

Class 19: Joins II

Last time

Relation S, N=500 pages Ps=80
 Relation R, M=1000 pages PR=100

Nested-Loop Joins

Simple $(P_R \cdot M) \cdot N + M$ w/ R outer
 Page-oriented $M \cdot N + M$
 Block-based $\frac{M \cdot N}{K} + M$ w/ K buffer
 Index $M + M \cdot P_R \cdot (\text{index_access_cost} + \text{data_access_cost})$

hash \swarrow \searrow B⁺-tree \searrow clustered
 ~ 1.2 2-4 1 I/O per page of matching tuples

1 I/O per matching tuple

Sort-Merge Joins

$3 \cdot (M+N)$ if $B > \sqrt{M}$ where M is # pages of the larger relation

$M+N$ if $B > N$ where N corresponds to the smallest relation

Today

- Hash Joins
- General Join Conditions
- Aggregates

Hash Joins

- Use a hash function h to create partitions of both relations hashing (building)
- match tuples only between the corresponding partitions probing (matching)

B buffers

$R \bowtie S$
 $i=j$

h hash function

building

$\forall r \in R$

read r and add it to buffer $h(r_i)$

$\forall s \in S$

read s and add it to buffer $h(s_j)$

searching

for $l=1, 2, \dots, K$

$\forall r \in R_l$

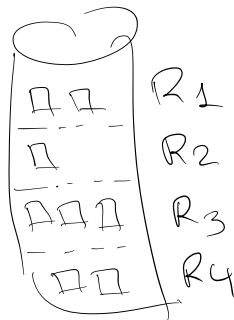
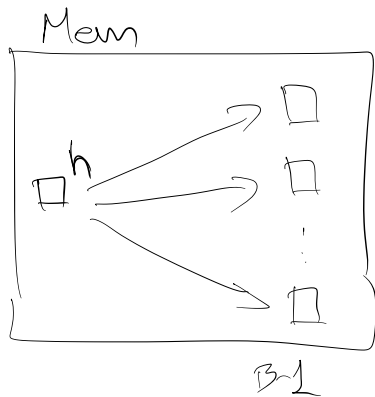
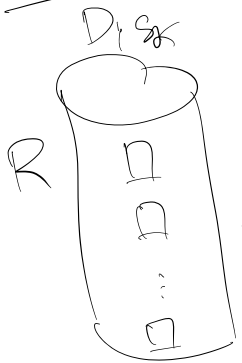
read r and insert into in-memory AT using $h_2(r_i)$

$\forall s \in S_l$

read s and probe AT using $h_2(s_j)$
if match found add (r, s) to the results

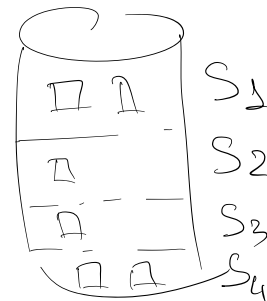
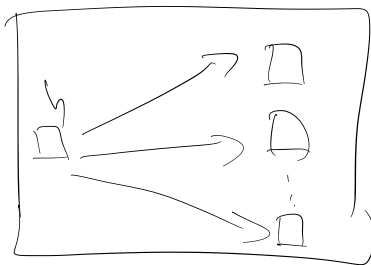
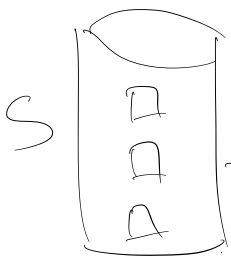
clear hash table from memory to proceed with next pair of partitions

Build



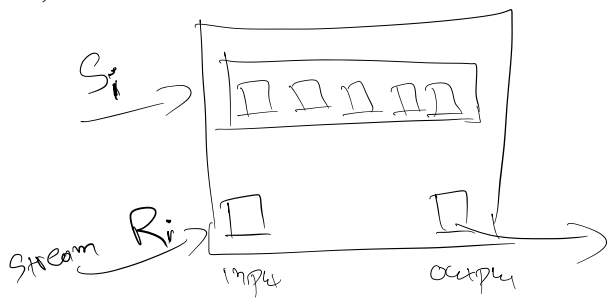
Cost

$2 \cdot M$



$2 \cdot N$

Matching



read every partition once
in-memory HT w/ $h_2 (\neq h)$

Search in S_i as we stream R_i

Cost: $M + N$

total cost of Hash Join = $3(M + N) = \boxed{4500} \rightarrow \boxed{93}$

Memory Requirements

→ enough buffer for the largest partition of the smaller relation (S)

→ Input page for the other relation

→ Output page

→ a few pages of hash metadata

fudge factor f (for example $f = 1.04$)

if $h \rightarrow$ uniform

size of a partition $\sim \frac{N}{B-1}$

$$B > \frac{f \cdot N}{B-1} + 2 \quad \approx \Rightarrow \quad B > \sqrt{f \cdot N}$$

What if not enough memory? (for S_i to fit in memory)

→ apply the same algorithm recursively

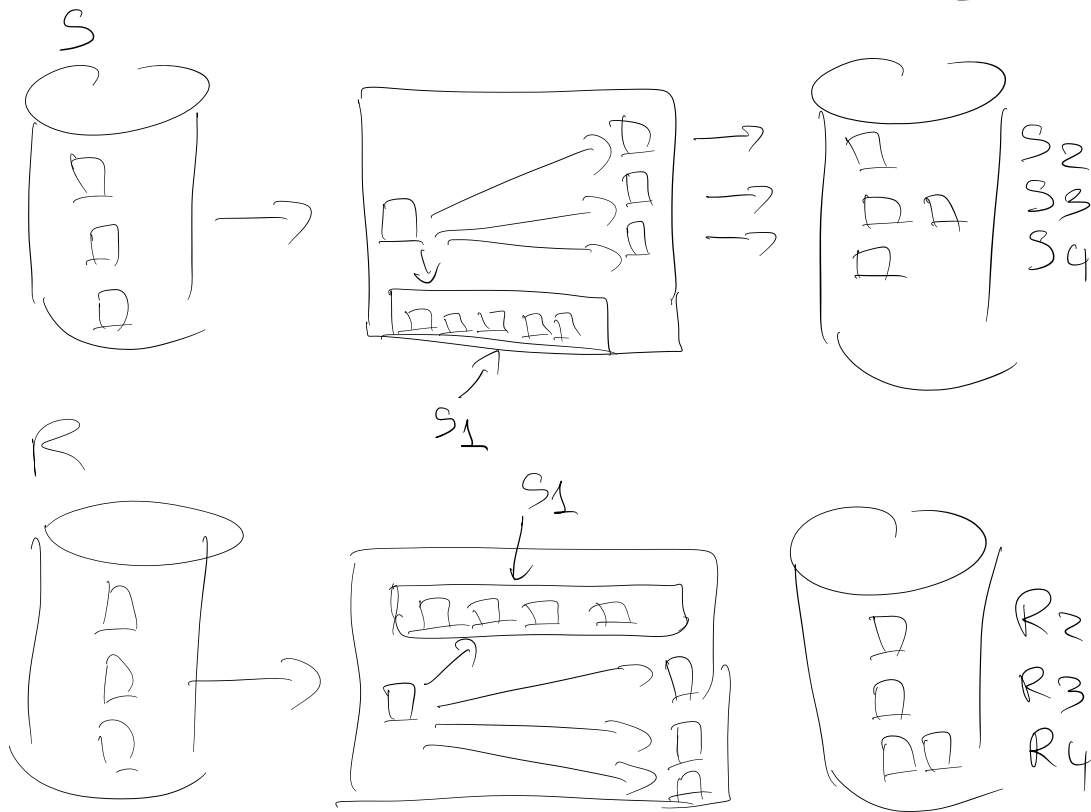
→ read, repartition S_i, R_i with $h_3 (\neq h_2, \neq h)$

→ matching per subpartition (mem. is enough)

* if not, again recursion

What if we have more memory?

Hybrid
Hash
Join



Cost

→ hashing S $N + N - \text{sizeof}(S_1)$
 → hashing R $M + M - \text{sizeof}(R_1)$
 → matching $M - \text{sizeof}(R_1) + N + \text{sizeof}(S_1)$

total $3(M+N) - 2(\text{sizeof}(S_1) + \text{sizeof}(R_1))$

$B = 300$

$M = 1000$

$N = 500$

$3(1000 + 500) - 2(500 + 250) = 4500 - 1500 = \boxed{3000}$

$\boxed{63}$

if $B = 600$

read S once + build hash table

scan R once prob S on-the-fly

Hash Join vs

SMJ

cost	$3(M+N)$	$3(M+N)$
mem. req	$B > \sqrt{RN} \leftarrow \text{smaller}$	$B > \sqrt{M} \leftarrow \text{larger}$
	$B > \sqrt{1.04 \cdot 500} = 23$	$B > \sqrt{1000} = 32$
$B(\geq m.r.) \leq N$	$3(M+N) - 2(\text{sizeof}(R_1) + \text{sizeof}(S_1))$	$3(M+N)$
$B > N$	$M+N$	$M+N$
output		sorted
if input sorted	$3(M+N)$	$M+N$

BUT sensitive to data skew

(a) equality joins on several attributes

(b) inequality joins

→ (a) for INLJ we need index with all attributes in join conditions

→ sort/hash use combination of all attributes

→ (b) INLJ w/ BT-Tree (not Hash Index)

HS/SMJ cannot work

Block NLJ the best approach

Set

UNION / EXCEPT (set difference)

→ Sorting

→ sort S+R on all attributes

→ merging → discard duplicates (UNION)

→ set-difference

→ hashing

→ partition $R+S$

→ $\#$ S-part probe corr. R-part

→ discard duplicates (UNION)

→ set-difference

→ Intersection → special case of Join
Equality across all attributes

Aggregation

→ SELECT AVG(sal) FROM E

→ SCAN once

→ GROUP BY

(age, avg-salary)

hash (age) → (age, salary, count)

sort (age) calculate "running info" of aggregation
on-the-fly

→ if we have an index on (Group-by, select, where)
can use only the index WAY FASTER

→ Buffering

Among things in parallel

tough to estimate what is needed by BP

SNLS BDN ✓.

BEN

LRU → sequential flooding.

MRU ✓.

INLS → sort the outer relation