Lecture 16: Data Storage

Wednesday, November 6, 2006

Outline

- Data Storage
  - The memory hierarchy – 11.2
  - Disks – 11.3
  - Merge sort – 11.4

The Memory Hierarchy

Main Memory
- Volatile
- Limited address spaces
- Expensive
- Average access time: 10-100 nanoseconds

Cache:
- Access time 10 nanos

Disk
- 5-10 MB/S transmission rates
- 2-10 GB storage
- Average time to access a block: 10-15 usecs.
  - Need to consider seek, rotation, transfer times.
  - Keep records “close” to each other.

Tape
- 1.5 MB/S transfer rate
- 280 GB typical capacity
- Only sequential access
- Not for operational data

Main Memory
- Fastest, most expensive
- Today: 512MB are common on PCs
- Many databases could fit in memory
  - New industry trend: Main Memory Database
    - E.g TimesTen, DataBlitz
- Main issue is volatility
  - Still need to store on disk

Secondary Storage
- Disks
- Slower, cheaper than main memory
- Persistent !!!
- Used with a main memory buffer

Buffer Management in a DBMS

Page Requests from Higher Levels

- Data must be in RAM for DBMS to operate on it!
- Table of <frame#, pageid> pairs is maintained.
- LRU is not always good.
Buffer Manager

Manages buffer pool: the pool provides space for a limited number of pages from disk.

Needs to decide on page replacement policy
- LRU
- Clock algorithm

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Tertiary Storage

- Tapes or optical disks
- Extremely slow: used for long term archiving only

The Mechanics of Disk

Mechanical characteristics:
- Rotation speed (5400RPM)
- Number of platters (1-30)
- Number of tracks (<10000)
- Number of bytes/track(10^9)

Disk Access Characteristics

- Disk latency = time between when command is issued and when data is in memory
- Disk latency = seek time + rotational latency
  - Seek time = time for the head to reach cylinder
    - 10ms - 40ms
  - Rotational latency = time for the sector to rotate
    - Rotation time = 10ms
    - Average latency = 10ms/2
- Transfer time = typically 10MB/s
- Disks read/write one block at a time (typically 4kB)

Average Seek Time

Suppose we have N tracks, what is the average seek time?

- Getting from cylinder x to y takes time $|x-y|$

$$\frac{1}{N} \int_0^N (x-y) \, dx = \frac{1}{N} \left( \frac{y^2}{2} \right) \bigg|_0^N = \frac{1}{2} \left( \frac{N^2}{2} \right) = \frac{N^2}{4}$$

$$\frac{1}{N} \int_0^N (x-y) \, dx \bigg|_0^N = \frac{1}{N} \int_0^N (y-x) \, dx \bigg|_0^N = \frac{1}{2} \left( \frac{N^2}{2} \right) = \frac{N^2}{4}$$

$$\frac{1}{N} \left( \frac{N^2}{2} \right) + \frac{1}{N} \left( \frac{N^2}{2} \right) = \frac{N}{2}$$

- Hence, may also be able to perform prefetching
- DBMS needs the ability to force pages to disk.
The I/O Model of Computation

- In main memory algorithms we care about CPU time
- In databases time is dominated by I/O cost
- Assumption: cost is given only by I/O
- Consequence: need to redesign certain algorithms
- Will illustrate here with sorting

Sorting

- Illustrates the difference in algorithm design when your data is not in main memory:
  - Problem: sort 1Gb of data with 1Mb of RAM.
- Arises in many places in database systems:
  - Data requested in sorted order (ORDER BY)
  - Needed for grouping operations
  - First step in sort-merge join algorithm
  - Duplicate removal
  - Bulk loading of B+-tree indexes

2-Way Merge-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.

Two-Way External Merge Sort

- Each pass we read + write each page in file.
- N pages in the file => the number of passes
  $= \lceil \log_2 N \rceil + 1$
- So total cost is:
  $2N \left( \lceil \log_2 N \rceil + 1 \right)$
- Improvement: start with larger runs
- Sort 1GB with 1MB memory in 10 passes

Can We Do Better?

- We have more main memory
- Should use it to improve performance

Cost Model for Our Analysis

- **B**: Block size
- **M**: Size of main memory
- **N**: Number of records in the file
- **R**: Size of one record
External Merge-Sort

- Phase one: load M bytes in memory, sort
  - Result: runs of length M/R records

![Diagram of External Merge-Sort](image1)

Phase Two

- Merge M/B – 1 runs into a new run
- Result: runs have now M/R (M/B – 1) records

![Diagram of Phase Two](image2)

Phase Three

- Merge M/B – 1 runs into a new run
- Result: runs have now M/R (M/B – 1)^2 records

![Diagram of Phase Three](image3)

Cost of External Merge Sort

- Number of passes: \(1 + \left\lceil \log_{\frac{M}{B}} \left\lceil \frac{NR}{M} \right\rceil \right\rceil\)
- Think differently
  - Given B = 4KB, M = 64MB, R = 0.1KB
  - Pass 1: runs of length M/R = 640000
    - Have now sorted runs of 640000 records
  - Pass 2: runs increase by a factor of M/B – 1 = 16000
    - Have now sorted runs of 10,240,000,000 = 10^{10} records
  - Pass 3: runs increase by a factor of M/B – 1 = 16000
    - Have now sorted runs of 10^{10} records
  - Nobody has so much data!
- Can sort everything in 2 or 3 passes!

![Diagram of Cost of External Merge Sort](image4)

External Merge Sort

- The `xsort` tool in the XML toolkit sorts using this algorithm
- Can sort 1GB of XML data in about 8 minutes

![Diagram of External Merge Sort](image5)