Introduction to Database Systems
CSE 414

Lecture 27: More Operator Costs
Announcements

• HW8 and WQ7 both due tonight!

• Please fill out course evals online!

• Last lecture on Friday
Final Exam

• Thursday 6/7, 2:30-4:20pm
• Location: here
• Can bring 2 letter-size sheets of notes
  – Handwritten or printed
• More info on course website

• Review session:
  – Sunday 6/3, 2:30-5pm, SMI 102
Big Picture

• How to choose the “best” query plan to run? (aka query optimization)
• To answer this question we need to understand:
  – Data organization on the disk
  – Index structures and how they are used in queries
  – A way to model query “costs”
  – Compute cost for each query operator
  – Compute cost for each physical plan

Last topics this quarter!
Big Picture

Why do we care about all these internal details?
Cost Parameters

• Cost = I/O + CPU + Network BW
  – We will focus on I/O in this class

• Parameters (a.k.a. statistics):
  – \( B(R) \) = # of blocks (i.e., pages) for relation R
  – \( T(R) \) = # of tuples in relation R
  – \( V(R, a) \) = # of distinct values of attribute a

When \( a \) is a key, \( V(R,a) = T(R) \)
When \( a \) is not a key, \( V(R,a) \) can be anything \( \leq T(R) \)

• DBMS collects statistics about base tables
  must infer them for intermediate results
Join Algorithms

• Nested loop join *(short review)*

• Hash join

• Sort-merge join
Nested Loop Joins (review)
Nested Loop Joins

- Tuple-based nested loop $R \bowtie S$
- $R$ is the outer relation, $S$ is the inner relation

```
for each tuple $t_1$ in $R$ do
  for each tuple $t_2$ in $S$ do
    if $t_1$ and $t_2$ join then output ($t_1$, $t_2$)
```

What is the Cost?
Nested Loop Joins

- Tuple-based nested loop \( R \bowtie S \)
- \( R \) is the outer relation, \( S \) is the inner relation

```plaintext
for each tuple \( t_1 \) in \( R \) do
  for each tuple \( t_2 \) in \( S \) do
    if \( t_1 \) and \( t_2 \) join then output \((t_1,t_2)\)
```

- Cost: \( B(R) + T(R) \cdot B(S) \)
- Multiple-pass since \( S \) is read many times
Page-at-a-time Refinement

for each page of tuples $r$ in $R$ do
    for each page of tuples $s$ in $S$ do
        for all pairs of tuples $t_1$ in $r$, $t_2$ in $s$
            if $t_1$ and $t_2$ join then output $(t_1,t_2)$

- Cost: $B(R) + B(R)B(S)$
Page-at-a-time Refinement

Do any pairs of these join?

Disk

Patient Insurance

Input buffer for Patient

Input buffer for Insurance

Output buffer
Page-at-a-time Refinement

Do any pairs of these join?

Disk

Patient | Insurance
--- | ---
1 2 | 2 4 6 6
3 4 | 4 3 1 3
9 6 | 2 8
8 5 | 8 9

Input buffer for Patient

1 2
4 3

Input buffer for Insurance

Output buffer
Page-at-a-time Refinement

<table>
<thead>
<tr>
<th>Disk</th>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td></td>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td></td>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td></td>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Do any pairs of these join?

**Input buffer for Patient**

1 2

2 8

**Input buffer for Insurance**

Keep going until read all of Insurance

Then repeat for next page of Patient… until end of Patient

2 2

**Output buffer**

Cost: $B(R) + B(R)B(S)$
Hash Join
Hash Join

Hash join: \( R \bowtie S \)
- Scan \( R \), build hash table in main memory
- Then scan \( S \) and join
- Cost: \( B(R) + B(S) \)
- Which relation to build the hash table on?

- One-pass algorithm when \( B(R) \leq M \)
  - \( M = \) number of memory pages available
# Hash Join Example

**Patient**\(\text{pid, name, address}\)  
**Insurance**\(\text{pid, provider, policy_nb}\)  
**Patient \(\bowtie\) Insurance**

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
<th>Two tuples per page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td>‘Bob’</td>
<td>‘Seattle’</td>
</tr>
<tr>
<td><strong>2</strong></td>
<td>‘Ela’</td>
<td>‘Everett’</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td>‘Jill’</td>
<td>‘Kent’</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td>‘Joe’</td>
<td>‘Seattle’</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>2</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘Blue’</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>123</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>4</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘Prem’</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>432</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>4</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘Prem’</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>343</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>3</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘GrpH’</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>554</strong></td>
</tr>
</tbody>
</table>
Hash Join Example

Patient \times Insurance

Showing pid only

Memory M = 21 pages

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Some large-enough #

This is one page with two tuples
Hash Join Example

Step 1: Scan Patient and **build** hash table in memory

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

Memory M = 21 pages

Hash h: pid % 5

<table>
<thead>
<tr>
<th>= 0</th>
<th>= 1</th>
<th>= 2</th>
<th>= 3</th>
<th>= 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

Disk

Input buffer
Hash Join Example

Step 2: Scan Insurance and probe into hash table

Memory M = 21 pages

Hash h: pid % 5

Disk

Patient

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

Insurance

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Input buffer

Output buffer

Write to disk or pass to next operator
Hash Join Example

Step 2: Scan Insurance and **probe** into hash table

Memory $M = 21$ pages

Hash $h: \text{pid} \% 5$

Disk

Patient  Insurance

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input buffer

Output buffer

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>= 0</th>
<th>= 1</th>
<th>= 2</th>
<th>= 3</th>
<th>= 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

21
Hash Join Example

Step 2: Scan Insurance and **probe** into hash table

Memory $M = 21$ pages

<table>
<thead>
<tr>
<th>Hash h: pid % 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 0</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

Cost: $B(R) + B(S)$
Sort-Merge Join

<table>
<thead>
<tr>
<th>R</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>43</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>24</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Sort $\rightarrow$ Sort $\rightarrow$ Merge
Sort-Merge Join

Sort-merge join: \( R \bowtie S \)

- Scan \( R \) and sort in main memory
- Scan \( S \) and sort in main memory
- Merge \( R \) and \( S \)

- Cost: \( B(R) + B(S) \)
- One pass algorithm when \( B(S) + B(R) \leq M \)
- Typically, this is NOT a one pass algorithm
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory

Memory M = 21 pages

Disk

Patient  Insurance

1 2  |  2 4  |  6 6
3 4  |  4 3  |  1 3
9 6  |  2 8  |
8 5  |  8 9  |
Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory

Memory M = 21 pages

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1 2 3 4 5 6 8 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 2 3 3 4 4 6</td>
</tr>
<tr>
<td>6 8 8 9</td>
</tr>
</tbody>
</table>
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Memory M = 21 pages

Disk

Patient | Insurance
---|---
1 2 | 2 4 6 6
3 4 | 4 3 1 3
9 6 | 2 8
8 5 | 8 9

Output buffer
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Memory M = 21 pages

Disk

Patient | Insurance
---|---
1 | 2
3 | 4
9 | 6
8 | 5

Output buffer

Keep going until end of first relation
Index Joins
Index Nested Loop Join

\( R \bowtie S \)

- Assume \( S \) has an index on the join attribute
- Iterate over \( R \), for each tuple fetch corresponding tuple(s) from \( S \)

- **Cost:**
  - If index on \( S \) is clustered:
    \[ B(R) + T(R) * (B(S) * 1/V(S,a)) \]
  - If index on \( S \) is unclustered:
    \[ B(R) + T(R) * (T(S) * 1/V(S,a)) \]
Index Nested Loop Join

If index on S is clustered:
\[ B(R) + T(R) \times (B(S) \times \frac{1}{V(S,a)}) \]

Still have to scan in R

Why is the multiplier term T(R)?

What does 1/V(S,a) represent?

T(R) must be used because we cannot assume that a whole block of R (B(R)) will have the same attribute to join on, and thus use the same index access on S for.

1/V(S,a) represents the nature of the B+ Tree index. We are only scanning as much as we need. Note that the performance of the index join will decrease as V decreases.
Index Nested Loop Join

If index on S is unclustered:

\[ B(R) + T(R) \times (T(S) \times \frac{1}{V(S,a)}) \]

Why did this change from \( B(R) \) to \( T(R) \)?

Remember that tuples are stored on contiguous blocks. In a clustered index from before we know we can scan a single chunk of the disk to get the entire desired range. In an unclustered index we no longer can assume contiguous access. Thus we estimate that every tuple needs its own I/O operation.
Generating Query Plans (review)
Review: Logical vs Physical Plans

• Logical plans:
  – Created by the parser from the input SQL text
  – Expressed as a relational algebra tree
  – Each SQL query has many possible logical plans

• Physical plans:
  – Goal is to choose an efficient implementation for each operator in the RA tree
  – Each logical plan has many possible physical plans
Relational algebra expression is also called the “logical query plan”
Review: Physical Query Plan 1

A physical query plan is a logical query plan annotated with physical implementation details.

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
Review: Physical Query Plan 2

Suppliers\((\text{sid, surname, scity, sstate})\)

Supplies\((\text{sid, pno, quantity})\)

(On the fly) \(\pi_{\text{surname}}\)

(On the fly) \(\sigma_{\text{scity='Seattle' and sstate='WA' and pno=2}}\)

(Hash join) \(\text{sid} = \text{sid}\)

Same logical query plan
Different physical plan

SELECT \(\text{sname}\)
FROM Suppliers \(x\), Supplies \(y\)
WHERE \(x.\text{sid} = y.\text{sid}\)
\(\text{and } y.\text{pno} = 2\)
\(\text{and } x.\text{scity} = \text{‘Seattle’}\)
\(\text{and } x.\text{sstate} = \text{‘WA’}\)
Supplier\((sid, sname, scity, sstate)\)
Supply\((sid, pno, quantity)\)

**Review: Physical Query Plan 3**

(On the fly) \(\Pi_{sname}\) (d)
(Sort-merge join) \(\sigma_{\text{scity='Seattle' and sstate='WA'}}\) (c)
(Scan & write to T1) \(\sigma_{pno=2}\) (Scan & write to T2)

Different but equivalent logical query plan; different physical

\[
\begin{align*}
\text{SELECT} & \quad \text{sname} \\
\text{FROM} & \quad \text{Supplier} \ x, \ \text{Supply} \ y \\
\text{WHERE} & \quad x.\text{sid} = y.\text{sid} \\
& \quad \text{and} \ y.\text{pno} = 2 \\
& \quad \text{and} \ x.\text{scity} = \text{'Seattle'} \\
& \quad \text{and} \ x.\text{sstate} = \text{'WA'}
\end{align*}
\]

Supplier \(\text{File scan}\)
Supply \(\text{File scan}\)
Query Optimization: Overview

• Compute cost of each operator
  – This depends on:
    • Table statistics (# of tuples etc)
    • Algorithm used

• Cost of a physical plan =
  \[ \text{sum(each operator cost)} \]

• Cost each plan and choose the one with lowest cost