Storage Devices: HDD, SDD, NVM & RAID

Azza Abouzied

Storage Devices

The Limiting Factor

Why are disks not fast enough?

1. At 7200 RPM, a full rotation is 8ms and **expected rotation time** is 4ms.

2. The **seek cost** is 4-10 ms.

3. Our **transfer bandwidth** is 40-125MB/s

Transfer 1KB

Seek + Rotational Delay + Transfer

4ms + 4ms + 1KB/125MB/s = 8ms + 0.007ms = 8.007ms

Our effective transfer rate is 1KB/8.007ms = 125KB/s = 1/1000 of 125MB/s!

How can we maximize performance?

Can we get an effective transfer rate that is 9/10 of disk bandwidth (bw) instead of 1/1000?

Amortization!

$$bw \times \frac{9}{10} = \frac{size}{size/bw + seek + rotation}$$

$$size = 9 * bw \times (seek + rotation)$$

$$size = 9 * 125MB/s \times (4ms + 4ms) = 9MB$$

Disk Scheduling for Amortization

FIFO

Assume you have the following track/cylinder requests:
98, 183, 37, 122, 14, 124, 65, 67

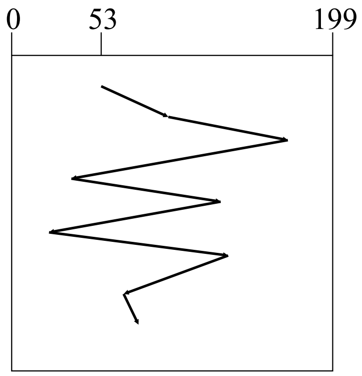
Pros:

1. Fairness: Blocks arrive in the order requested

Cons:

1. Long seeks
2. Wild swings

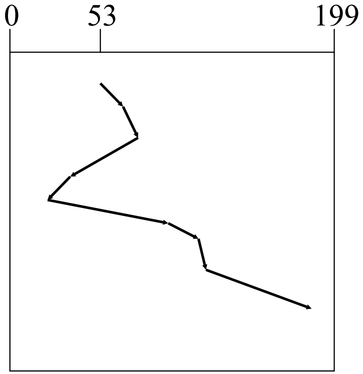
How many tracks are visited?
640 tracks!



98, 183, 37, 122, 14, 124, 65, 67

Shortest Seek Time First (SSTF)

Pick track closest on disk to the current head position.



98, 183, 37, 122, 14, 124, 65, 67
(65, 67, 37, 14, 98, 122, 124, 183)

Pros:

1. Minimize seek time

Cons:

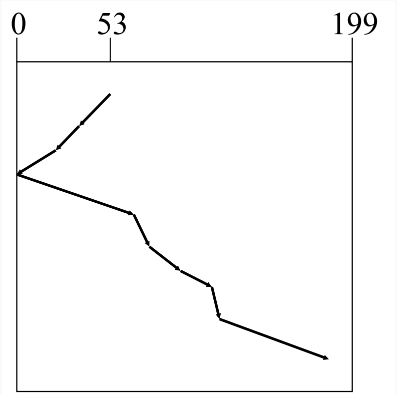
1. Starvation
2. Ignore rotation**

How many tracks are visited?

236 tracks

Elevator

Pick the closest in the direction of head (So no back and forth head movement)



98, 183, 37, 122, 14, 124, 65, 67
(37, 14, 65, 67, 98, 122, 124, 183)

Pros:

1. No Starvation

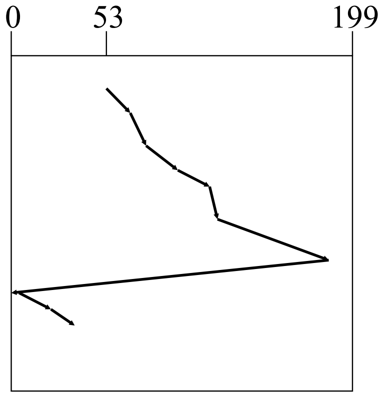
Cons:

1. Can still do better!

How many tracks are visited?
230 tracks

Elevator

Like elevator, except once it reaches the end it jumps to the other end. Always moves in one direction.



98, 183, 37, 122, 14, 124, 65, 67
(65, 67, 98, 122, 124, 183, 14, 37)

Pros:

1. Uniform Service time

How many tracks are visited? 187 (a jump is not counted as a scan!) An optimization where you jump to the furthest track request instead of track 0 lowers the cost to 157 tracks.

The Linux IO Schedulers

The Linus Elevator - Linux 2.4

- maintains on IO queue for all requests roughly sorted by sector
- merges new requests with existing nearby requests in the queue
- or inserts request within the queue based on sector location
- if a request has aged, then new requests regardless of sector location are placed at the tail

How does this affect latency? Can it cause starvation?
Writes Starve Reads Why is writes-starve-reads so problematic?

The Linux IO Schedulers

The Deadline IO Scheduler

1. Read latency is more important because reads are synchronous — the process blocks — while writes can occur asynchronously.

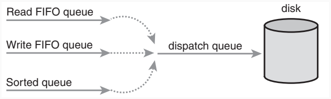
2. Reads are dependent on the previous read

3. Whether you read or write, you still need to read the metadata related to a file.

4. If each read request is individually starved, the total delay compounds and can become enormous.

5. Associate deadlines with reads (500 ms), writes (5 seconds).

6. Each request goes to two queues: sorted and read/write.



The Linux IO Schedulers

Anticipatory IO Scheduler

1. Disk throughput goes down if you keep shuttling from read requests to an ongoing write.
2. After servicing read requests, **wait and do nothing**!
3. If nothing comes in, then process the write, else you might get a nearby read request so service it.

Complete Fair Queuing IO Scheduling

1. A queue per process.
2. Service queues round-robin and service *x* requests from each queue at a time.

Noop Scheduler does nothing! Works for random access devices like SSDs.

SSD

SSD drives

NAND Flash as high capacity as HDD

Block Structure

1. Each block has 32 or 64 Pages
2. Each Page is 512+16 or 2048+64 Bytes

Speed *(Tends to be asymmetrical)*

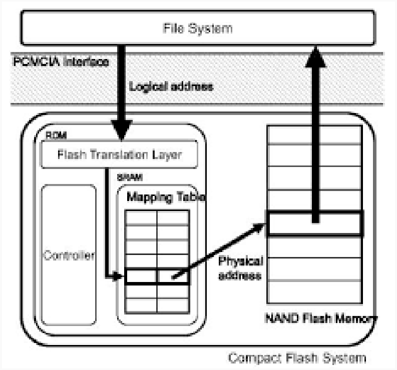
1. Read Page: 10μs
2. Write Page: 20-200μs
3. Erase **Block**: 1-2ms

Erase sets the whole block to 1. We actually write 0s.

How do we write to flash memory?

The Flash Translation Layer: Many problems can be solved by adding a layer of indirection. The FTL makes flash access similar to disk access.

1. Map virtual page address to physical page address
2. Write performance impacted by the availability of free/erased blocks to write to.
3. Erases unused blocks
4. Performs wear-leveling.



Why do we need wear-leveling on flash?

The life span of a page depends on the type of flash.

1. $\approx$ 50-100 thousand writes/page for **SLC.** It is faster, more reliable, and less dense but at least twice as expensive than MLC

2. $\approx$ 15-60 thousand writes/page for the higher density, slower **MLC.**

With desktop applications, if you do some basic computations, it would take about hundreds of years to wear out your flash drive completely (the drive will naturally die within 10 years). With enterprise applications, you can wear out a drive in as little as 6 months!

The future of storage devices

NVM

Non-volatile memory is becoming more and more likely with potential impact on how we design file systems and operating systems in general.

Despite advances, we still don’t have universal memory yet, i.e. storage that performs as fast as caches or main memory, but we are pushing latency numbers down.

What is the key impact of faster storage?

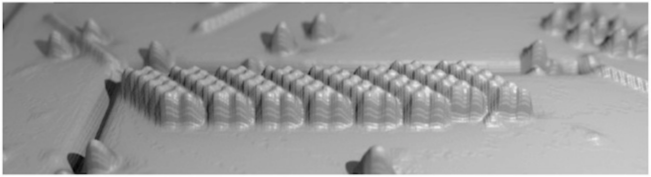
The new latency numbers

Memory	Latency	Bandwidth	Capacity / IOPS
Register	0.25ns		
L1-L3 cache	1-11 ns		L3: 45 MB
DRAM	62ns	102GBps	6TB
NVRAM DIMM	620ns	60GBps	24TB
1-sided RDMA	1.4μs	100GbE	$\approx$ 700K IOPS
RPC	2.4μs	100GbE	$\approx$ 400K IOPS
NVRAM NVMe	12μs	6GBps	16TB/disk
SSD NVMe	90μs	5GBps	16TB/disk,
SAS/SATA SSD	110μs	1.5GBps	16TB/disk
SAS/SATA HDD	10ms	1.5GBps	10TB/disk

From Adrian Coyler's blog: the morning paper

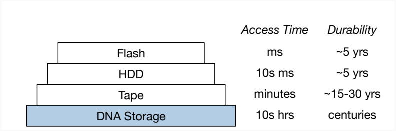
Storage is a hot research topic

How many atoms do you need to store a bit?

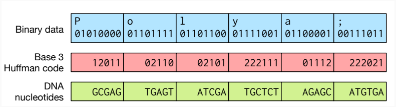


12 atoms – IBM’s nanotechnology storage device. Need a machine the size of a washing machine to write a bit. The BBC article, 2012

Using DNA for archival storage?



	Access Time	*Durability*
Flash	ms	~5 yrs
HDD	10s ms	~5 yrs
Tape	minutes	~15-30 yrs
DNA Storage	10s hrs	centuries



Binary data	P 01010000	o 01101111	l 01101100	y 01111001	a 01100001	; 00111011
Base 3 Huffman code	12011	02110	02101	222111	01112	222021
DNA nucleotides	GCGAG	TGAGT	ATCGA	TGCTCT	AGAGC	ATGTGA

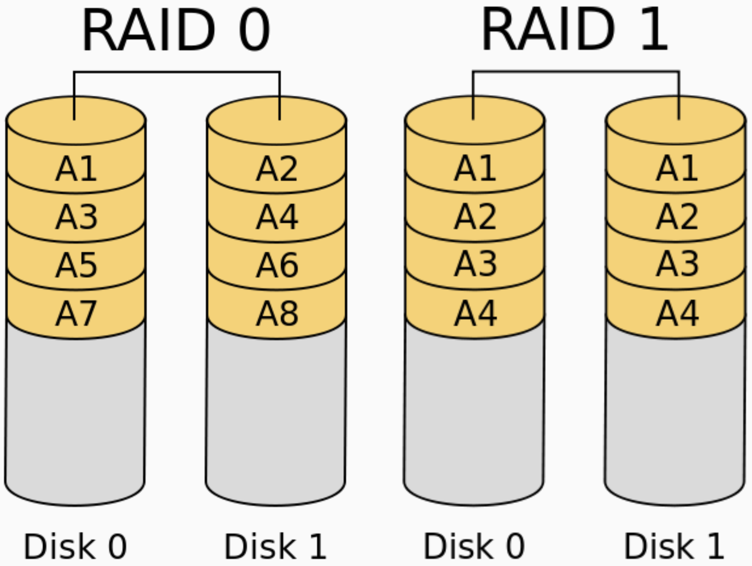
Published in ASPLOS’16

RAID

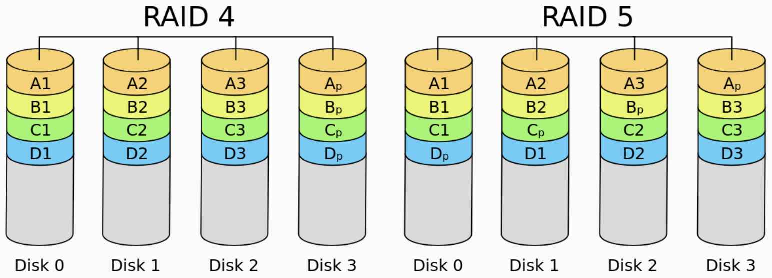
RAID is neither of these

1 RUTHLESS CRIME LORD. 20 ELITE COPS. 30 FLOORS OF CHAOS.
"THE BEST ACTION MOVIE IN DECADES!"
TWITCHFILM
OFFICIAL SELECTION SUNDANCE FILM FESTIVAL
WINNER TORONTO FILM FESTIVAL
OFFICIAL SELECTION SXSW FILM FESTIVAL
OFFICIAL SELECTION MOMA ND/NF FILM FESTIVAL
THE RAID
REDEMPTION
WWW.THERAIDMOVIE.COM
WWW.SONYCLASSICS.COM

Redundant Array of Independent Disks



Redundant Array of Independent Disks



Parity computation is

$$\mathrm{P} = \bigoplus_i D_i = \mathrm{D}_0 \oplus \mathrm{D}_1 \oplus \mathrm{D}_2 \oplus \ldots \oplus \mathrm{D}_{n-1}$$

Raid-4 has a dedicated parity disk, while Raid-5 stripes the parity disk across devices. **What benefit do we get from this striping?**

Performance and Reliability

Consider RAID 0, 1 and 5 and 8 equivalent disks

1. How much usable storage does the system receive?
2. If we only do reads without verification. What is the expected throughput if each disk does 100 reads/second?
3. If we only do writes. What is the throughput now?
4. What is the min number of disk that may fail before data is lost?
5. What is the minimum number of disks that must fail to guarantee data loss?

Questions?