CS 133: Databases

Spring 2017
Lec 10 – 2/16
Query Evaluation
Prof. Beth Trushkowsky

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

Cost-based Query Sub-System

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:

  ```relational
  \( \sigma_{\text{bid}=100 \land \text{rating} > 5} \left( R \bowtie_{\text{sid}=\text{sid}} S \right) \)
  ```

Query Plan: Tree of Iterators

Each node is a relational operator that implements iterator interface

- Evaluation starts at the top of the tree: "pull"-based data flow
- Edges represent inputs (one or two)

Iterator model provides encapsulation:
  • Any operator can be input to any other
  • A parent operator doesn’t need to know what it takes for its child to produce the next() tuple

Logical 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

Implementing the Project Operator

• Suppose we do not care about removing duplicates
  
  ```sql
  SELECT R.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 one pass over R

Join algorithm choice:
  • Simple nested loop
  • Index nested loop
  • Sort-merge?
  • ...

Access path choice:
  Sequential scan? Indexes?
External Algorithms: Sorting and Hashing

- In various parts of a query plan, important to get "same" tuples together
  - DISTINCT (duplicate elimination)
  - GROUP BY (form the groups)
  - Sort-merge JOIN algorithm
  - ORDER BY (user wants output sorted)

**Problem:** sort 100GB of data with 1GB of RAM

**Solution:** out-of-core (external) algorithms that divide and conquer
- Idea: intelligent use of available buffer pool space

---

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

- Pass 1, 2, ..., etc. (merging):
  - Requires three buffer pages: two input, one output
  - Merge pairs of runs into runs twice as long

Two-Way External Merge Sort

- Each pass: read + write each page in file.
- $N$ pages in the file => 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

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

![Diagram](image)

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>

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

- In each of Pass 1, 2, etc.: merge \( B-1 \) runs
  - Creates runs of \( (B-1) \times \text{(size of runs from previous pass)} \)
Sort: Kind of a Big Deal

<table>
<thead>
<tr>
<th>Kind</th>
<th>Big Deal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Gray</td>
<td>Tencent Sort</td>
</tr>
<tr>
<td></td>
<td>- 100 TB in 134 Seconds</td>
</tr>
<tr>
<td></td>
<td>512 nodes x 2 (OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei E5360P V3 1.27B NVMe SSD, 100Gb Mellanox ConnectX4 EN)</td>
</tr>
<tr>
<td></td>
<td>Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao</td>
</tr>
<tr>
<td></td>
<td>Tencent Corporation</td>
</tr>
<tr>
<td></td>
<td>Mark R. Nutter, Jeremy D. Schaub</td>
</tr>
<tr>
<td>Cloud</td>
<td>NADSort</td>
</tr>
<tr>
<td></td>
<td>- 100 TB for $144</td>
</tr>
<tr>
<td></td>
<td>394 Alibaba Cloud ECS ecs.nl.large nodes x</td>
</tr>
<tr>
<td></td>
<td>Haswell E52680 V3, 8 GB memory,</td>
</tr>
<tr>
<td></td>
<td>40GB Ultra Cloud Disk, 4x 135GB SSD Cloud Disk</td>
</tr>
<tr>
<td></td>
<td>Qian Wang, Rong Gu, Yihua Huang</td>
</tr>
<tr>
<td></td>
<td>Nanjing University</td>
</tr>
<tr>
<td></td>
<td>Reynold Xin</td>
</tr>
<tr>
<td></td>
<td>Databricks Inc.</td>
</tr>
<tr>
<td></td>
<td>Wei Wu, Jun Song, Junjuan Xia</td>
</tr>
<tr>
<td></td>
<td>Alibaba Group Inc.</td>
</tr>
</tbody>
</table>

(source: sortbenchmark.org)

Projection (with Duplicate Elim.)

- Suppose Reserves is 1000 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):

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

- 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.

```
SELECT DISTINCT R.sid, R.bid
FROM Reserves R
```

Projection (with Duplicate Elim.)

- Exercise 4: Cost of improved version:
  - Pass 0: read 1000 pages, write out 250 in 13 runs of 20 pages
  - Pass 1: merge runs by reading in 250
  - Total I/Os: 1000+250+250 = 1500s

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

- 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

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

  > Average length of a run is \(~2B\)

> Why would we care?