CS632 Final Exam Part C

A. Demers

7 May 2003

Here is the last piece of the final – the promised question on query processing.

As usual, you may consult others for ideas and proof approaches, but please *reference your sources* and write up answers independently.

4 Query Processing

This is a squishy "thought question" to expose some of the over-simplification hidden in our discussion of query optimizers.

In a production database system, most queries may be precompiled rather than *ad hoc*. Such a query has zero or more *parameters*. The query is compiled and optimized once, then executed repeatedly with different values of the parameters. The salary updating transaction from part (1e) is an example of this. The value of the parameter w is supplied at execution time; it is not available to the optimizer.

An obvious argument in favor of this approach is that query optimization is expensive, and precompiling queries should allow the cost of the optimizer to be amortized over a large number of query evaluations.

Unfortunately, things are not quite so simple. In general, the execution cost of a query depends strongly on the parameter values; it also depends on the number of memory buffers made available when the query runs.

> It also depends on the buffer manager's replacement policy (LRU, FIFO, LIFO, ...) but at this level of detail we ignore the replacement policy, assuming the buffer manager never does something *stupid* like evicting a page when there are empty buffers available.
>
> We also ignore the effect of running more than one query concurrently. This too is unrealistic – in many real systems there are "hot"

tables, used by a large fraction of the queries, which essentially live in the shared buffer pool – but it simplifies the discussion to ignore this effect.

To summarize, we are computing the cost in I/O operations of a single query running in isolation, as a function of its parameter values $p_1, \ldots, p_k$ and the buffer pool size B.

Problem 4a: First consider varying only the parameter values, letting each query run in an infinite buffer pool. Give an example of a parameterized conjunctive query (and the database instance it is applied to) for which at least three different query plans are optimal depending on the parameter values.

In addition to specifying the relation contents and what indices exist, don't forget to specify important physical properties like the size of the relation(s), the number of tuples or index entries per disk block, whether indices are clustered or unclustered, and so forth.

Can you extend your example to arbitrarily large values of three – that is, for any n can you construct an example query for which n distinct query plans are optimal, depending on parameter values? How fast do the query, database schema and/or database instance sizes grow as a function of n? (Hint: I don't know the best answer to this last part).

Problem 4b: Now consider varying the size of the available buffer pool, keeping parameter values constant.

Every query plan has a *required space*, which is the least number of buffers B with which it is feasible to execute the plan at all, and a *maximum cost*, which is the cost of executing the plan in its required space. In the I/O cost model a couple of things are obvious:

The execution cost decreases monotonically with B (this justifies the terminology "maximum cost" above).

In the limit as $B \to \infty$, the cost of any query plan is at most the sum of the sizes of its leaf relations (why?).

Now every query plan has an *unconstrained cost*, which is the cost of evaluating it with an infinite, initially-empty buffer pool, and an *optimum space*, which is the least number of buffers B for which the plan achieves its unconstrained cost.

What are the values of the above parameters for the following query processing algorithms:

Nested loop join of relations A and B.

Index nested loop join of relations A and B using an unclustered B+ tree index on B.

Select by full table scan of relation A.

Select by range query using (clustered or unclustered) B+ tree index on A.

State any assumptions you need to make about physical properties of the database, selectivities, etc.

Problem 4c: Repeat (4a) but for buffer pool size rather than query parameter. That is, for various values of n give examples for which n different query plans are optimal, depending on the buffer pool size B. How does the size of the example (query, schema, instance, indices) depend on n? (The same hint applies: I don't know the optimal answer).