External Sorting

UVic C SC 370

Dr. Daniel M. German
Department of Computer Science
University of Victoria

July 2, 2003 Version: 1.1.0

Overview

- Why do we sort?
- Why is in-memory sorting different from on-disk sorting?
- How does external merge sort work?

Introduction

The DBMS needs to sort all the time:

- `order by`
- Eliminating duplicates
- Performing **join**
- Bulk loading of cluster data and tree indexes

A simple two way sort

- This is an over simplification of the way the DBMS sorts data.
- We assume we have only 3 pages to our disposal. What do we do with them?

Two-way sort

- Divide and conquer: divide the file in smaller, sorted subfiles (called **runs**)
- First zero:
  - Read one page at a time.
  - Sort it in memory (most likely use qsort)
  - Write page back to disk (run of size 1 page)
- *While we still have runs to sort* (pass i)
  - **Merge** runs from previous pass into runs of twice the size
  - Use one page for reading one run, another page for the other run, and one for writing

Two-way sort

- Complexity:
  - Assume 2^k pages
  - Pass i produces 2^{k-i} runs of 2^i pages each
  - Last pass (k) produces 1 run of 2^k pages
- How many disk accesses?
  - In each pass we read and write each page once.
  - We have $\lceil log_2 N \rceil + 1$ passes
  - We have to use: $2N(\lceil log_2 N \rceil + 1)$ I/O operations
- See figures 13.2 and 13.3 for an example
- What do we do if we have more buffer space available?

External Merge Sort

- Assume we have B buffer pages available in memory and we need to sort N pages
- Pass 0:
  - Read in B pages at a time, creating $\lceil N/B \rceil$ runs of B pages each of B pages each
- *While we still have runs to sort* (pass i)
  - Use $B - 1$ buffer pages for input and use the other page for output: we do a $(B - 1)$ way merge in each pass (use $B - 1$ for read $B - 1$ runs, one for writing result)
- See figures 13.4 and 13.5 in book

External Merge Sort Complexity

- Complexity:
  - Pass 0 produces $run_0 = \lceil N/B \rceil$ runs
  - Pass 1 produces $run_1 = \lceil run_0/(B - 1) \rceil$ runs
  - Pass 2 produces $run_2 = \lceil run_1/(B - 1) \rceil$ runs
  - Pass k produces $run_k = \lceil run_{k-1}/(B - 1) \rceil = 1$ runs
- Number of passes: $\lceil log_{B-1} run_0 \rceil + 1$
- Ergo: $\lceil log_{B-1} \lceil N/B \rceil \rceil + 1$
- Again, in each pass we read and write each page, so we use $2N(\lceil log_{B-1} \lceil N/B \rceil \rceil + 1)$ I/O operations

Where are the savings?

N	B = 3	B = 5	B = 9	B = 17	B = 129	B = 257
10^2	7	4	3	2	1	1
10^3	10	5	4	3	2	2
10^4	13	7	5	4	2	2
10^5	17	9	6	5	3	3
10^6	20	10	7	5	3	3
10^7	23	12	8	6	4	3
10^8	26	14	9	7	4	4
10^9	30	15	10	8	5	4

Sorting can be made faster

- By using more sophisticated algorithms
- By using pre-fetching and clustered IO