Lecture 21: Query Execution

Friday, May 18, 2007
Outline

• Hash-tables (13.4)
• Query execution: 15.1 – 15.5
Architecture of a Database Engine

- Parse Query
- Select Logical Plan
- Select Physical Plan
- Query Execution

SQL query

Query optimization

Logical plan

Physical plan
Logical Algebra Operators

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

Will learn today and the following lectures:

- **Join:**
  - Main-memory hash based join
  - Block-based nested-loop join
  - Partitioned hash-based join
  - Merge-join
  - Index-join

- **Group-by / Duplicate-elimination:**
  - . . . .
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) |x| 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  \( \text{time} = \)
- Sort and merge = merge-join  \( \text{time} = \)
- Hash join  \( \text{time} = \)
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 \textit{tuple-at-a-time} algorithms
- Cost: $B(R)$
Hash Tables

- Key data structure used in many operators
- May also be used for indexes, as alternative to B+trees
- Recall basics:
  - There are $n$ *buckets*
  - A hash function $f(k)$ maps a key $k$ to $\{0, 1, \ldots, n-1\}$
  - Store in bucket $f(k)$ a pointer to record with key $k$
- Secondary storage: bucket = block, use overflow blocks when needed
Hash Table Example

• Assume 1 bucket (block) stores 2 keys + pointers
• $h(e)=0$
• $h(b)=h(f)=1$
• $h(g)=2$
• $h(a)=h(c)=3$

Here: $h(x) = x \mod 4$
Searching in a Hash Table

- Search for a:
- Compute $h(a)=3$
- Read bucket 3
- 1 disk access
## Insertion in Hash Table

- Place in right bucket, if space
- E.g. $h(d)=2$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>b</td>
<td>d</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f</td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td>g</td>
<td></td>
</tr>
</tbody>
</table>

Example: Insertion of $d$ into the hash table with $h(d)=2$. The table is initialized with buckets at positions $0$, $1$, $2$, and $3$, with characters $e$, $b$, $f$, and $a$ respectively.
Insertion in Hash Table

- Create overflow block, if no space
- E.g. \( h(k) = 1 \)

- More overflow blocks may be needed
Hash Table Performance

• Excellent, if no overflow blocks
• Degrades considerably when number of keys exceeds the number of buckets (i.e. many overflow blocks).
Main Memory Hash Join

Hash join: $R \mid \times \mid S$

- 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 $R \bowtie S$

```plaintext
for each tuple $r$ in $R$ do
  for each tuple $s$ in $S$ do
    if $r$ and $s$ join then
      output $(r,s)$
```

- Cost: $T(R) \cdot B(S)$ when $S$ is clustered
- Cost: $T(R) \cdot T(S)$ 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?

  – $B(R) = 1000, B(S) = 2, M = 4$

  – $B(R) = 1000, B(S) = 3, M = 4$

  – $B(R) = 1000, B(S) = 6, M = 4$
Block-Based Nested-loop Join

for each (M-2) blocks bs of S do
    for each block br of R do
        for each tuple s in bs
            for each tuple r in br do
                if “r and s join” then
                    output(r,s)
Block-Based Nested-loop Join

- R & S
- Hash table for block of S (M-2 pages)
- Input buffer for R
- Output buffer
- Join Result
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
Index Based Join

- \( R \bowtie S \)
- Assume S has an index on the join attribute

```plaintext
for each tuple r in R do
    lookup the tuple(s) s in S using the index
    output (r,s)
```
Index Based Join

Cost (Assuming R is clustered):

- 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 Selection

Selection on equality: $\sigma_{a=v}(R)$

- Clustersed index on a: $\text{cost } B(R)/V(R,a)$
- Unclustered index on a: $\text{cost } T(R)/V(R,a)$
  - We have seen that this is like a join
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!
Operations on Very Large Tables

- Partitioned hash algorithms
- Merge-sort algorithms
Partitioned Hash Algorithms

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

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

• Recall: $\delta(R) =$ 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) = \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 |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 $h_2$ ($\neq h$). Scan matching partition of S, search for matches.
Partitioned Hash Join

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

- Merge M – 1 runs into a new run
- Result: runs of length M (M – 1) ≈ M²

If B <= M² 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 \( \times \) S

- Step 1a: initial runs for R
- Step 1b: initial runs for S
- Step 2: merge and join
Merge-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 $R \mid x \mid S$

- 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: $B(R) + B(S) \leq M^2$
Summary of External Join Algorithms

- Block Nested Loop: $B(S) + B(R) \times B(S)/M$

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

- Partitioned Hash: $3B(R) + 3B(S)$;
  - $\min(B(R), B(S)) \leq M^2$

- Merge Join: $3B(R) + 3B(S)$
  - $B(R) + B(S) \leq M^2$