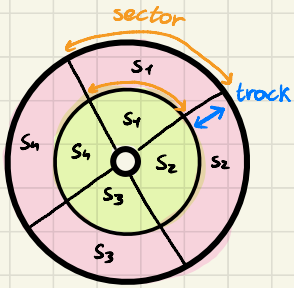
DISK MANAGEMENT



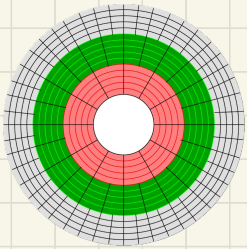
partitioning:

tracks -> concentric circles
sectors -> fixed number of bytes, unit of read/write op.

classical layout:

- fixed number of sectors per track
- higher density towards the center
- low density on the outside
- fixed r/w speed

zone bit recording



- disks divided in zones, within zone fixed number of sectors on the track
- num. of sectors decreases as zones go towards the inside
- up to zx sectors in outer zone
- almost double r/w speed on outside

Access times

T_a for b bytes in seconds

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

N -> bytes/track r: spinning speed
T_s: positioning of the head (seek time)

The positioning of the head is only sufficient for random readings
for multiple reads order matters

Seek times

4 phases: speed-up, coast, slowdown, settle

disk controller determines power of motor driving arm (power values in a table, must be recalibrated)

Skewing

want to keep read speed high

shift sectors on consecutive tracks -> reduce rotational delay

Sparing

Map faulty (by factory) sectors to other ones with a list passed to the controller

Disk controller

connected to the bus

recieves req. through device's driver software and handles them

can buffer at least one sector (since bus is faster)

command queueing -> can recieve multiple commands and decide the order of processing

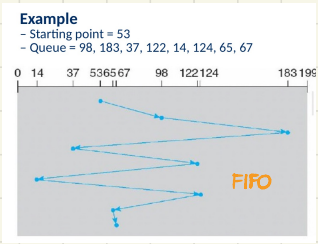
disk caching

read-ahead strategy

when I read smth I replace it immediately

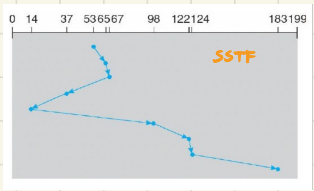
aggressive read-ahead -> over multiple tracks

DISK SCHEDULING ALGORITHMS



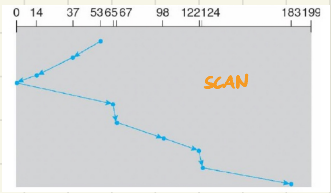
FIFO

- fair
- efficient for clustered operations
- inefficient for random reads (there might be a more efficient order)



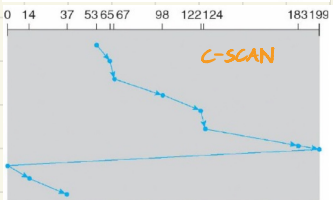
shortest seek time first (SSTF)

- requires estimation of seek time (non linear)
- high efficiency (locality of reference)
- favores too much near requests (starvation for others)



SCAN

- SSTF with arm changing direction only when reaching extremes of the disk
- less starvation than SSTF
- lower delay variance, higher average
- unfair for center



cyclic SCAN

- SCAN but when extreme reached, continue from other extreme
- more fair

LOOK

- SCAN but arm changes direction if no more requests in current direction
- slightly more efficient than SCAN
- more complex
- unfair for center (solved with C-look)

V-SCAN

introduces parameter $R \in [0,1]$

use distance = SSTF distance + 1 (if changing direction) · R · width of disk (in Ts)

VSCAN(0) = SSTF

VSCAN(1) = SCAN

other values is balanced

still preferes clustered requests in same direction

F-SCAN

use a two stage buffer

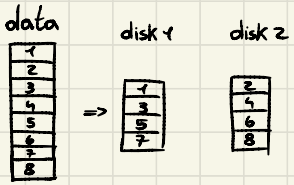
SCAN on first buffer, then process the ones in second buffer

RAID

many inexpensive disks form one large, fast, reliable disk

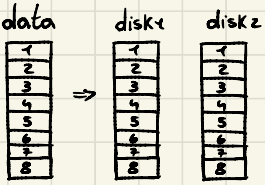
Raid 0

- data striping over N disks
- stripe lenght
  - large: random reads simultaneous
  - small: sequential read fast (seek & rot. a bit worse)
- poor robustness



Raid 1

- data mirroring over 2 disks
- very high robustness
- expensive
- can load balance for random reads
- write is slower than with one disk



Raid 3 and 4

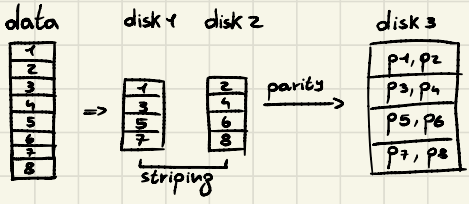
Raid 0 over N disks +1 extra parity bit (sum is even)

Raid 3:

- synchronized disk heads
- short stripe
  - fast sequential reads
  - parity on write, mostly no read required

Raid 4:

- long stripe
  - random read fast
  - 1 W = 2R + 2R

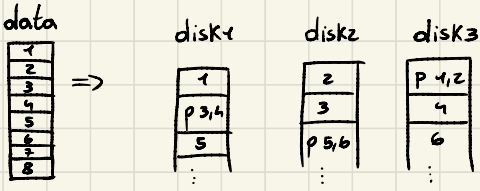


Raid 5 and 6

Raid 4 with parity info spread across all disks

- better random read performance
- random write still expensive

Raid 6: one more disk



Raid configs can be combined

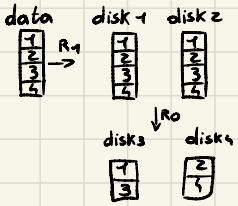
Raid x+y

organize data according to y, subdived each of the obtained disks with x

notation can be messy

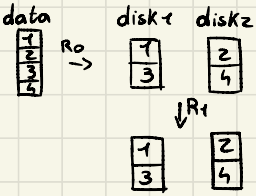
Raid 0+1

first mirror, then stripe



Raid 1+0

stripe, then duplicate



more robust, can handle many disk failures