Last time

Nested-Loop Joins

Simple \((P_R \cdot M) \cdot N + M\) w/ R outer

Page-oriented \(M \cdot N + M\)

Block-based \(M \cdot N + M\)

Index \(M + M \cdot P_R \cdot (\text{index-access-cost} + \text{data-access-cost})\)

Hash Join

Sort-Merge Joins

\(3 \cdot (M+N)\) if \(B \geq M\)

\(M+N\) if \(B < N\)

Relation S, \(N=500\) pages \(P_S=80\)

R, \(M=1000\) pages \(P_R=100\)

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)]
Buffers $B$ and hash function $h$

For $r \in R$
- Read $r$ and add it to buffer $h(r_i)$

For $s \in S$
- Read $s$ and add it to buffer $h(s_j)$

For $k = 1, 2, \ldots, K$
- For $r \in R_k$
  - Read $r$ and insert into in-memory HT using $h_2(r_i)$
  - For $s \in S_k$
    - Read $s$ and probe HT using $h_2(s_j)$
      - If match found, add $(r_i, s_j)$ to the result list.

Clear hash table from memory to proceed with next pair of partitions.

Cost is $2M + 2N$.?
Matching

\[ C_i \rightarrow [\text{memory} : S_i \rightarrow \text{output}] \]

Stream \( R_i \) into memory, read every partition once in-memory HT w/ h2 (th)

Search in \( S_i \) as we stream \( R_i \),

Cost: \( M + N \)

The total cost of Hash Join is \( 3(M + N) = 4500 \rightarrow 93 \)

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 \( h \approx \text{uniform} \)

Size of a partition \( \sim \frac{N}{B - 1} \)

\[ B > \frac{fN}{B - 1} + 2 \implies B > \sqrt{fN} \]

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

- Apply the same algorithm recursively
- Read, repartition \( S_i, R_i \) with \( h_3 \) (\( \neq h_2, \neq h \))
- Marching per subpartition (mem. is enough)
- If not, again recursion
What if we have more memory?

Hybrid Hash Join

Cost:

- Hashing S: \( N + N \cdot \text{sizeof}(S_1) \)
- Hashing R: \( M + M \cdot \text{sizeof}(R_1) \)
- Marching: \( M \cdot \text{sizeof}(R_1) + N \cdot \text{sizeof}(S_1) \)

Total: \( 3(M+N) - 2(\text{sizeof}(S_1) + \text{sizeof}(R_1)) \)

- \( B = 300 \)
- \( M = 1000 \)
- \( N = 500 \)

If \( B \leq 600 \):

- Read S once + build hash table
- Scan R once + prob S on-the-fly
**Hash Join vs SMJ**

<table>
<thead>
<tr>
<th>Cost</th>
<th>(3(M+N))</th>
<th>(3(M+N))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mem. req</td>
<td>(BD &lt; N) (\leq N) (\geq N)</td>
<td>(BD &lt; \sqrt{209.6} = 9.5) (BD &lt; \sqrt{100} = 10)</td>
</tr>
</tbody>
</table>

- **B (m.r.) \(\leq N\)**
  - \(3(M+N) - 2(\text{size} (R_1) - \text{size} (S_1))\)
  - \(M + N\)

- **B \(> N\)**
  - \(3(M+N)\)
  - \(M + N\)

**Output**
- If input sorted: \(3(M+N)\)
- \(M + N\)

**BUT** sensitive to data skew

(a) equality joins on several attributes

(b) inequality joins

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

\(\rightarrow\) sort/has to use combination of all attributes

\(\rightarrow (b)\) INLJ w/ B+Tree (not Hash Index)

HS/SMJ cannot work

Blob vs NLJ the best approach

**Set**

- **Union / Except (set difference)**
  - Sorting
    - sort \(S\) + \(R\) on all attributes
  - merging
    - discard duplicates (\text{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
  Alot of things in parallel
tough to estimate what is needed by BP

SNLS: B2N √
BN: LRU ← sequential flooding,
MRU V.

INLS: sort the outer relation