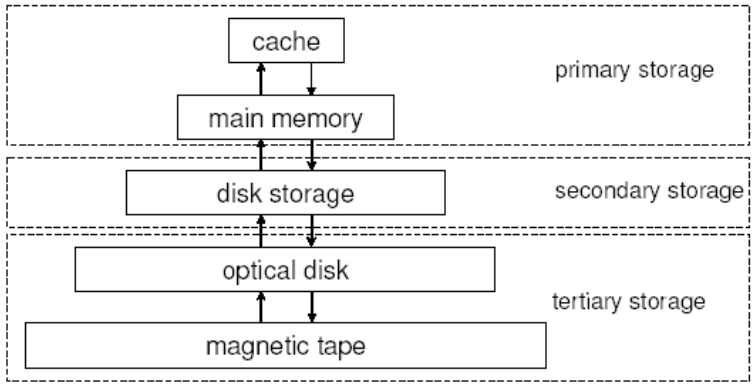
Storage Hierarchy

Storage Hierarchy

Primary, secondary and tertiary storage



Primary Storage

Primary Storage consists of main memory and processor cache

Very fast, for modern main memory:

- sequential read: 50 GByte/s
- latency: 100ns

Data can be directly processed by CPU

Access to data with fine granularity: each byte can be addressed

Number of accessible bytes depends on address scheme: 32-bit address scheme implies that only 2^{32} bytes are addressable

Volatile, non-reliable storage media

Secondary Storage

hard disk storage or SSD

stable, non-volatile, reliable

much larger, e.g. 10 TByte per medium

By orders of magnitude cheaper

Data can not be directly processed

Access granularity is coarse: blocks of e.g., 4 KBytes

Much slower, even for modern drives:

- for HDD: sequential read: up to 280 MByte/s, latency: 1-30 ms
- for SSD: sequential read: up to 7000 MByte/s, latency: 10-100 µs

Necessary:

- good buffer management (high hit ratio)
- good query management

SDD versus HDD

Advantages SSD:

- faster
- physically smaller

Advantages HDD:

- cheaper

In particular the cost limits the use of SSDs to store large amounts of data

- SSDs dominate for storing boot-partitions and small amounts of data
- HDDs still frequently used for storing large databases

Sales figures for 2024:

- the number of sold drives is quite similar for HDDs and SSDs
- the sold capacity of HDDs is around three times the capacity of SSDs

Tertiary Storage

Usage:

- for long-term archival storage or
- short-term logging (journals) of database updates

Media: optical discs, magnetic tapes, hard drives

Size of a magnetic tape: several terabytes

Medium is typically switched: “offline storage”

Disadvantage: access gap extremely large: manually access medium, insert medium

Performance Metrics (1)

Crucial for the performance of a DBMS is the transition between main memory (primary storage) and secondary storage

- to work with data, it has to be in primary storage (in a region called buffer), but:
- main memory is not persistent
- the main memory buffer is typically much smaller than the database. Consequence: We have to discard data from the buffer to load other parts of the database into the buffer; modified data has to transferred to secondary storage before it can be discarded
- Result: Data has to be transferred frequently between main memory and secondary storage.

Data is transferred between main memory and secondary storage in units called blocks (size depends on media, e.g., 4KByte).

Performance Metrics (2)

A disk I/O (read or write of a block) is very expensive compared with what is likely to be done with the block once it arrives in main memory.

- Perhaps 1,000,000 machine instructions in the time to do one random disk I/O.
- Random block accesses is the norm if there are several processes accessing disks, and the disk controller does not schedule accesses carefully.

Reasonable model of computation that requires secondary storage: count only the disk I/O's.

Good DBMS algorithms

a) Try to make sure that if we read a block, we use much of the data on the block.

b) Try to put blocks that are accessed together in physically adjacent locations.

c) Try to buffer commonly used blocks in main memory.

Sorting Example

Setup:

- 10^7 records of 100 bytes = 10^9 bytes file.
  - Stored on a disk with 4KByte blocks, each holding 40 records + header information.
  - Entire file takes 250,000 blocks.
- 50MByte available main memory = 12,800 blocks ≈ 1/20th of file.
- Task: Sort records of file by primary key field.

Merge Sort

Common main memory sorting algorithms do not perform well when you take disk I/O's into account. Variants of Merge Sort do better.

- Merge = take two sorted lists and repeatedly chose the smaller of the 'heads" of the lists (head = first of the unchosen).
- Example: merge 1,3,4,8 with 2,5,7,9 = 1,2,3,4,5,7,8,9.
- Merge Sort based on recursive algorithm:
  - divide records into two parts;
  - recursively merge sort the parts, and merge the resulting lists.

Two Phase, Multiway Merge Sort (1)

Plain merge sort is still not very good in number of disk I/O's.

- $\log_2(n)$ passes, so each record is read/written from disk $\log_2(n)$ times.

Better variant: 2PMMS

- 2 reads + 2 writes per block.

Two Phase, Multiway Merge Sort (2)

Phase 1

- 1. Fill main memory with records.
- 2. Sort using favorite main memory sort.
- 3. Write sorted sublist to disk.
- 4. Repeat until all records have been put into one of the sorted lists.

Two Phase, Multiway Merge Sort (3)

Phase 2

- Use one buffer for each of the sorted sublists and one buffer for an output block.
- Initially load input buffers with the first blocks of their respective sorted lists.
- Repeatedly run a competition among the first unchosen records of each of the buffered blocks.
  - Move the record with the least key to the output block; it is now "chosen."
- Manage the buffers as needed:
  - If an input block is exhausted, get the next block from the same file.
  - If the output block is full, write it to disk.

Analysis

We count the I/O's:

- File stored on 250,000 blocks, read and written once in each phase.
- 4 * 250,000 I/O's = 1,000,000 I/O's

Depending on how the data is organized on the storage media, the cost of an individual I/O can vary!!!

Remaining question: How many records can you sort with 2PMMS, under our assumptions about records, main memory, and the disk? What would you do if there were more?

Extension of 2PMMS to Larger Relations

Possible size of relation can be calculated as follows

- block size: B bytes
- main memory available for buffering blocks: M bytes
- size of record: R bytes

Then

- number of available buffer blocks in main memory: M/B
- during the second phase all but one buffer may be devoted to one of the sorted sublists; the remaining buffer is for the output block
  - maximum number of sorted sublists is (M/B) – 1
- Each sublist completely fits into main memory in phase one
  - sublist can contain M/R records
- Total number of records we can sort: $(M/R)\ ((M/B) - 1) \approx M^2 / (R * B)$
- In our example: $6.71 * 10^9$ records

If this is not enough: add a third pass