External Sorting

Chapter 11

2-Way Sort: Requires 3 Buffers
- Pass 1: Read a page, sort it, write it
- only one buffer page is used
- Pass 2, 3, ..., etc.: three buffer pages used.

General External Merge Sort
- More than 3 buffer pages. How can we utilize them?
- To sort a file with \(N\) pages using \(B\) buffer pages:
  - Pass 0: use \(B\) buffer pages. Produce \(\lceil N/B \rceil\) sorted runs of \(B\) pages each.
  - Pass 2, ..., etc.: merge \(B+1\) runs.

Why Sort?
- A classic problem in computer science!
- Data requested in sorted order
  - e.g., find students in increasing GPA order
- Sorting is first step in bulk loading B+ tree index.
- Sorting useful for eliminating duplicate copies in a collection of records (Why?)
- Sort-merge join algorithm involves sorting.
- Problem: sort 1Gb of data with 1Mb of RAM.
  - why not virtual memory?

Two-Way External Merge Sort
- Each pass read + write each page in file.
- \(N\) pages in the file \(\rightarrow\) the number of passes
  \(= \lceil \log_2 N \rceil + 1\)
- So total cost is:
  \[2N \left(\lceil \log_2 N \rceil + 1\right)\]

Cost of External Merge Sort
- Number of passes \(1 + \lceil \log_2 \lceil N/B \rceil \rceil\)
- \(\text{Cost} = 2N \times \text{(# of passes)}\)
- 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 run is only 3 pages)
  - Pass 1: \(\lceil 22/4 \rceil = 6\) sorted runs of 20 pages each
    (last run is only 8 pages)
  - Pass 2: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages
**Number of Passes of External Sort**

<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>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>

*Block size = 32, initial pass produces runs of size 2B.*

**Internal Sort Algorithm**

- Quicksort is a fast way to sort in memory.
- An alternative is “tournament sort” (a.k.a. “heapsort”)
  - **Top**: Read in B blocks
  - **Output**: move smallest record to output buffer
    - Read in a new record r
    - Insert r into “heap”
    - If r not smallest, then **GOTO** Output
    - Else remove r from “heap”
    - Output “heap” in order: **GOTO** Top

**More on Heapsort**

- Fact: average length of a run in heapsort is 2B
  - The “snowpaw” analogy
- Worst-Case:
  - What is min length of a run?
  - How does this arise?
- Best-Case:
  - What is max length of a run?
  - How does this arise?
- Quicksort is faster, but...

**I/O for External Merge Sort**

- ... longer runs often mean fewer passes!
- Actually, do I/O a page at a time
- In fact, read a block of pages sequentially!
- Suggests we should make each buffer (input/output) be a block of pages
  - But this will reduce fanout during merge passes!
  - In practice, most files still sorted in 2-3 passes.

**Number of Passes of Optimized Sort**

<table>
<thead>
<tr>
<th>N</th>
<th>B=1,000</th>
<th>B=5,000</th>
<th>B=10,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,000</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10,000</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>100,000</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1,000,000</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10,000,000</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>100,000,000</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**Double Buffering**

- To reduce wait time for I/O request to complete, can **prefetch** into ‘shadow block’.
  - Potentially, more passes; in practice, most files still sorted in 2-3 passes.
**Sorting Records!**

- Sorting has become a blood sport!
- Parallel sorting is the name of the game...
- Data size: Sort 1M records of size 100 bytes
  - Typical DBMS: 15 minutes
  - Would record: 35 seconds
    - 12-CPU SGI machine, 16 disks, 2GB of RAM
- New benchmarks proposed:
  - Minute Sort: How many can you sort in 1 minute?
  - Dollar Sort: How many can you sort for $1.00?

**Using B+ Trees for Sorting**

- Scenario: Table to be sorted has B+ tree index on sorting column(s)
- Idea: Can retrieve records in order by traversing leaf pages
- Is this a good idea?
- Cases to consider:
  - B+ tree is clustered: Good idea!
  - B+ tree is not clustered: Could be a very bad idea!

**Clustered B+ Tree Used for Sorting**

- Cost: root to the leftmost leaf, then retrieve all leaf pages
  (Alternative 1)
- If Alternative 2 is used? Additional cost of retrieving data records on each page fetched just once.

 ↔ Always better than external sorting!

**Unclustered B+ Tree Used for Sorting**

- Alternative (2) for data entries: each data entry contains rid of a data record. In general, one I/O per data record!

**Summary**

- External sorting is important; DBMS may dedicate part of buffer pool for sorting!
- External merge sort minimizes disk I/O cost:
  - Pass 0: Produces sorted runs of size B (# buffer pages)
  - Later passes: merge runs.
  - # of runs merged at a time depends on B and block size.
  - Larger block size means less I/O cost per page.
  - Larger block size means smaller # runs merged.
  - In practice, # of runs rarely more than 2 or 3.

<table>
<thead>
<tr>
<th>N</th>
<th>Sorting</th>
<th>p=1</th>
<th>p=10</th>
<th>p=100</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>200</td>
<td>100</td>
<td>1,000</td>
<td>10,000</td>
</tr>
<tr>
<td>1,000</td>
<td>2,000</td>
<td>1,000</td>
<td>10,000</td>
<td>100,000</td>
</tr>
<tr>
<td>10,000</td>
<td>40,000</td>
<td>10,000</td>
<td>100,000</td>
<td>1,000,000</td>
</tr>
<tr>
<td>100,000</td>
<td>600,000</td>
<td>100,000</td>
<td>1,000,000</td>
<td>10,000,000</td>
</tr>
<tr>
<td>1,000,000</td>
<td>8,000,000</td>
<td>1,000,000</td>
<td>10,000,000</td>
<td>100,000,000</td>
</tr>
<tr>
<td>10,000,000</td>
<td>80,000,000</td>
<td>10,000,000</td>
<td>100,000,000</td>
<td>1,000,000,000</td>
</tr>
</tbody>
</table>

- p: # of records per page
- B=1,000 and block size=12 for sorting
- p=100 is the more realistic value.
Summary, cont.

- Choice of internal sort algorithm may matter:
  - Quicksort: Quick!
  - Heap/tournament sort: slower (2x), longer runs
- The best sorts are wildly fast:
  - Despite 40+ years of research, we’re still
    improving!
- Clustered B+ tree is good for sorting;
  unclustered tree is usually very bad