Advanced Database Technology
February 6, 2006

DATA STORAGE

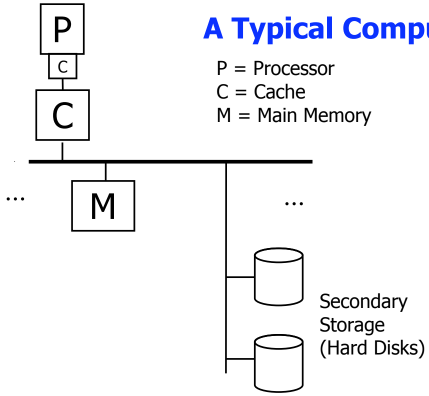
(Lecture based on [GUW 11.2-11.5], [Sanders03, 1.3-1.5], and [MaheshwariZeh03, 3.1-3.2])

Slides based on
Notes 02: Hardware
for Stanford CS 245, fall 2002
by Hector Garcia-Molina

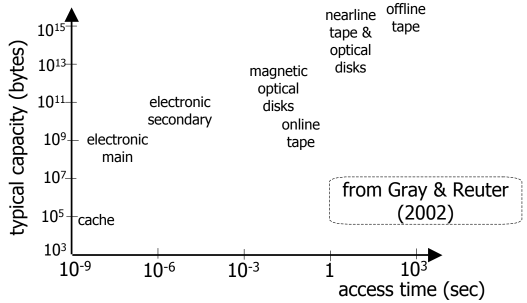
Today

- Hardware, external memory
- Disk access times
- The I/O model
- Case study: Sorting
- Lower bound for sorting
- Optimizing disk usage

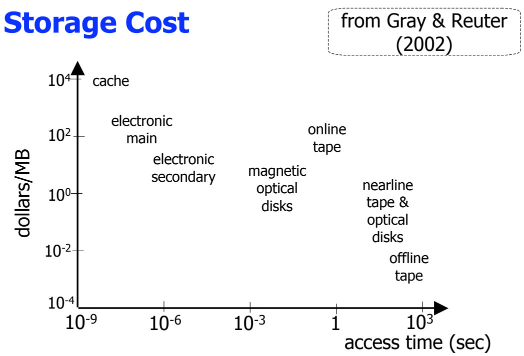
A Typical Computer



Storage Capacity



Storage Cost



Caching

Cache: Memory holding *frequently used* parts of a *slower, larger* memory.

Examples:
- A small (L1) cache holds a few kilobytes of the memory "most recently used" by the processor.
- Most operating systems keep the most recently used "pages" of memory in main memory and put the rest on disk.

Virtual memory

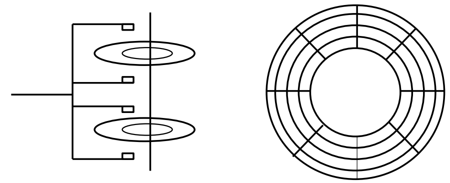
- In most operating systems, programs don't know if they access main memory or a page that resides on secondary memory.
- This is called **virtual memory** (the book is a little fuzzy on this).
- Database systems often take explicit control over secondary memory accesses.

Secondary storage

Traditionally many flavours:
- Disk: Floppy, **Winchester**, Optical, CD-ROM (arrays), DVD-R,…
- Tape: Reel, cartridge robots

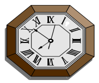
Other: Storage Area Networks (SANs)

Typical Disk



Terms:
- Platter, Head, Cylinder, Track, Sector (all physical).
- Block (logical).

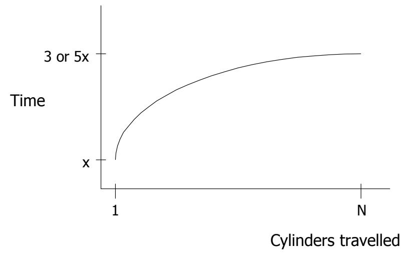
Disk Access Time

I want block X →  → block x in memory

?

Time = Seek Time +
Rotational Delay +
Transfer Time +
Other

Seek Time



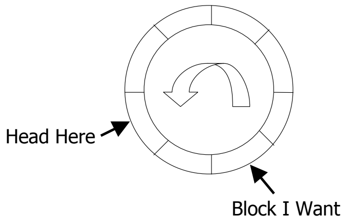
Average Random Seek Time

$$S = \frac{\sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \text{SEEKTIME}(i \rightarrow j)}{N(N-1)}$$

Typical S on the order of 10 ms

10s of millions clock cycles!

Rotational Delay



Average Rotational Delay

R = 1/2 revolution

Typical R = 4.16 ms (7200 RPM)

Transfer Rate (t)

- typical t: 10 → 50 MB/second
- transfer time: $\frac{\text{block size}}{t}$
- E.g., block size 32 kB, t=32 MB/second gives transfer time 1 ms.

Other Delays

- CPU time to issue I/O
- Contention for controller
- Contention for bus, memory

Typical value: Nearly 0

- **So far:** Random Block Access
- **What about:** Reading the **next** block?

If we don't need to change cylinder

$$\text{Time to get block} = \frac{\text{Block Size}}{t} + \text{Negligible}$$

- switch track
- once in a while,
  next cylinder

Rule of thumb
Random I/O: Expensive
Sequential I/O: Less expensive

- Example: 1 KB Block
  - » Random I/O: ~ 20 ms.
  - » Sequential I/O: ~ 1 ms.

However, the relative difference is smaller if we use larger blocks.

Cost for *Writing* similar to *Reading*

If we want to verify:
Need to add (full) rotation + $\frac{\text{Block size}}{t}$

To *Modify* a Block:

(a) Read Block
(b) Modify in Memory
(c) Write Block
[(d) Verify?]

Short problem session

We usually express the running time of algorithms as the number of operations.

- Argue that this can be misleading when data is stored on disk. Consider:
  - Adding a list of integers stored in an array.
  - Adding a list of integers stored in a linked list.
- What should we count instead?
- How do we do the counting?

The I/O model of computation

- Count the number of disk blocks read or written by an algorithm (I/Os).
- Ignore the number of operations! (?)
- Explicit control of which blocks are in main memory.
- **Notation**: M = size of main memory
  B = size of disk blocks
  N = size of data set

A successful model

- Used through the last 15 years to:
  - Better understand the possibilities and limitations of disk-based algorithms.
  - Devise external memory algorithms that are also very efficient in practice.
- Increasingly, the model is also relevant for the **internal** memory block structure of modern computers.
- Recently: An elegant "**cache-oblivious**" model of the whole memory hierarchy.

Today's case study: Sorting

- Used as subroutine in many algorithms, especially in a DBMS.
- Often, a solution using sorting can be shown to be asymptotically optimal.
- Highlights many of the key differences between internal memory and the I/O model.

Problem session

Consider the following sorting algorithms:

- Mergesort
- Quicksort (assume: splitting always "good")

- What are the running times in internal memory?
- What is the number of I/Os if we use the algorithm on external memory
  - if no caching is done?
  - when storing the M/B most recently accessed blocks in main memory?

External memory sorting

Let's see if we can do better!

Design guidelines for external memory algorithms:

- Achieve **spatial** locality (things stored in a block "belong together", and we can use it all)
- Achieve **temporal** locality (at any point, most of the contents of internal memory is "relevant" for the current step of the algorithm)

Sorting in GUW

- More practical viewpoint: Two passes over the data enough unless data is huge.
- TPMMS = **T**wo-**p**ass **m**ulti-way **m**erge**s**ort.
- More general treatment in [MaheshwariZeh03, 3.2]

Problem session

Now it is your turn:

- Analyse selection sort in external memory.
- Make an improved external memory selection sort.
- Details on exercise sheet handed out.

External memory sorting

Let's see if we did as well as we possibly could:

A lower bound on the number of I/Os for sorting

(based on [Sanders03, 1.5])

Optimizations

Some practically oriented ways of speeding up external memory algorithms:

- Disk Scheduling
  - e.g., using the "elevator algorithm"
  - reduces average seek time when there are multiple simultaneous "unpredictable" requests
- Track buffer / cylinder buffer
  - good when data are arranged and accessed "by cylinder"

- Pre-fetching
  - Speeds up access when the needed blocks are known, but the order of requests is data dependent
- Disk arrays
  - Increases the rate at which data can be transferred
  - **Striping**: Blocks from each disk are grouped to form a large logical block
- Mirrored disks
  - Same speed for writing
  - Several blocks can be read in parallel

- Randomized placement of blocks on disk
  - Not mentioned in GUW
  - Based on recent research (Sanders et al.)
  - Implementation and experiments would make a great (thesis) project!

Block Size Selection

- Big Block $\rightarrow$ Amortize I/O Cost
- Big Block $\Rightarrow$ Read in more useless stuff!
Takes longer to read
- **However:** Blocks get bigger

Problem session

Which of the mentioned optimizations apply to (parts of) the optimal sorting algorithm we developed earlier?

- Disk scheduling
- Cylinder buffer
- Pre-fetching
- Disk arrays
- Mirrored disks

Summary

- External memories are complicated.
- Essential features captured by the I/O model (to be used henceforth).
- We saw matching I/O upper and lower bounds for sorting.
- A bit (and the last bit) about optimizing constant factors in external memory access.

Next week (Srinivasa)

- How relations are stored on external memory.
- Simple index structures.

- Remember: **Hand-in due Feb. 20**, on using sorting to make a compromise between bag and set representations.