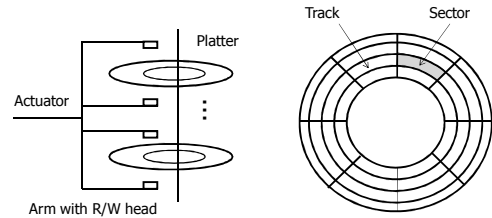


## CS422 Principles of Database Systems Disk Access

Chengyu Sun  
California State University, Los Angeles

## Disk Drive ...



<http://www.youtube.com/watch?v=PCipea9xEXE>

## ... Disk Drive ...

- ◆ Each disk drive contains a number of rotating platters
- ◆ Each platter has a number of tracks on which data is recorded
- ◆ Each track is divided into equal-sized (in bytes) sectors

## ... Disk Drive

- ◆ The tracks with the same track number on different platters form a cylinder
- ◆ Data can be accessed through read/write heads
- ◆ Read/write heads can move from one track to another controlled by an actuator

## Access Data on Disk

1. Move the read/write head to the requested track
2. Rotate the platter so the first requested byte is beneath the r/w head
3. Continue to rotate the platter until all the requested data is transferred

## Disk Access Time

- ◆ Seek time
- ◆ Rotational delay
- ◆ Transfer time

$$\text{Transfer Rate} = \frac{\text{Number of Bytes per Track}}{\text{Time for One Revolution of Platter}}$$

## Measures of Disk Drive Performance

- ◆ Capacity
- ◆ Average seek time
- ◆ Rotation speed
- ◆ Transfer rate

## Seagate ST3500410AS

- ◆ Capacity: 500G
- ◆ Bytes per sector: 512
- ◆ *Default* sectors per track: 63
- ◆ Average seek time (read): <8.5ms
- ◆ Average seek time (write): <9.5ms
- ◆ RPM: 7200rpm

## Examples: Disk Access Time

- ◆ Use the specs of ST3500410AS to calculate the time for the following disk accesses
  - Read 1KB on one track
  - Read 4KB on one track
  - Read 4KB on four tracks

## What We Learned from the Examples

- ◆ Reading more only costs little
- ◆ *Sequential access* is much more efficient than *random access*

## Improve Disk Performance

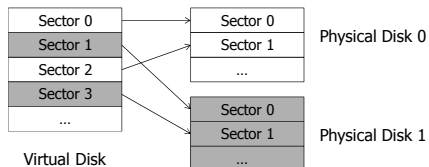
- ◆ Caching
- ◆ Striping
- ◆ Mirroring
- ◆ Storing parity

## Caching

- ◆ Read more data than requested
  - Read one sector vs. read one track
- ◆ Transfer data from cache
  - No seek time
  - No rotational delay
  - Transfer rate 3Gb/s (SATA II)

## Striping

- ◆ Multiple small disks are faster than one large disk ...
- ◆ ... but only when the I/O requests are evenly distributed among the disks



## Example: Striping

- ◆ Suppose we using  $N$  disks for striping, and each disk has  $k$  sectors. An access to virtual sector  $x$  is mapped to an access to which sector on which disk??

## Mirroring

- ◆ Store the same data on two or more disks
- ◆ Improve reliability
- ◆ Do *not* improve speed
  - Same read speed
  - Slower write (*why??*)

## Storing Parity ...

- ◆ Let  $S$  be a set of bits. The parity of  $S$  is
  - 1 if  $S$  contains odd number of 1's
  - 0 if  $S$  contains even number of 1's

Disk 1	1	0	0	1	1	0
Disk 2	1	1	0	0	1	0
Disk 3	1	0	0	1	0	1
Parity Disk	1	1	0	0	??	??

## ... Storing Parity

- ◆ Backup any number of disks with one disk
- ◆ Can only recovery from single disk failure

## Storing Parity without a Parity Disk

Disk 1	Disk 2	Disk 3	Disk 4
1	1	0	0
0	1	0	1
1	1	1	1
0	0	1	1
0	0	0	??
1	0	??	0
0	??	1	1
??	1	0	1

What's the benefit of distributing parity to all disks??

## RAID ...

- ◆ Redundant Array of Inexpensive Drives
- ◆ RAID 0 – striping
- ◆ RAID 1 – mirroring
- ◆ RAID 1+0 – mirroring + striping
- ◆ RAID 2 – striping (bit)
- ◆ RAID 3 – striping (byte) + parity
- ◆ RAID 4 – striping + parity

## ... RAID

- ◆ RAID 5 – striping + parity (no separate parity disk)
- ◆ RAID 6 – striping + 2\*parity (no separate parity disk)

## OS Disk Access API

- ◆ Block-level API
- ◆ File-level API

## Block and Page ...

- ◆ A block is similar to a sector except that the size is determined by the OS
  - E.g. NTFS default block size on Vista is 4KB
- ◆ A file always starts at the beginning of a block
  - *Tradeoff between large and small block sizes??*

## ... Block and Page

- ◆ A page is a block-sized area of main memory
- ◆ Each block/page is uniquely numbered by the OS

## OS Block-Level API

- ◆ `read_block(n, p)` – read block  $n$  into page  $p$
- ◆ `write_block(n, p)` – write page  $p$  to block  $n$
- ◆ `allocate(n, k)` – allocate  $k$  continuous blocks; the new blocks should be as close to block  $n$  as possible
- ◆ `deallocate(n, k)` – mark  $k$  continuous blocks starting at block  $n$  as unused

## OS File-Level API

- ◆ Similar to the API of `RandomAccessFile` in Java
  - <http://java.sun.com/javase/6/docs/api/java/io/RandomAccessFile.html>
  - Treat a file as a continuous sequence of bytes
  - `seek(long position)`
  - Read and write various data types

## DBMS Disk Access API ...

- ◆ Approach 1: use OS block-level API
  - Full control of disk access
    - ◆ Most efficient
    - ◆ Not constrained by file system limitations (e.g. file size)
  - Complex to implement
  - Disks must be mounted as *raw disk*
  - Difficult to administrate

## ... DBMS Disk Access API ...

- ◆ Approach 2: use OS file-level API
  - Easy to implement
  - Easy to administrate
  - No block I/O
    - ◆ Much less efficient
    - ◆ No paging, which is required by DBMS buffer management

## ... DBMS Disk Access API

- ◆ Approach 3: build a block I/O API on top of OS's file I/O API
  - The approach taken by most DBMS

## Readings

- ◆ Chapter 12 of the textbook
- ◆ SimpleDB disk access code in the package `simpledb.file`
- ◆ The SSD Anthology - <http://www.anandtech.com/show/2738>