

# Selinger Optimizer

6.830 Lecture 10  
October 15, 2009

Sam Madden

# The Problem

- How to order a series of N joins, e.g.,

A.a = B.b AND A.c = D.d AND B.e = C.f

N! ways to order joins (e.g., ABCD, ACBD, ....)

(N-1)! plans per ordering (e.g., (((AB)C)D), ((AB)(CD)), ...)

Multiple implementations (e.g., hash, nested loops, etc)

- Naïve approach doesn't scale, e.g., for 20-way join
  - $10! \times 9! = 1.3 \times 10^{12}$
  - $20! \times 19! = 2.9 \times 10^{35}$

# Selinger Optimizations

- Left-deep only  $((AB)C)D$  (eliminate  $(N-1)!$ )
- Push-down selections
- Don't consider cross products
- Dynamic programming algorithm

# Dynamic Programming

$R \leftarrow$  set of relations to join (e.g., ABCD)

For  $d$  in  $\{1 \dots |R|\}$ :

for  $S$  in {all length  $d$  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  $a$

**optjoin**( $S-a$ ) is cached from previous iteration

# Example

optjoin(ABCD) – assume all joins are NL

$\partial=1$

A = best way to access A

(e.g., sequential scan, or predicate pushdown into index...)

B = best way to access B

C = best way to access C

D = best way to access D

Cache		
Subplan	Best choice	Cost
A	index	100
B	seq scan	50
...		

Total cost computations: *choose*(N,1), where N is number of relations

# Example

optjoin(ABCD)

$\partial=2$

$\{A,B\} = AB \text{ or } BA$

(using previously computed best way to access A and B)

$\{B,C\} = BC \text{ or } CB$

$\{C,D\} = CD \text{ or } DC$

$\{A,C\} = AC \text{ or } CA$

$\{A,D\} = AD \text{ or } DA$

$\{B,D\} = BD \text{ or } DB$

Total cost computations:  $choose(N,2) \times 2$

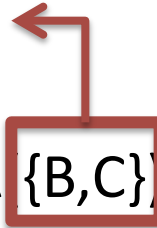
Cache		
Subplan	Best choice	Cost
A	index	100
B	seq scan	50
{A,B}	BA	156
{B,C}	BC	98
...		

# Example

optjoin(ABCD)

$\partial=3$

Already computed –  
lookup in cache



$\{A,B,C\}$  = remove A, compare A  $\{B,C\}$  to  $(\{B,C\})A$   
 remove B, compare B  $(\{A,C\})$  to  $(\{A,C\})B$   
 remove C, compare C  $(\{A,B\})$  to  $(\{A,B\})C$   
 $\{A,B,D\}$  = remove A, compare A  $(\{B,D\})$  to  $(\{B,D\})A$

....

$\{A,C,D\}$  = ...

$\{B,C,D\}$  = ...

Cache

Subplan	Best choice	Cost
A	index	100
B	seq scan	50
$\{A,B\}$	BA	156
$\{B,C\}$	BC	98
$\{A,B,C\}$	BCA	125
...		
$\{B,C,D\}$	BCD	115

Total cost computations:  $choose(N,3) \times 3 \times 2$

# Example

optjoin(ABCD)

$\partial=4$

Already computed –  
lookup in cache



$\{A,B,C,D\}$  = remove A, compare A  $\{B,C,D\}$  to  $(\{B,C,D\})A$   
 remove B, compare B  $\{A,C,D\}$  to  $(\{A,C,D\})B$   
 remove C, compare C  $\{A,B,D\}$  to  $(\{A,B,D\})C$   
 remove D, compare D  $\{A,B,C\}$  to  $(\{A,B,C\})D$

Cache

Subplan	Best choice	Cost
A	index	100
B	seq scan	50
$\{A,B\}$	BA	156
$\{B,C\}$	BC	98
$\{A,B,C\}$	BCA	125
$\{B,C,D\}$	BCD	115
$\{A,B,C,D\}$	ABCD	215

**Final answer is plan with minimum cost of these four**

Total cost computations:  $choose(N,4) \times 4 \times 2$



# Complexity

$choose(n,1) + choose(n,2) + \dots + choose(n,n)$  total subsets considered

All subsets of a size  $n$  set = power set of  $n = 2^n$

Equiv. to computing all binary strings of size  $n$

000,001,010,100,011,101,110,111

Each bit represents whether an item is in or out of set

# Complexity (continued)

For each subset,

k ways to remove 1 join

$k < n$

m ways to join 1 relation with remainder

Total cost:  $O(nm2^n)$  plan evaluations

$n = 20, m = 2$

$4.1 \times 10^7$

# Interesting Orders

- Some queries need data in sorted order
  - Some plans produce sorted data (e.g., using an index scan or merge join)
- May be non-optimal way to join data, but overall optimal plan
  - Avoids final sort
- In cache, maintain best overall plan, plus best plan for each interesting order
- At end, compare cost of
  - best plan + sort into order
  - to
  - best in order plan
- Increases complexity by factor of  $k+1$ , where  $k$  is number of interesting orders

# Example

SELECT A.f3, B.f2 FROM A,B where A.f3 = B.f4  
ORDER BY A.f3

Subplan	Best choice	Cost	Best in A.f3 order	Cost
A	index	100	index	100
B	seq scan	50	seqscan	50
{A,B}	BA hash	156	AB merge	180

compare:

cost(sort(output)) + 156

to

180