CS 245: Database System Principles

Notes 7: Query Optimization

Hector Garcia-Molina

To generate plans consider:

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

Estimating IOs:

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

To estimate costs, we may have additional parameters:

\[ B(R) = \text{# of blocks containing } R \text{ tuples} \]
\[ f(R) = \text{max # of tuples of } R \text{ per block} \]
\[ M = \text{# memory blocks available} \]
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
- Clustered relation
- Clustering index

**Example**

\( R_1 \bowtie R_2 \) over common attribute \( C \)

\[
\begin{align*}
T(R_1) &= 10,000 \\
T(R_2) &= 5,000 \\
S(R_1) &= S(R_2) = 1/10 \text{ block} \\
\text{Memory available} &= 101 \text{ blocks}
\end{align*}
\]

**Caution!**

This may not be the best way to compare

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

• Transformations: $R_1 \bowtie R_2$, $R_2 \bowtie R_1$

• Join algorithms:
  – Iteration (nested loops)
  – Merge join
  – Join with index
  – Hash join

• Iteration join (conceptually)
  for each $r \in R_1$ do
    for each $s \in R_2$ do
      if $r.C = s.C$ then output $r,s$ pair

• Merge join (conceptually)
  (1) if $R_1$ and $R_2$ not sorted, sort them
  (2) $i \leftarrow 1$; $j \leftarrow 1$
      While ($i \leq T(R_1)$) $\land$ ($j \leq T(R_2)$) do
        if $R_1{i}.C = R_2{j}.C$ then output tuples
        else if $R_1{i}.C > R_2{j}.C$ then $j \leftarrow j+1$
        else if $R_1{i}.C < R_2{j}.C$ then $i \leftarrow i+1$

• Join with index (Conceptually)
  For each $r \in R_1$ do
    $X \leftarrow$ index($R_2$, $C$, $r.C$)
    for each $s \in X$ do
      output $r,s$ pair

  Assume $R_2.C$ index

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

Example

<table>
<thead>
<tr>
<th>$i$</th>
<th>$R_1{i}.C$</th>
<th>$R_2{j}.C$</th>
<th>$j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Procedure Output-Tuples

While ($R_1{i}.C = R_2{j}.C \land i \leq T(R_1)$) do
  $jj \leftarrow j$
  while ($R_1{i}.C = R_2{jj}.C \land jj \leq T(R_2)$) do
    output pair $R_1{i}$, $R_2{jj}$
    $jj \leftarrow jj+1$
  $i \leftarrow i+1$
• Hash join (conceptual)
  - Hash function \( h \), range 0 \( \rightarrow k \)
  - Buckets for \( R_1 \): \( G_0, G_1, \ldots, G_k \)
  - Buckets for \( R_2 \): \( H_0, H_1, \ldots, H_k \)

**Algorithm**
1. Hash \( R_1 \) tuples into \( G \) buckets
2. Hash \( R_2 \) tuples into \( H \) buckets
3. For \( i = 0 \) to \( k \) do
   match tuples in \( G_i, H_i \) buckets

---

**Simple example** hash: even/odd

<table>
<thead>
<tr>
<th>( R_1 )</th>
<th>( R_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
</tr>
</tbody>
</table>

**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 \( R_1 \bowtie R_2 \)**

- Relations **not** contiguous
- Recall
  \[ T(R_1) = 10,000 \quad T(R_2) = 5,000 \]
  \[ S(R_1) = S(R_2) = 1/10 \text{ block} \]
  \[ \text{MEM}=101 \text{ blocks} \]

**Cost:** for each \( R_1 \) tuple:
\[ \text{Read tuple + Read } R_2 \]
Total = 10,000 \( \times [1+5000] \) = 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
\[ \frac{6000}{6000} \]

Total = \( 10,000 \times \frac{6000}{1000} = 60,000 \) IOs

• Can we do better?

Reverse join order: R2 \( \bowtie \) R1
Total = \( 5000 \times (1000 + 10,000) = \frac{10000}{1000} \)
\[ 5 \times 11,000 = 55,000 \) IOs
Example 1(b) Iteration Join  \( R_2 \bowtie R_1 \)
- Relations contiguous

**Cost**
For each \( R_2 \) chunk:
- Read chunk: 100 IOs
- Read \( R_1 \): 1000 IOs
  \( 1,100 \)

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

---

Example 1(c) Merge Join
- Both \( R_1, R_2 \) ordered by \( C \); relations contiguous

**Memory**

\[
\begin{array}{c}
\text{R1} \\
\text{R2}
\end{array}
\cdots
\begin{array}{c}
\text{.....} \\
\text{R1}
\end{array}
\]

\[
\begin{array}{c}
\text{R1} \\
\text{R2}
\end{array}
\cdots
\begin{array}{c}
\text{.....} \\
\text{R2}
\end{array}
\]

**Total cost:** Read \( R_1 \) cost + read \( R_2 \) cost
\( = 1000 + 500 = 1,500 \) IOs

---

Example 1(d) Merge Join
- \( R_1, R_2 \) not ordered, but contiguous

\( \rightarrow \) Need to sort \( R_1, R_2 \) first.... HOW?

---

One way to sort: Merge Sort

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

\[
\begin{array}{c}
\text{R1} \\
\text{R2}
\end{array}
\rightarrow
\begin{array}{c}
\text{sorted} \\
\text{chunks}
\end{array}
\]

\[
\begin{array}{c}
\text{Memory}
\end{array}
\]
(ii) Read all chunks + merge + write out

Sorted file Memory Sorted
Chunks

Cost: Sort
Each tuple is read, written, read, written
so...
Sort cost R1: $4 \times 1,000 = 4,000$
Sort cost R2: $4 \times 500 = 2,000$

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!

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

Iterate: \[5000 \times (100+10,000) = 50 \times 10,100\]
$= 505,000$ IOs

Merge join: \[5(10,000+5,000) = 75,000\] 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!
In general:
Say $k$ blocks in memory
$x$ blocks for relation sort
# chunks = $(x/k)$, size of chunk = $k$

In general:
Say $k$ blocks in memory
$x$ blocks for relation sort
# chunks = $(x/k)$, size of chunk = $k$
# chunks $\leq$ buffers available for merge

In our example
$R_1$ is 1000 blocks, $k \geq 31.62$
$R_2$ 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 $R_1$ + write $R_1$ into runs
+ read $R_2$ + write $R_2$ 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 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
  \( E = \left( 0 \right) \frac{99}{200} + \left( 1 \right) \frac{101}{200} = 0.5 \)

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
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 × 1 + (1/100) × 1]
- 500 + 2500 + 50 = 3050 IOs

So far

- Iterate R2 \(\times\) R1 55,000 (best)
- Merge Join __________
- Sort + Merge Join __________
- R1.C Index __________
- R2.C Index __________

For contiguous:

- Iterate R2 \(\times\) R1 5500
- Merge join 1500
- Sort + Merge Join 7500 → 4500
- R1.C Index 5500 → 3050 → 550
- R2.C Index __________

Example 1(f) Hash Join

- R1, R2 contiguous (un-ordered)
  - Use 100 buckets
  - Read R1, hash, + write buckets

Cost:

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

Total cost = 3 × (1000 + 500) = 4500

- Therefore 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

Trick: keep some buckets in memory
E.g., k’=33 R1 buckets = 31 blocks keep 2 in memory

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

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

• How many buckets in memory?

• To illustrate cost computation, assume:
  - 100 <val,ptr> pairs/block
  - expected number of result tuples is 100

• 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 =

  \[
  \begin{align*}
  \text{Read R2:} & \quad 500 \\
  \text{Read R1:} & \quad 1000 \\
  \text{Get tuples:} & \quad \frac{100}{1600}
  \end{align*}
  \]
So far:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterate</td>
<td>5500</td>
</tr>
<tr>
<td>Merge join</td>
<td>1500</td>
</tr>
<tr>
<td>Sort+merge joint</td>
<td>7500</td>
</tr>
<tr>
<td>R1.C index</td>
<td>5500 (\rightarrow) 550</td>
</tr>
<tr>
<td>R2.C index</td>
<td>-----</td>
</tr>
<tr>
<td>Build R.C index</td>
<td>-----</td>
</tr>
<tr>
<td>Build S.C index</td>
<td>-----</td>
</tr>
<tr>
<td>Hash join, with trick, R1 first</td>
<td>4414</td>
</tr>
<tr>
<td>Hash join, with trick, R2 first</td>
<td>1600</td>
</tr>
</tbody>
</table>

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

Join strategies for parallel processors

Later on....

Chapter 16 [16] summary

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