Query Optimization

Lab 2 due today

Recap: last time saw join algorithms

Selinger

Famous paper. Pat Selinger was one of the early System R researchers; still active today.

Lays the foundation for modern query optimization. Some things are weak but have since been improved upon.

Idea behind query optimization:
   (Find query plan of minimum cost )

How to do this?
   (Need a way to measure cost of a plan (a cost model) )

Single table operations

How do I compute the cost of a particular predicate? (Compute the size of its input)

How do I estimate the size of an operators input? (From stats over base tables, or using
   "selectivity" - fraction F of tuples -- it's children passed)

How does Selinger estimate size of base tables? -- Using some (simple) statistics :
   - NCARD(R) - "relation cardinality" -- number of tuples in R
   - TCARD(R) - # pages R occupies
   - ICARD(I) - keys (distinct values) in index I
   - NINDEX(I) - pages occupied by index I
   - min and max keys in indexes

(have to realize that the complexity of statistics you could keep in 1978 was pretty simple!)

How does Selinger estimate selectivity F:

col = val
   F = 1/ICARD() (if index available)
   F = 1/10 (where does this come from?)

col > val
   (max key - value) / (max key - min key) (if index available)
   1/3 o.w.

col1 = col2
   1/\text{MAX}(ICARD(col1, col2))
   1/10 o.w.

Example: suppose emp has 1000 records, dept has 10 records
Total records is 1000 \times 10, selectivity is 1/1000, so 10 tuples expected to pass join
(note that this is wrong if doing key/fk join on emp.did = dept.did, which will produce 1000 results!)

Note that selectivity is defined relative to size of cross product for joins!
p1 and p2

\[ F_1 \times F_2 \]

p1 or p2

\[ 1 - (1-F_1) \times (1-F_2) \]

**Estimating the cost of single table operations**

How is cost defined?
(in terms of number of pages read + a weighted factor of # predicate evals)


(W is CPU cost per predicate eval in terms of fraction of a time to read a page)

Range scan:

Clustered index, boolean factors: \( F(\text{preds}) \times (\text{NINDX} + \text{TCARD}) + W\times(\text{tuples read}) \)

Unclustered index, boolean factors: \( F(\text{preds}) \times (\text{NINDX} + \text{NCARD}) + W\times(\text{tuples read}) \)

...unless all pages fit in buffer -- why?

Seq (segment) scan: \( \text{TCARD} + W\times(\text{NCARD}) \)

Is an index always better than a segment scan? (no)

**Multi-table operations**

How do I compute the cost of a particular join?

Algorithms:

\[ \text{NL}(A,B,\text{pred}) \]

\[ \text{Cost}(A) + \text{NCARD}(A) \times \text{Cost}(B) \]

Note that inner is always a relation; cost to access depends on access methods for B; e.g.,

w/ index -- 1 + 1 + W

w/out index -- TCARD(B) + W\times\text{NCARD}(B)

Cost(A) is cost of subtree under outer

How to estimate \# NCARD(outer)? product of F factors of children, cardinalities of children

example:

\[ \text{Merge-Join}_x(P,A,B), \text{equality pred} \]

\[ \text{Cost}(A) + \text{Cost}(B) + \text{sort cost} \]

(Saw cost models for these last time)
At time of paper, didn't believe hashing was a good idea

Overall plan cost is just sum of costs of all access methods and join operators
Then, need a way to enumerate plans

Iterate over plans, pick one of minimum cost

**Problem:**

Huge number of plans. Example:

suppose I am joining three relations, A, B, C
Can order them as:

(AB)C  
A(BC)  
(AC)B  
A(CB)  
(BA)C  
B(AC)  
(BC)A  
B(AC)  
(CA)B  
C(AB)  
(CB)A  
C(BA)

Is C(AB) different from (CA)B?  
Is (AB)C different from C(AB)?  
yes, inner vs. outer

n! strings * # of parenthetizations

How many parenthetizations are there?

Consider N=1,2,3,4:

- A: (A)  
- AB: ((A)(B))  
- ABC: ((AB)C), (A(BC))  
- ABCD: (((AB)(C))D), ((A(BC))D), ((AB)(CD)), (A((BC)D)), (A(B(CD)))

The numbers of plans for N=1,2,3,4 are:

- plans(1) = 1
- plans(2) = 1
- plans(3) = 2
- plans(4) = 5

(Some of these plans Selinger wouldn't consider because they aren't left deep)

Generally, plans(N) = choose(2(N-1),(N-1))/(N) *

* The Art of Computer Programming, Volume 4A, page 440-450

```plaintext
=> n! * choose(2(N-1),(N-1))/(N)
=> 4 choose 2 / 3 == 6 / 3 == 2
6 * 2 == 12 for 3 relations
```
(study break -- postgres)

Ok, so what does Selinger do?

Push down selections and projections to leaves
Now left with a bunch of joins to order.

Selinger simplifies using 2 heuristics? What are they?

- only left deep; e.g., ABCD => ((AB)(CD)) show
- ignore cross products

  e.g., if A and B don't have a join predicate, doing consider joining them

still n! orderings. can we just enumerate all of them?

10! -- 3million
20! -- 2.4 * 10^18

so how do we get around this?

Estimate cost by dynamic programming:

idea: if I compute join (ABC)DE -- I can find the best way to combine ABC and then consider all the ways to combine that with DE.

I can remember the best way to compute (ABC), and then I don't have to re-evaluate it. Best way to do ABC may be ACB, BCA, etc -- doesn't matter for purposes of this decision.

algorithm: compute optimal way to generate every sub-join of size 1, size 2, ... n (in that order).

R <--- set of relations to join
for δ in {1...|R|}:
  for S in {all length δ subsets of R}:
    optjoin(S) = a join (S-a), where a is the single relation that minimizes:
      cost(optjoin(S-a)) +
      min cost to join (S-a) to a +
      min. access cost for a

example: ABCD

only look at NL join for this example

A = best way to access A (e.g., sequential scan, or predicate pushdown into index...)
B = " " " " " B
C = " " " " " C
D = " " " " " D

{A,B} = AB or BA
{A,C} = AC or CA
{B,C} = BC or CB
{A,D}
{B,D}
{C,D}
\{A,B,C\} = \text{remove } A \hspace{1em} \text{compare } A(\{B,C\}) \text{ to } (\{B,C\})A \\
\text{remove } B \hspace{1em} \text{compare } (\{A,C\})B \text{ to } B(\{A,C\}) \\
\text{remove } C \hspace{1em} \text{compare } C(\{A,B\}) \text{ to } (\{A,B\})C \\
\{A,C,D\} \\
\{A,B,D\} \\
\{B,C,D\} \\
\{A,B,C,D\} = \text{remove } A \hspace{1em} \text{compare } A(\{B,C,D\}) \text{ to } (\{B,C,D\})A \\
\text{.... remove } B \\
\text{remove } C \\
\text{remove } D \\

\text{Complexity:}

\text{number of subsets of size 1 * work per subset} = W+ \\
\text{number of subsets of size 2 * W +} \\
\text{...} \\
\text{number of subsets of size } n \text{ * } W+ \\
\begin{array}{cccc}
n + n + n & \cdots & n \\
1 & 2 & 3 & n \\
\end{array} \\
\text{number of subsets of set of size } n \text{ = power set of } n = 2^n \\
\text{(string of length } n, 0 \text{ if element is in, } 1 \text{ if it is out; clearly, } 2^n \text{ such strings)} \\
\text{(reduced an } n! \text{ problem to a } 2^n \text{ problem)} \\

\text{what's } W? \hspace{1em} (\text{at most } n) \\
\text{so actual cost is: } 2^n \times n \\
n=12 \rightarrow 49K \text{ vs } 479M

\text{So what's the deal with sort orders? Why do we keep interesting sort orders?}

\text{Selinger says: although there may be a 'best' way to compute } ABC, \text{ there may also be ways that produce interesting orderings -- e.g., that make later joins cheaper or that avoid final sorts.} \\
\text{So we need to keep best way to compute } ABC \text{ for different possible sort orders.} \\
\text{so we multiply by "k" -- the number of interesting orders}

\text{how are things different in the real world?}
\text{- real optimizers consider bushy plans (why?)} \\
\begin{array}{cccc}
A & \text{D} & \text{B} & \text{C} & \text{E} \\
\end{array} \\
\text{- selectivity estimation is much more complicated than selinger says and is very important.}

\text{how does selinger estimate the size of a join?}
\text{- selinger just uses rough heuristics for equality and range predicates.}
what can go wrong?
consider ABCD
suppose sel (A join B) = .1
everything else is .01
If I don’t leave A join B until last, I’m off by a factor of 10

- how can we do a better job?
(multi-d) histograms, sampling, etc.

example: 1d hist

example: 2d hist