Lecture 22:
Query Execution

Monday, March 6, 2006

Outline

- Query execution: 15.1 – 15.5
Architecture of a Database Engine

Logical Algebra Operators

- Union, intersection, difference
- Selection $\sigma$
- Projection $\Pi$
- Join $|x|$
- Duplicate elimination $\delta$
- Grouping $\gamma$
- Sorting $\tau$
Logical Query Plan

```
SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100
```

\[ T1, T2, T3 = \text{temporary tables} \]

Logical Query Plan

```
SELECT P.buyer
FROM Purchase P, Person Q
WHERE P.buyer=Q.name AND
    P.city='seattle' AND
    Q.phone > '5430000'
```

```
\[ T1, T2, T3 = \text{temporary tables} \]
```
Physical Query Plan

```
SELECT S.buyer
FROM Purchase P, Person Q
WHERE P.buyer=Q.name AND
  Q.city='seattle' AND
  Q.phone > '5430000'
```

Query Plan:
- logical tree
- implementation choice at every node
- scheduling of operations.

Some operators are from relational algebra, and others (e.g., scan) are not.

Question in Class

Logical operator:

```
Product(pname, cname) \times Company(cname, city)
```

Propose three physical operators for the join, assuming the tables are in main memory:

1. 
2. 
3. 
Question in Class

Product(pname, cname) \times Company(cname, city)

- 1000000 products
- 1000 companies

How much time do the following physical operators take if the data is in main memory?

- Nested loop join \quad time =
- Sort and merge = merge-join \quad time =
- Hash join \quad time =

Cost Parameters

The cost of an operation = total number of I/Os
result assumed to be delivered in main memory

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
- M = size of main memory buffer pool, in blocks
Cost Parameters

- **Clustered** table R:
  - Blocks consists only of records from this table
  - \( B(R) \ll T(R) \)
- **Unclustered** table R:
  - Its records are placed on blocks with other tables
  - \( B(R) \approx T(R) \)

- When a is a key, \( V(R,a) = T(R) \)
- When a is not a key, \( V(R,a) \)

Selection and Projection

Selection \( \sigma(R) \), projection \( \Pi(R) \)

- Both are *tuple-at-a-time* algorithms
- Cost: \( B(R) \)
Main Memory Hash Join

Hash join: $R \ |x| \ S$

- Scan $S$, build buckets in main memory
- Then scan $R$ and join

- Cost: $B(R) + B(S)$
- Assumption: $B(S) \leq M$

Duplicate Elimination

Duplicate elimination $\delta(R)$

- Hash table in main memory

- Cost: $B(R)$
- Assumption: $B(\delta(R)) \leq M$
Grouping

Grouping:
Product(name, department, quantity)
\[ \gamma_{\text{department, sum(quantity)}} \text{(Product)} \rightarrow \]
Answer(department, sum)

Main memory hash table
Question: How?

Nested Loop Joins

• Tuple-based nested loop \( R \bowtie S \)

```
for each tuple r in R do
    for each tuple s in S do
        if r and s 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

• We can be much more clever

• Question: how would you compute the join in the following cases? What is the cost?
  
  – $B(R) = 1000$, $B(S) = 2$, $M = 4$
  
  – $B(R) = 1000$, $B(S) = 3$, $M = 4$
  
  – $B(R) = 1000$, $B(S) = 6$, $M = 4$

Nested Loop Joins

• Block-based Nested Loop Join

  \[
  \begin{align*}
  &\text{for each (}M-2\text{) blocks } b_s \text{ of } S \text{ do} \\
  &\quad \text{for each block } b_r \text{ of } R \text{ do} \\
  &\quad \quad \text{for each tuple } s \text{ in } b_s \\
  &\quad \quad \quad \text{for each tuple } r \text{ in } b_r \\
  &\quad \quad \quad \text{if “}r \text{ and } s \text{ join” then output}(r,s)
  \end{align*}
\]
Nested Loop Joins

• Block-based Nested Loop Join
• Cost:
  – Read S once: cost $B(S)$
  – Outer loop runs $B(S)/(M-2)$ times, and each time need to read R: costs $B(S)B(R)/(M-2)$
  – Total cost: $B(S) + B(S)B(R)/(M-2)$
• Notice: it is better to iterate over the smaller relation first
• $R \mid x \mid S$: R=outer relation, S=inner relation
Partitioned Hash Algorithms

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

\[
\begin{align*}
\text{Relation } R & \quad \text{INPUT} \\
1 & \quad \text{hash function } h \\
2 & \\
\cdots & \\
B(R) & \quad \text{OUTPUT} \\
\end{align*}
\]

- Does each bucket fit in main memory?
  - Yes if $B(R)/M \leq M$, i.e. $B(R) \leq M^2$

Duplicate Elimination

- 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: $B(R) \leq M^2$
Grouping

• Recall: $\gamma(R) = $ 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

$R |x| S$

• 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 $h_2 (<> h_1)$. Scan matching partition of S, search for matches.

Partitioned Hash Join

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

• Problem:
• Sort a file of size $B$ with memory $M$
• Where we need this:
  – ORDER BY in SQL queries
  – Several physical operators
  – Bulk loading of $B$-tree indexes.
• Will discuss only 2-pass sorting, for when $B < M^2$

External Merge-Sort: Step 1

• Phase one: load $M$ bytes in memory, sort

![Diagram showing disk and main memory with M bytes, runs of length M bytes]
External Merge-Sort: Step 2

- Merge $M - 1$ runs into a new run
- Result: runs of length $M (M - 1) \approx M^2$

If $B \leq M^2$ then we are done

Cost of External Merge Sort

- Read+write+read = $3B(R)$
- Assumption: $B(R) \leq M^2$
Duplicate Elimination

Duplicate elimination $\delta(R)$

- Idea: do a two step merge sort, but change one of the steps

- Question in class: which step needs to be changed and how?

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

Grouping

Grouping: $\gamma_{a, \text{sum}(b)}(R)$

- Same as before: sort, then compute the sum(b) for each group of a’s
- Total cost: $3B(R)$
- Assumption: $B(R) \leq M^2$
Merge-Join

Join R ∣x| S
• Step 1a: initial runs for R
• Step 1b: initial runs for S
• Step 2: merge and join

\[ M_1 = \frac{B(R)}{M} \text{ runs for } R \]
\[ M_2 = \frac{B(S)}{M} \text{ runs for } S \]

If \( B \leq M^2 \) then we are done
Two-Pass Algorithms Based on Sorting

Join R \(|x| S\)

- 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: \(B(R) + B(S) \leq M^2\)

Index Based Selection

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

- Example:
  - Table scan (assuming R is clustered):
    - $B(R) = 2,000$ I/Os
  - Index based selection:
    - If index is clustered: $B(R)/V(R,a) = 100$ I/Os
    - If index is unclustered: $T(R)/V(R,a) = 5,000$ I/Os

- Lesson: don’t build unclustered indexes when $V(R,a)$ is small!

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)

Summary of External Join Algorithms

- Block Nested Loop: B(S) + B(R)*B(S)/M

- Partitioned Hash: 3B(R)+3B(S);
  - min(B(R),B(S)) <= M^2

- Merge Join: 3B(R)+3B(S)
  - B(R)+B(S) <= M^2

- Index Join: B(R) + T(R)B(S)/V(S,a)