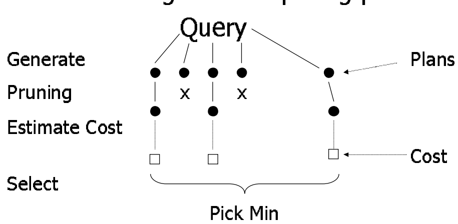
Chapters 15 and 16b:
Query Optimization

(Slides by Hector Garcia-Molina,
http://www-db.stanford.edu/~hector/cs245/notes.htm)

Query Optimization

--> Generating and comparing plans



To generate plans consider:

- Transforming relational algebra expression
  (e.g. order of joins)
- Use of existing indexes
- Building indexes or sorting on the fly

- Implementation details:
  e.g. - Join algorithm
  - Memory management
  - Parallel processing

Estimating IOs:

- Count # of disk blocks that must be read (or written) to execute query plan

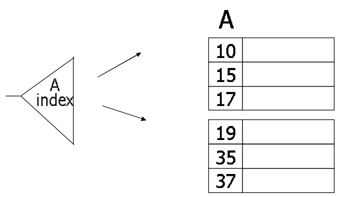
To estimate costs, we may have additional parameters:

B(R) = # of blocks containing R tuples
f(R) = max # of tuples of R per block
M = # memory blocks available

HT(i) = # levels in index i
LB(i) = # of leaf blocks in index i

Clustering index

Index that allows tuples to be read in an order that corresponds to physical order



Notions of clustering

- Clustered file organization

  | R1 R2 S1 S2 | | R3 R4 S3 S4 | ..... |

- Clustered relation

  | R1 R2 R3 R4 | | R5 R5 R7 R8 | ..... |

- Clustering index

Example R1 $\bowtie$ R2 over common attribute C

T(R1) = 10,000
T(R2) = 5,000
S(R1) = S(R2) = 1/10 block
Memory available = 101 blocks

$\rightarrow$ Metric: # of IOs
(ignoring writing of result)

Caution!

This may not be the best way to compare

- ignoring CPU costs
- ignoring timing
- ignoring double buffering requirements

Options

- Transformations: R1 $\bowtie$ R2, R2 $\bowtie$ R1
- Joint algorithms:
  - Iteration (nested loops)
  - Merge join
  - Join with index
  - Hash join

- Iteration join (conceptually)

  for each r $\in$ R1 do
    for each s $\in$ R2 do
      if r.C = s.C then output r,s pair

- Merge join (conceptually)

(1) if R1 and R2 not sorted, sort them

(2) $i \leftarrow 1$; $j \leftarrow 1$;

While $(i \leq T(R1)) \wedge (j \leq T(R2))$ do

if R1{ i }.C = R2{ j }.C then outputTuples

else if R1{ i }.C > R2{ j }.C then $j \leftarrow j+1$

else if R1{ i }.C < R2{ j }.C then $i \leftarrow i+1$

Procedure Output-Tuples

While (R1{ i }.C = R2{ j }.C) $\wedge$ $(i \leq T(R1))$ do

[$jj \leftarrow j$;

while (R1{ i }.C = R2{ jj }.C) $\wedge$ $(jj \leq T(R2))$ do

[output pair R1{ i }, R2{ jj };

$jj \leftarrow jj+1$]

$i \leftarrow i+1$]

Example

i	R1{i}.C	R2{j}.C	j
1	10	5	1
2	20	20	2
3	20	20	3
4	30	30	4
5	40	30	5
		50	6
		52	7

- Join with index (Conceptually)

Assume R2.C index

For each r ∈ R1 do
  [X ← index (R2, C, r.C)
    for each s ∈ X do
      output r,s pair]

Note: X ← index(rel, attr, value)
  then X = set of rel tuples with attr = value

- Hash join (conceptual)
  - Hash function h, range 0 → k
  - Buckets for R1: G0, G1, ... Gk
  - Buckets for R2: H0, H1, ... Hk

Algorithm

(1) Hash R1 tuples into G buckets
(2) Hash R2 tuples into H buckets
(3) For i = 0 to k do
  match tuples in Gi, Hi buckets

Simple example   hash: even/odd

R1	R2
2	5
4	4
3	12
5	3
8	13
9	8
	11
	14

Buckets

	R1	R2
Even	2 4 8	4 12 8 14
Odd:	3 5 9	5 3 13 11

Factors that affect performance

(1) Tuples of relation stored
physically together?

(2) Relations sorted by join attribute?

(3) Indexes exist?

Example 1(a) Iteration Join R1 $\bowtie$ R2

- Relations not contiguous
- Recall $\begin{cases} T(R1) = 10{,}000 \quad T(R2) = 5{,}000 \\ S(R1) = S(R2) = 1/10 \text{ block} \\ MEM=101 \text{ blocks} \end{cases}$

Cost: for each R1 tuple:
[Read tuple + Read R2]
Total =10,000 [1+5000]=50,010,000 IOs

- Can we do better?

Use our memory

(1) Read 100 blocks of R1
(2) Read all of R2 (using 1 block) + join
(3) Repeat until done

Cost: for each R1 chunk:

Read chunk: 1000 IOs

Read R2 5000 IOs

6000

$$\text{Total} = \frac{10{,}000}{1{,}000} \times 6000 = 60{,}000 \text{ IOs}$$

Chapters 15-16b 22

- Can we do better?

☛ Reverse join order: R2 ⋈ R1

$$\text{Total} = \frac{5000}{1000} \times (1000 + 10{,}000) =$$

5 x 11,000 = 55,000 IOs

Chapters 15-16b 23

Example 1(b) Iteration Join R2 ⋈ R1

- Relations contiguous

Cost

For each R2 chunk:

Read chunk: 100 IOs

Read R1: 1000 IOs

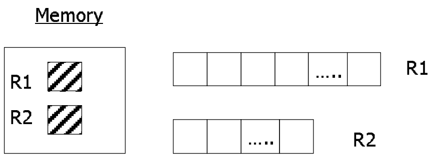
1,100

Total= 5 chunks x 1,100 = 5,500 IOs

Chapters 15-16b 24

Example 1(c) Merge Join

- Both R1, R2 ordered by C; relations contiguous



Total cost: Read R1 cost + read R2 cost
= 1000 + 500 = 1,500 IOs

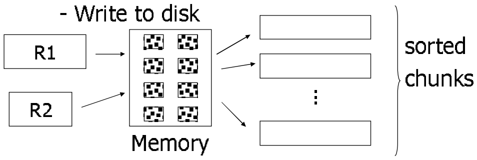
Example 1(d) Merge Join

- R1, R2 not ordered, but contiguous

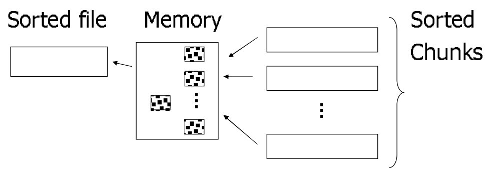
--> Need to sort R1, R2 first…. HOW?

One way to sort: Merge Sort

(i) For each 100 blk chunk of R:
- Read chunk
- Sort in memory
- Write to disk



(ii) Read all chunks + merge + write out



Cost: Sort

Each tuple is read,written,
read, written

so...

Sort cost R1: 4 x 1,000 = 4,000

Sort cost R2: 4 x 500 = 2,000

Example 1(d) Merge Join (continued)

R1,R2 contiguous, but unordered

Total cost = sort cost + join cost
= 6,000 + 1,500 = 7,500 IOs

But: Iteration cost = 5,500
so merge joint does not pay off!

But say R1 = 10,000 blocks contiguous
R2 = 5,000 blocks not ordered

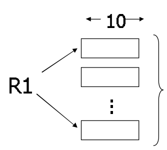
<u>Iterate:</u> $\frac{5000}{100} \times (100+10{,}000) = 50 \times 10{,}100$
$= 505{,}000$ IOs

<u>Merge join:</u> $5(10{,}000+5{,}000) = 75{,}000$ IOs

Merge Join (with sort) WINS!

How much memory do we need for merge sort?

E.g: Say I have 10 memory blocks



100 chunks ⇒ to merge, need 100 blocks!

<u>In general:</u>

Say k blocks in memory
x blocks for relation sort

chunks = (x/k) size of chunk = k

chunks $\leq$ buffers available for merge

so... $(x/k) \leq k$
or $k^2 \geq x$ or $k \geq \sqrt{x}$

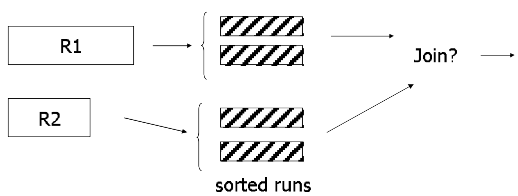
In our example

R1 is 1000 blocks, $k \geq 31.62$
R2 is 500 blocks, $k \geq 22.36$

Need at least 32 buffers

Can we improve on merge join?

Hint: do we really need the fully sorted files?



Cost of improved merge join:

C = Read R1 + write R1 into runs
+ read R2 + write R2 into runs
+ join
= 2000 + 1000 + 1500 = 4500

--> Memory requirement?

Example 1(e) Index Join

- Assume R1.C index exists; 2 levels
- Assume R2 contiguous, unordered
- Assume R1.C index fits in memory

Cost: Reads: 500 IOs
for each R2 tuple:
- probe index - free
- if match, read R1 tuple: 1 IO

What is expected # of matching tuples?

(a) say R1.C is key, R2.C is foreign key
then expect = 1

(b) say V(R1,C) = 5000, T(R1) = 10,000
with uniform assumption
expect = 10,000/5,000 = 2

What is expected # of matching tuples?

(c) Say DOM(R1, C)=1,000,000
T(R1) = 10,000
with alternate assumption

$$\text{Expect} = \frac{10{,}000}{1{,}000{,}000} = \frac{1}{100}$$

Total cost with index join

(a) Total cost = 500+5000(1)1 = 5,500

(b) Total cost = 500+5000(2)1 = 10,500

(c) Total cost = 500+5000(1/100)1=550

What if index does not fit in memory?

Example: say R1.C index is 201 blocks

- Keep root + 99 leaf nodes in memory
- Expected cost of each probe is

$$E = (0)\frac{99}{200} + (1)\frac{101}{200} \approx 0.5$$

Total cost (including probes)

= 500+5000 [Probe + get records]
= 500+5000 [0.5+2]   uniform assumption
= 500+12,500 = 13,000   (case b)

For case (c):
= 500+5000[0.5 $\times$ 1 + (1/100) $\times$ 1]
= 500+2500+50 = 3050 IOs

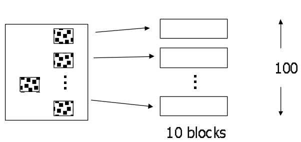
So far

not contiguous	Iterate R2 $\bowtie$ R1	55,000 (best)
	Merge Join	_______
	Sort+ Merge Join	_______
	R1.C Index	_______
	R2.C Index	_______
contiguous	Iterate R2 $\bowtie$ R1	5500
	Merge join	1500
	Sort+Merge Join	7500 $\rightarrow$ 4500
	R1.C Index	5500 $\rightarrow$ 3050 $\rightarrow$ 550
	R2.C Index	________

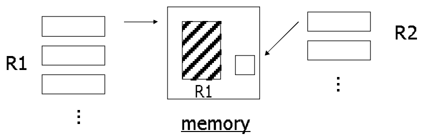
Example 1(f)   Hash Join

- R1, R2 contiguous (un-ordered)

$\rightarrow$ Use 100 buckets
$\rightarrow$ Read R1, hash, + write buckets

R1 $\rightarrow$


-> Same for R2
-> Read one R1 bucket; build memory hash table
-> Read corresponding R2 bucket + hash probe



⇒ Then repeat for all buckets

Cost:

"Bucketize:" Read R1 + write
Read R2 + write

Join: Read R1, R2

Total cost = 3 x [1000+500] = 4500

Note: this is an approximation since buckets will vary in size and we have to round up to blocks

Minimum memory requirements:

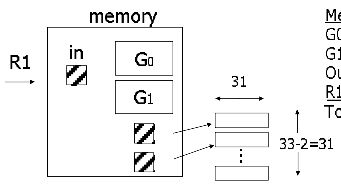
Size of R1 bucket = (x/k)
k = number of memory buffers
x = number of R1 blocks

So... $(x/k) < k$

$k > \sqrt{x}$ need: k+1 total memory buffers

Trick: keep some buckets in memory

E.g., k'=33 R1 buckets = 31 blocks
keep 2 in memory



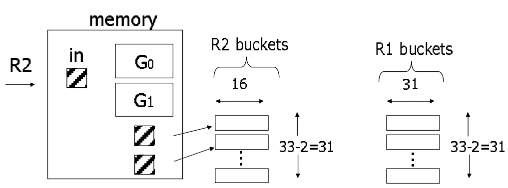
Memory use:

G0	31 buffers
G1	31 buffers
Output	33-2 buffers
R1 input	1
Total	94 buffers
	6 buffers to spare!!

called hybrid hash-join

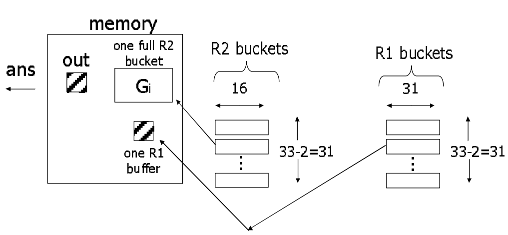
Next: Bucketize R2

- R2 buckets =500/33= 16 blocks
- Two of the R2 buckets joined immediately with G0,G1



Finally: Join remaining buckets

- for each bucket pair:
  - read one of the buckets into memory
  - join with second bucket

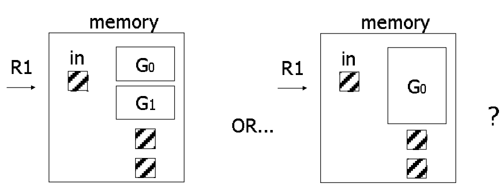


Cost

- Bucketize R1 = 1000+31×31=1961
- To bucketize R2, only write 31 buckets:
  so, cost = 500+31×16=996
- To compare join (2 buckets already done)
  read 31×31+31×16=1457

Total cost = 1961+996+1457 = 4414

- How many buckets in memory?



☞ See textbook for answer...

Another hash join trick:

- Only write into buckets
  <val,ptr> pairs
- When we get a match in join phase,
  must fetch tuples

- To illustrate cost computation, assume:
  - 100 <val,ptr> pairs/block
  - expected number of result tuples is 100
- Build hash table for R2 in memory
  5000 tuples → 5000/100 = 50 blocks
- Read R1 and match
- Read ~ 100 R2 tuples

Total cost =	Read R2:	500
	Read R1:	1000
	Get tuples:	100
		1600

So far:

contiguous

Iterate	5500
Merge join	1500
Sort+merge joint	7500
R1.C index	5500 → 550
R2.C index	_____
Build R.C index	_____
Build S.C index	_____
Hash join	4500+
with trick,R1 first	4414
with trick,R2 first	_____
Hash join, pointers	1600

Summary

- Iteration ok for "small" relations
  (relative to memory size)
- For equi-join, where relations not
  sorted and no indexes exist,
  hash join usually best

- Sort + merge join good for non-equi-join (e.g., R1.C > R2.C)
- If relations already sorted, use merge join
- If index exists, it could be useful

  (depends on expected result size)

Join strategies for parallel processors

Later on….

Chapter 7 summary

- Relational algebra level
- Detailed query plan level
  - Estimate costs
  - Generate plans
    - Join algorithms
  - Compare costs