Question in Class

Logical operator:
Product(pname, cname) ![ CONCAT ] Company(cname, city)

Propose three physical operators for the join, assuming the tables are in main memory:
1.
2.
3.

Cost Parameters

In database systems the data is on disks, not in main memory

The cost of an operation = total number of I/Os

Cost parameters:
- \( B(R) = \) number of blocks for relation \( R \)
- \( T(R) = \) number of tuples in relation \( R \)
- \( V(R, a) = \) number of distinct values of attribute \( a \)

Cost

Cost of an operation =
- number of disk I/Os needed to:
  - read the operands
  - compute the result

Cost of writing the result to disk is not included on the following slides

Question: the cost of sorting a table with \( B \) blocks?

Answer:
Nested Loop Joins

- Tuple-based nested loop R \times S
  
  \[
  \text{for each tuple } r \text{ in } R \text{ do} \\
  \text{for each tuple } s \text{ in } S \text{ do} \\
  \text{if } r \text{ and } s \text{ join then output } (r,s)
  \]

- Cost: T(R) B(S) when S is clustered
- Cost: T(R) T(S) when S is unclustered

Nested Loop Joins

- Block-based Nested Loop Join
  
  \[
  \text{for each } (M-2) \text{ blocks } bs \text{ of } S \text{ do} \\
  \text{for each block } br \text{ of } R \text{ do} \\
  \text{for each tuple } s \text{ in } bs \\
  \text{for each tuple } r \text{ in } br \text{ do} \\
  \text{if } r \text{ and } s \text{ join then output } (r,s)
  \]

- Cost: \text{Read S once: cost } B(S) \\
  - Outer loop runs } B(S)/(M-2) \text{ times, and each} \\
  - \text{time need to read } R: \text{costs } B(S)B(R)/(M-2) \\
  \text{- Total cost: } B(S) + B(S)B(R)/(M-2) \\
  \text{- Notice: it is better to iterate over the smaller} \\
  \text{relation first} \\
  \text{- R } \times \text{ S: R=outer relation, S=inner relation}

Merge-join

- Start by sorting both R and S on the join attribute: \\
  - Cost: 4B(R)+4B(S) (because need to write to disk) \\
  - Read both relations in sorted order, match tuples \\
  - Cost: B(R)+B(S) \\
  \text{- Difficulty: many tuples in R may match many in S} \\
  \text{- If at least one set of tuples fits in M, we are OK} \\
  \text{- Otherwise need nested loop, higher cost} \\
  \text{- Total cost: } 5B(R)+5B(S) \\
  \text{- Assumption: } B(R) \leq M' \text{, } B(S) \leq M'
Merge-join

Join R $\times$ S

- If the number of tuples in R matching those in S is small (or vice versa) we can compute the join during the merge phase
- Total cost: $3B(R)+3B(S)$
- Assumption: $B(R) + B(S) \leq M^2$

Partitioned Hash-based Algorithms

- Idea: partition a relation R into buckets, on disk
- Each bucket has size approx. $B(R)/M$

Hash Based Algorithms for $\delta$

- Recall: $\delta(R) = \text{duplicate elimination}$
- Step 1. Partition R into buckets
- Step 2. Apply $\delta$ to each bucket (may read in main memory)

- Cost: $3B(R)$
- Assumption: $B(R) \leq M^2$

Hash Based Algorithms for $\gamma$

- Recall: $\gamma(R) = \text{grouping and aggregation}$
- Step 1. Partition R into buckets
- Step 2. Apply $\gamma$ to each bucket (may read in main memory)

- Cost: $3B(R)$
- Assumption: $B(R) \leq M^2$

Partitioned Hash Join

R $\times$ S

- Step 1:
  - Hash S into $M$ buckets
  - send all buckets to disk
- Step 2
  - Hash R into $M$ buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

Hash-Join

- Partition both relations using hash fn $h$. R tuples in partition $i$ will only match S tuples in partition $i$.

- Read in a partition of R, hash it using $h_2$ ($\geq h$). Scan matching partition of S, search for matches.
Partitioned Hash Join

- Cost: $3B(R) + 3B(S)$
- Assumption: $\min(B(R), B(S)) \leq M^2$

Hybrid Hash Join Algorithm

- Partition $S$ into $k$ buckets
  - $t$ buckets $S_1, \ldots, S_t$ stay in memory
  - $k-t$ buckets $S_{t+1}, \ldots, S_k$ go to disk
- Partition $R$ into $k$ buckets
  - First $t$ buckets join immediately with $S$
  - Rest $k-t$ buckets go to disk
- Finally, join $k-t$ pairs of buckets:
  $(R_{t+1}, S_{t+1}), (R_{t+2}, S_{t+2}), \ldots, (R_k, S_k)$

Hybrid Join Algorithm

- How to choose $k$ and $t$?
  - Choose $k$ large but s.t. $k \leq M$
  - Choose $t/k$ large but s.t. $t/k \cdot B(S) \leq M$
  - Moreover: $t/k \cdot B(S) + k-t \leq M$
- Assuming $t/k \cdot B(S) \gg k-t$: $t/k = M/B(S)$

Indexed Based Algorithms

- Recall that in a clustered index all tuples with the same value of the key are clustered on as few blocks as possible

Index Based Selection

- Selection on equality: $\sigma_{a=v}(R)$
- Clustered index on $a$: cost$= B(R)/V(R,a)$
- Unclustered index on $a$: cost$= T(R)/V(R,a)$
Index Based Selection

- Example: \( B(R) = 2000, \quad T(R) = 100,000, \quad V(R,a) = 20 \), compute the cost of \( \sigma_{a=v}(R) \)
- Cost of table scan:
  - If \( R \) is clustered: \( B(R) = 2000 \) I/Os
  - If \( R \) is unclustered: \( T(R) = 100,000 \) I/Os
- Cost of index based selection:
  - If index is clustered: \( B(R)/V(R,a) = 100 \)
  - If index is unclustered: \( T(R)/V(R,a) = 5000 \)
- Notice: when \( V(R,a) \) is small, then unclustered index is useless

Index Based Join

- \( R \bowtie S \)
- Assume \( S \) has an index on the join attribute
- Iterate over \( R \), for each tuple fetch corresponding tuple(s) from \( S \)
- Assume \( R \) is clustered. Cost:
  - If index is clustered: \( B(R) + T(R)B(S)/V(S,a) \)
  - If index is unclustered: \( B(R) + T(R)T(S)/V(S,a) \)

Index Based Join

- Assume both \( R \) and \( S \) have a sorted index (B+ tree) on the join attribute
- Then perform a merge join (called zig-zag join)
- Cost: \( B(R) + B(S) \)

Example

Select \( Product.pname \)
From \( Product, Company \)
Where \( Product.maker = Company.cname \) and \( Company.city = "Seattle" \)

How do we execute this query?

Logical Plan:

```
Product(pname, maker), Company(cname, city)
Assume:
Clustered index: Product.pname, Company.cname
Unclustered index: Product.maker, Company.city
```

```
\( \sigma_{city="Seattle"} \)
Product.pname

\( \sigma_{maker=cname} \)
Company.cname
```

```
Physical plan 1:

Index-based selection

\[ \sigma_{\text{city} = \text{Seattle}} \]

Company (cname, city)  Product (pname, maker)

Index-based join

Physical plans 2a and 2b:

Which one is better ??

Index scan

Merging plan

\[ \sigma_{\text{city} = \text{Seattle}} \]

Product (pname, maker)  Company (cname, city)

Scan and sort (2a)  Index scan (2b)

Total cost:

1. Plan 1: \( T(\text{Company}) \times V(\text{Company}, \text{city}) \)
2. Plan 2a: \( B(\text{Company}) + 3B(\text{Product}) \)
3. Plan 2b: \( B(\text{Company}) + T(\text{Product}) \)

Example

\[ T(\text{Company}) = 5,000 \quad B(\text{Product}) = 500 \quad M = 100 \quad T(\text{Product}) = 100,000 \quad B(\text{Product}) = 1,000 \]

We may assume \( V(\text{Product, maker}) = T(\text{Company}) \) (why?)

1. Case 1: \( V(\text{Company, city}) = T(\text{Company}) \)
   \( V(\text{Company, city}) = 2,000 \)
2. Case 2: \( V(\text{Company, city}) \ll T(\text{Company}) \)
   \( V(\text{Company, city}) = 20 \)

Which one is better ??

It depends on the data !!
Which Plan is Best?

Plan 1: \( T(\text{Company}) \times V(\text{Company, city}) \times T(\text{Product}) \times V(\text{Product, maker}) \)
Plan 2a: \( B(\text{Company}) \times 3B(\text{Product}) \)
Plan 2b: \( B(\text{Company}) \times T(\text{Product}) \)

Case 1:
- Plan 1: \( 2.5 \times 20 = 50 \)
- Plan 2: \( 500 + 3000 = 3500 \)
- Plan 3: \( 500 + 100000 = 100500 \)

Case 2:
- Plan 1: \( 250 \times 20 = 5000 \)

Lessons

- Need to consider several physical plan
  - even for one, simple logical plan
- No magic “best” plan: depends on the data
- In order to make the right choice
  - need to have statistics over the data
  - the B’s, the T’s, the V’s