CS 133: Databases

Fall 2016
Lec 10 – 10/3
Prof. Beth Trushkowsky

Midterm Logistics

• Take-home: October 10 – October 14
  – Available by Monday Oct 10th 1pm
  – Due: Friday Oct 14th 5pm
  – 75 minute exam

– Review class: Oct 10th
– No class Wednesday Oct 12th

Goals for Today

• Learn about the components of query processing and idea of different algorithms for relational operators

• Understand the importance of out-of-core a.k.a. external sorting and hashing algorithms

• Reason about the I/O cost of sorting or hashing algorithms given size of relations and available buffer pool space

Simplified RDBMS Architecture

Application
Query
Query optimizer
Query executor
Access methods
Buffer management
Disk management
Data records

Cost-based Query Sub-system
Cost-based Query Sub-System

Queries: Select * From Sailors S Where S.sid = 42;

Query Parser → Query Optimizer → Query Plan Generator → Plan Cost Estimator → Catalog Manager → Query Plan Evaluator

Usually there is a heuristics-based rewriting step before the cost-based steps.

Logical Query Plan: Example

- Example SQL query:
  ```sql
  SELECT S.sname
  FROM Reserves R, Sailors S
  WHERE R.sid = S.sid
  AND R.bid=100 AND S.rating>5
  ```

- Equivalent Relational Algebra expression:
  \[
  \Pi_{\text{sname}} (\sigma_{\text{bid}=100 \land \text{rating}>5} (\text{Reserves} \bowtie_{\text{sid}=\text{sid}} \text{Sailors}))
  \]

Logical Query Plan: Tree of Iterators

- Each node is a relational operator that implements iterator interface.
- Edges represent inputs (one or two).
- Evaluation starts at the top of the tree: “pull”-based data flow.

Query Plan to Physical Plan

- Logical query plan partially shows us how to evaluate query:
  - Missing: choice of specific algorithm for executing operators
  - Also: ordering of operators
- How to choose?!
  - No one algorithm always best
  - Cost-based optimization
  - Forms physical query plan

- Access path choice: Sequential scan? Indexes?
- Selections: how to apply multiple constraints? Projections: duplicate elim.?
- Join algorithm choice:
  - Simple nested loop
  - Index nested loop
  - Sort-merge?
  - …
Implementing the Project Operator

• Suppose we do not care about removing duplicates

\[
\text{SELECT } R.\text{attribute FROM } R;
\]

• How many I/Os? What would this process look like (with respect to the disk and buffer pool)?

Read in pages of \( R \) one at a time, remove unwanted fields in \textit{one pass} over \( R \)

External Algorithms: Sorting and Hashing

• In various parts of a query plan, important to get “same” tuples \textit{together}
  – \texttt{DISTINCT} (duplicate elimination)
  – \texttt{GROUP BY} (form the groups)
  – Sort-merge \texttt{JOIN} algorithm
  – \texttt{ORDER BY} (user wants output sorted)

• \textbf{Problem}: sort 100GB of data with 1GB of RAM

• Solution: \textit{out-of-core} (external) algorithms that \textit{divide and conquer}
  – Idea: intelligent use of available buffer pool space

Two-Way Sort

• Algorithm operates in a sequence of \textit{passes}
• Pass 0 -- For all pages in file:
  – Read page, sort it in RAM, write the sorted page to disk (don’t overwrite original).
  – Only one buffer page is used in this pass
  – Each sorted page output called a sorted \textit{run}

Two-Way Sort: Passes 1, 2, ...

• Pass 1, 2, ..., etc. (merging):
  – Requires \textit{three buffer pages}: two input, one output
  – \textbf{Merge} pairs of runs into \textit{runs twice as long}
Two-Way External Merge Sort

- Each pass: read + write each page in file.
- \( N \) pages in the file \( \Rightarrow \) number of passes?
  \[ \lceil \log_2 N \rceil + 1 \]
- So total cost is?
  \[ 2N \left( \lceil \log_2 N \rceil + 1 \right) \]
- Idea: Divide, conquer, merge

General External Merge Sort: Passes 1, 2, ...

- In each of Pass 1, 2, etc.: merge B-1 runs
  - Creates runs of (B-1) * (size of runs from previous pass)

General External Merge Sort

*We have more than 3 buffer frames. How can we utilize them?*

- To sort a file with \( N \) pages using \( B \) buffer frames
  - Pass 0: use \( B \) buffer pages. Produce \( \lceil N / B \rceil \) sorted runs of \( B \) pages each

Cost of External Merge Sort

- Cost = \( 2N \times \# \text{ of passes} \)
  - In each pass, read and write each page of file
  - (\( N \) is size of relation in pages)

- Try Exercise (2-3)

- E.g., with 5 buffer pages, to sort 108 page file:
  - Pass 0: \( \lceil 108 / 5 \rceil = 22 \) sorted runs of 5 pages each (last only 3)
  - Pass 1: \( \lceil 22 / 4 \rceil = 6 \) sorted runs of 20 pages each (last only 8)
  - Pass 2: yields 2 sorted runs, 80 pages and 28 pages
  - Pass 3: yields one sorted run of 108 pages

- Number of passes: \( 1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil \)
  - Formula check: \( 1 + \log_4 221 = 1 + 3 \Rightarrow 4 \) passes
Number of Passes with External Sort (with $B$ Buffer Frames and $N$ pages)

<table>
<thead>
<tr>
<th>$N$</th>
<th>$B=3$</th>
<th>$B=5$</th>
<th>$B=9$</th>
<th>$B=17$</th>
<th>$B=129$</th>
<th>$B=257$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,000</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10,000</td>
<td>13</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>100,000</td>
<td>17</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1,000,000</td>
<td>20</td>
<td>10</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10,000,000</td>
<td>23</td>
<td>12</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>100,000,000</td>
<td>26</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>30</td>
<td>15</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Projection (with Duplicate Elim.)

- Suppose Reserves is 1,000 pages, $sid$ and $bid$ make up 25% of each record.

- Basic approach with sorting:
  - 1. Scan R, extract only the needed fields
  - 2. Sort the resulting set
  - 3. Read in, removing adjacent duplicates

- Cost: Reserves with size ratio 0.25 = 250 pages.

Using 20 buffer pages can sort in 2 passes, (ignores cost of final output):

$$1000 + 250 + 2 \times 250 + 250 = 2500 \text{ I/Os}$$

- Can improve by modifying external sort algorithm (Exercise 4):
  - Modify Pass 0 of external sort to eliminate unwanted fields.
  - Modify merging passes to eliminate duplicates.

Sort: Kind of a Big Deal

- FuxiSort: 100 TB in 377 seconds
  - 1,134 nodes x (2 Xeon E5-2630 2.3Ghz, 96 GB memory, 12x2 TB SATA HD, 10 Gb/s Ethernet)
  - 243 nodes x (2 Xeon E5-2650v2 2.6Ghz, 128 GB memory, 12x2 TB SATA HD, 10 Gb/s Ethernet)
  - Jamang Wang, Yongjun Wu, Hua Cali, Zhipeng Tang, Zhiqiang Lv,
    Bin Lu, Yangyu Tao, Chao Li, Jingren Zhou, Hong Tang
    Alibaba Group Inc

- TritonSort: 100 TB for $4.51
  - 330 Amazon EC2 r3.4xlarge nodes x
  - 16 vCores - 2.50GHz Intel Xeon E5-2670 v2, 122 GB memory, 320GB SSD, 8x135GB EBS gp2
  - Michael Conley, Amin Vahdat, George Porter
    University of California, San Diego

(source: sortbenchmark.org)
Alternative: Hashing

- We do not always require order for tuples
  - Removing duplicates
  - Forming groups

- Just need “like” things to be together
  - Hashing!
  - But how to build hash table without staying in RAM?

External Hashing: Divide and Conquer

- Divide: Use a hash function $h_p$ to separate records into disk-based partitions

- Conquer: Read partitions into RAM-based hash table one at a time
  - For each partition, hash with another hash function $h_r$

- Note: Two different hash functions: $h_p$ is coarser-grained than $h_r$

Projection: DupElim Based on Hashing

- Partition phase:
  - Read relation using one input buffer frame, retaining only necessary fields for projection
  - Hashing on $h_p$ to yield B-1 partitions

Projection: DupElim Based on Hashing

- Duplicate Elimination phase
- For each partition:
  - Read in pages
  - Build an in-memory hash table, using second hash function $h_r$, and eliminating duplicates as you go

- If a partition does not entirely fit in buffer pool, need to recursively partition

Note: ignoring small overhead in RAM of hash data structure
Example: Hashing DupElim

- Cost for Projection with DupElim using hashing?
  - assuming each of the partitions formed in first pass fits in buffer pool

- For Reserves query:
  - Read 1000 pages
  - Write out partitions of projected tuples
    • 250 pages, because 25% of record retained
  - Read and do duplicate elimination on each partition
    • total 250 page reads
- Total: $1000 + 250 + 250 = 1500$ I/Os.

Algorithm for Internal Sort

- Quicksort is a fast way to sort in memory.
- An alternative is “tournament sort” (a.k.a. “heapsort”)
  - Idea: create initial sorted runs that are longer than B pages

- Algorithm sketch:
  - Use one page each for input and output, remaining B-2 are the “workspace”
  - Iteratively insert a tuple into output page:
    • Find smallest tuple in workspace that is greater than last tuple placed in output page
    • Use input page to replenish tuples in workspace
  - The current run ends when all tuples in workspace are smaller than last tuple that was output

Example: Internal sort using Heap Sort

- Extend current run:
  - Move 8 and 10 to output
  - Move 12 and 4 from input to workspace
  - Move 12 from workspace to output
  - Replenish input page

- Average length of a run using heap sort is $\sim 2B$

- Quicksort is faster, but ...
  - longer runs often mean fewer passes

Reasoning about Passes

- Exercise 5-6:
  - How big of a relation can we sort or hash in two passes?

- Both: B(B-1) pages
  - Sort: pass 0 yields runs each of size B, then second pass can merge up to B-1 sorted runs
  - Hash: pass 0 creates B-1 partitions, each no more than B pages in size

See Knuth's Art of Computer Programming Vol III: Sorting and Searching