CIS 5930 Advanced Topics in Data Management

Feifei Li

Fall 2008

(Many slides were made available by Ke Yi)
Massive Data

- Massive datasets are being collected everywhere
- Storage management software is a billion-$ industry

Examples (2002):
- **Phone**: AT&T 20TB phone call database, wireless tracking
- **Consumer**: WalMart 70TB database, buying patterns
- **WEB**: Web crawl of 200M pages and 2000M links, Akamai stores 7 billion clicks per day
- **Geography**: NASA satellites generate 1.2TB per day
Example: LIDAR Terrain Data

- Massive (irregular) point sets (1-10m resolution)
  - Becoming relatively cheap and easy to collect
- Appalachian Mountains between 50GB and 5TB
- Exceeds memory limit and needs to be stored on disk
Example: Network Flow Data

- AT&T IP backbone generates 500 GB per day
- Gigascope: A data stream management system
  - Compute certain statistics

- Can we do computation without storing the data?
Random Access Machine Model

- Standard theoretical model of computation:
  - Infinite memory
  - Uniform access cost
- Simple model crucial for success of computer industry
Modern machines have complicated memory hierarchy
- Levels get larger and slower further away from CPU
- Data moved between levels using large blocks
Slow I/O

• Disk access is $10^6$ times slower than main memory access

  “The difference in speed between modern CPU and disk technologies is analogous to the difference in speed in sharpening a pencil using a sharpener on one’s desk or by taking an airplane to the other side of the world and using a sharpener on someone else’s desk.” (D. Comer)

  4835 1915 5748 4125

• Disk systems try to amortize large access time transferring large contiguous blocks of data (8-16Kbytes)

• Important to store/access data to take advantage of blocks (locality)
Scalability Problems

• Most programs developed in RAM-model
  – Run on large datasets because OS moves blocks as needed

• Moderns OS utilizes sophisticated paging and prefetching strategies
  – But if program makes scattered accesses even good OS cannot take advantage of block access

Scalability problems!
Solution 1: Buy More Memory

- Expensive
- (Probably) not scalable
  - Growth rate of data is higher than the growth of memory
Solution 2: Cheat! (by random sampling)

- Provide approximate solution for some problems
  - average, frequency of an element, etc.
- What if we want the exact result?
- Many problems can’t be solved by sampling
  - maximum, and all problems mentioned later
Solution 3: Using the Right Computation Model

- External Memory Model
- Streaming Model
- Uncertain Data Model
External Memory Model

\[ N = \# \text{ of items in the problem instance} \]
\[ B = \# \text{ of items per disk block} \]
\[ M = \# \text{ of items that fit in main memory} \]
\[ T = \# \text{ of items in output} \]

I/O: Move block between memory and disk

We assume (for convenience) that \( M > B^2 \)
Fundamental Bounds

<table>
<thead>
<tr>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanning: $N$</td>
<td>$\frac{N}{B}$</td>
</tr>
<tr>
<td>Sorting: $N \log N$</td>
<td>$\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}$</td>
</tr>
<tr>
<td>Permuting: $N$</td>
<td>$\min{ N, \frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B} }$</td>
</tr>
<tr>
<td>Searching: $\log_2 N$</td>
<td>$\log_B N$</td>
</tr>
</tbody>
</table>

Note:
- Linear I/O: $O(N/B)$
- Permuting not linear
- Permuting and sorting bounds are equal in all practical cases
- $B$ factor VERY important: $\frac{N}{B} < \frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B} << N$
- Cannot sort optimally with search tree
Queues and Stacks

- **Queue:**
  - Maintain push and pop blocks in main memory
  
  \[ O(1/B) \] Push/Pop operations

- **Stack:**
  - Maintain push/pop block in main memory
  
  \[ O(1/B) \] Push/Pop operations
Puzzle #1: Majority Counting

- A huge file of characters stored on disk
- Question: Is there a character that appears > 50% of the time
- Solution 1: sort + scan
  - A few passes (O(\(\log_{M/B} N\))): will come to it later
- Solution 2: divide-and-conquer
  - Load a chunk in to memory: \(N/M\) chunks
  - Count them, return majority
  - The overall majority must be the majority in >50% chunks
  - Iterate until < \(M\)
  - Very few passes (O(\(\log_{M} N\)), geometrically decreasing
- Solution 3: O(1) memory, 2 passes (answer to be posted later)
Sorting

• \(<M/B\) sorted lists (queues) can be merged in \(O(N/B)\) I/Os

\[M/B\] blocks in main memory
Sorting

• Merge sort:
  – Create \( N/M \) memory sized sorted lists
  – Repeatedly merge lists together \( \Theta(M/B) \) at a time

\[ O(\log_{M/B} \frac{N}{M}) \text{ phases using } O\left(\frac{N}{B}\right) \text{ I/Os each } \Rightarrow O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right) \text{ I/Os} \]
2-Way Sort: Requires 3 Buffers

- Phase 1: PREPARE.
  - Read a page, sort it, write it.
  - only one buffer page is used
- Phase 2, 3, …, etc.: MERGE:
  - three buffer pages used.

![Diagram](image.png)
Two-Way External Merge Sort

Idea: Divide and conquer: sort subfiles and merge into larger sorts

Input file
- PASS 0
  - 1-page runs
  - PASS 1
    - 2-page runs
    - PASS 2
      - 4-page runs
      - PASS 3
        - 8-page runs
Two-Way External Merge Sort

- Costs for pass: all pages
- # of passes: height of tree
- Total cost: product of above

Input file
PASS 0
1-page runs
PASS 1
2-page runs
PASS 2
4-page runs
PASS 3
8-page runs

1,2
2,3
3,4
4,5
6,6
7,8
9

1,3
2

PASS 2

PASS 3

PASS 1

PASS 0

Input file

1,2
2,3
3,4
4,5
6,6
7,8
9

1,3
2

PASS 2

PASS 3

PASS 1

PASS 0

Input file
Two-Way External Merge Sort

- Each pass we read + write each page in file.
- N/B pages in file => 2N/B

\[ \text{Number of passes} = \lceil \log_2 \frac{N}{B} \rceil + 1 \]

- So total cost is:

\[ 2N / B \left( \lceil \log_2 \frac{N}{B} \rceil + 1 \right) \]
External Merge Sort

- What if we had more buffer pages?
- How do we utilize them wisely?

→ Two main ideas!
Phase 1: Prepare

• Construct as large as possible starter lists.
Phase 2: Merge

Compose as many sorted sublists into one long sorted list.
General External Merge Sort

*How can we utilize more than 3 buffer pages?*

- To sort a file with $N/B$ pages using $M/B$ buffer pages:
  - Pass 0: use $M/B$ buffer pages. Produce sorted runs of $M/B$ pages each. $\lceil N / B \rceil$
  - Pass 1, 2, …, etc.: merge $M/B-1$ runs.

```
Disk
```
```
INPUT 1
```
```
INPUT 2
```
```
INPUT M/B-1
```
```
OUTPUT
```
```
INPUT 1
```
```
INPUT 2
```
```
INPUT M/B-1
```
```
OUTPUT
```
```
Disk
```

$M/B$ Main memory buffers
Selection Algorithm

• In internal memory (deterministic) quicksort split element (median) found using linear time selection

• Selection algorithm: Finding $i$’th element in sorted order
  1) Select median of every group of 5 elements
  2) Recursively select median of $\sim N/5$ selected elements
  3) Distribute elements into two lists using computed median
  4) Recursively select in one of two lists

• Analysis:
  – Step 1 and 3 performed in $O(N/B)$ I/Os.
  – Step 4 recursion on at most $\sim \frac{7}{10} N$ elements
  $\Rightarrow T(N) = O(\frac{N}{B}) + T(\frac{N}{5}) + T(\frac{7N}{10}) = O(\frac{N}{B})$ I/Os
Toy Experiment: Permuting

• Problem:
  – Input: $N$ elements out of order: 6, 7, 1, 3, 2, 5, 10, 9, 4, 8
    * Each element knows its correct position
  – Output: Store them on disk in the right order

• Internal memory solution:
  – Just scan the original sequence and move every element in the right place!
  – $O(N)$ time, $O(N)$ I/Os

• External memory solution:
  – Use sorting
  – $O(N \log N)$ time, $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$
Takeaways

• Need to be very careful when your program’s space usage exceeds physical memory size

• If program mostly makes highly localized accesses
  – Let the OS handle it automatically

• If program makes many non-localized accesses
  – Need I/O-efficient techniques

• Three common techniques (recall the majority counting puzzle):
  – Convert to sort + scan
  – Divide-and-conquer
  – Other tricks