Scalable Systems and Data (70022)

1.1 - Database Storage Layer

This lecture covers the DBMS layers, storage hierarchy as well as the role disks takes in the DBMS. The DBMS layers are as follows, with the query going into the first layer, and storage being the final layer. Note that the last two layers (buffer management and disk space management are typically done by the OS).

1. **query optimization and execution** tries to reorganise the query to execute efficiently
2. **relational operators**
3. **file and access methods** understand which files need to be accessed and indexes we can use
4. **buffer management** handles reading disk pages and buffering in main memory for fast access
5. **disk space management**

An example of this could be a search engine, which is much simpler than a general DBMS - albeit having similar layers;

1. search string modifier
2. ranking engine
3. query execution
4. buffer management
5. disk space management

This is simpler than a DBMS as it can simply use the OS for the bottom two layers, there is typically no concurrency (nor any need for transactions, being mostly read-only), and typically has hard-wired queries. The ranking engine and query execution is a simple DBMS.

DBMS versus Using OS

An important question is why we don't simply use the OS. The layers of abstractions can be useful, but we have a lot of knowledge on how to access the data. In addition, the OS can often get in the way of the DBMS - it has some idea on what to query and what files to touch, and knows more about the OS regarding future access, which can be exploited. A DBMS needs to do things its own way, for example specialised pre-fetching with the knowledge of future access. Additionally, if we control the buffer management, we can also control the replacement policy, likely with something better than the OS. With more control over the thread / process scheduling, the DBMS can achieve a more optimal execution of the workflow as the DB locks aren't going to conflict with the OS locks (high contention). There's also control over flushing data to the disk, including writing log file (important for recovery), and shouldn't be left to the OS.

Disks and Files

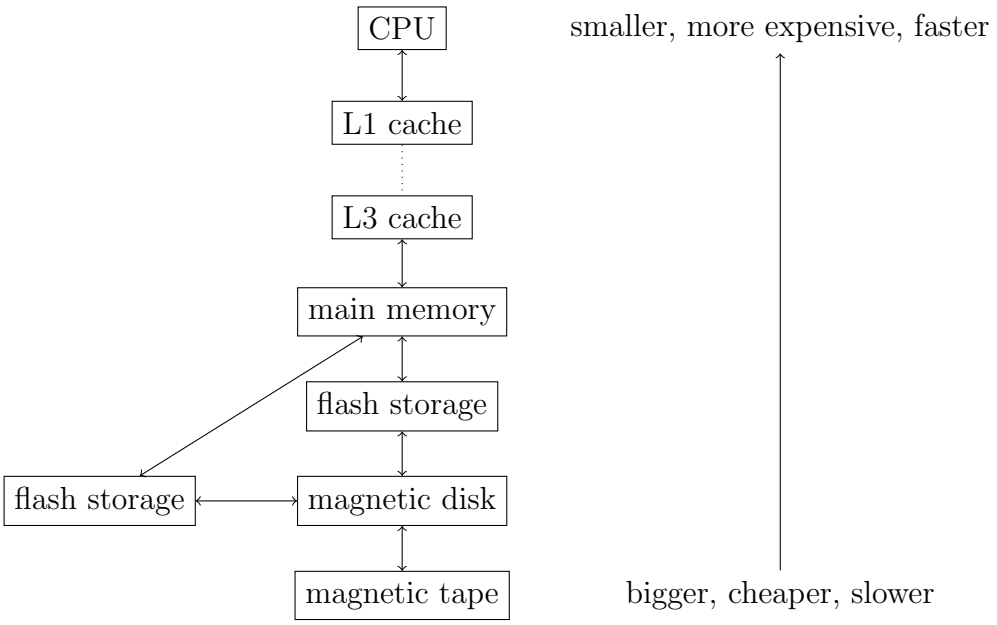
Today, disks are still the go-to storage medium for large amounts of files, and have become fairly affordable (not as affordable as tape for archival storage, but much cheaper than other media, such as SSD or main memory). Unlike other media, disks have mechanical parts leading to differences access patterns or behaviour - the time to access a piece of data is affected by **where** the data is on the disk. A lot of databases today are still on disks, as it's cheap with a reasonable access time - typically the

data is read from the disk onto main memory (buffered), and changes are written back to the disk from the buffer.

High-end databases are in the petabyte range, therefore using main memory would be extremely expensive. In addition to that, main memory is volatile - meaning the data is lost if power goes out. However, main-memory databases do exist for smaller sized, performance optimised systems, in which volatility is tolerable.

Flash storage isn't commonly used in DBMS as main storage (due to the cost), but can also be used as an accelerator or enabler, in the form of a specialised cache, etc.

The storage hierarchy is as follows. Note that flash storage can either be used to the side of magnetic disk (for a specific portion of the files), or as a buffer on top (before main memory). *Jim Gray* has an analogy for how far away the data is (registers being knowledge in your head, etc), illustrating the scale of access time.



Disks are still the main secondary storage device of choice, with the main advantage over tapes being the ability to perform random access, rather than purely sequential. Data is stored in units in **disk blocks** or **pages** (may be used interchangeably), typically in the order of kilobytes and occasionally megabytes. Unlike RAM, the time to retrieve / write a disk page varies depending on the location, with a tremendous impact on time to retrieve, hence relative placement of pages on a disk has a major impact on performance. The lecture then goes over the anatomy of a disk;

- spindle and platters spin between 5,000 - 15,000 RPM (with physical limitations)
- arm assembly has multiple disk head, one for each platter - the tracks under the heads form a cylinder
- only one head reads or writes at once

The time to access a disk block has three elements;

- **seek time** time to move the arm to the correct position (in and out, towards the spindle)
- **rotational delay** arm assembly in a fixed position, block needs to be rotated under head
- **transfer time** actual time to read data from disk into main memory

The time for a smaller number of cylinders traversed is dominated by the seek time (head positioning). In modern disks, short seeks are dominated by the **settle time**, which gets larger with increased disk track density.

Generally, the seek time and rotational delay dominates, with the former being in the range of 1 to 20ms, and the latter being in the range of 0 to 10ms. On the other hand, the transfer rate is less than

1 millisecond for a 4KB page. As such, the key to lower I/O cost is to reduce the delays caused by seek and rotation. Additionally, in shared disks, most of the time is spent waiting for access to the arm.

The concept of the next block is as follows;

- blocks on the same track, followed by
- blocks on the same cylinder
- blocks on adjacent cylinder

We can't control where we write on the disk - that's controlled by the device driver. Typically, if data is written together, it will also be read together. Defragmentation is disk optimization, as data can be spread out all over a disk for a given file, leading to slower read times - this is done by putting all the pieces of a file closer together.

Note that an adjacent block doesn't necessarily mean physically adjacent. Data can be physically spread out over a disk but in a certain pattern that can lead to near sequential access. The adjacent blocks can be the blocks under the disk head after rotating during settle time.

In general, memory access is much faster than disk I/O (roughly $1,000\times$), and sequential I/O is faster than random (roughly $10\times$).

The lowest layer of DBMS manages the space on the disk and higher levels can call this layer to allocate / de-allocate a page, or read / write a page.

Summary

In general, the key for storing data on a disk is to store data together if it's queried together. Random access should be avoided, preferably use sequential access. Data structures should be aligned for page size - for example, if a data structure were to have a few bytes in the next page, an additional page would have to be retrieved, despite mostly being irrelevant.