occur anywhere interior to tree.
- Other operators
  - Join(type, # of buffers) - for example type might be two pass hash-join, and number of buffers might be 100

Ordering of Physical Operations

- Physical query plan is typically represented as a tree. Data must flow up the tree.
- Since interiors node may not be ancestors or descendents of each other in a bushy tree, we need to figure out an order to evaluate subtrees of the same height.
- To do this we evaluate subtrees from the bottom up and evaluate subtrees of the same height from left to right. (Preorder)
- Execute all nodes of each subtree using a network of iterators.

System Failures

- Would like our DBMS to:
  - protect data in the case of system failure.
  - keep the database in a consistent state even if multiple operations are being carried out at the same time.

Possible Kind of Errors

- Errorneous Data Entry -- can try to prevent by type checking
- Media Failures -can try to prevent by using a RAID system or archiving data to tape
- Catastrophic failures -- fires. Can try to distribute the database.
- System failures -- power loss or software errors that cause a particular transaction to be lost. -- What we are going to focus for the next several lectures