PERSISTENCE: DISK SCHEDULING

Shivaram Venkataraman
CS 537, Spring 2020
Project 4a is out! Due April 2\textsuperscript{th}.

More details in discussion section.

Midterm grading in progress.
AGENDA / LEARNING OUTCOMES

How do you calculate sequential and random tput of a disk?

What algorithms are used to schedule I/O requests?
RECAP
EXAMPLE WRITE PROTOCOL

<table>
<thead>
<tr>
<th>Status</th>
<th>COMMAND</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller (CPU+RAM)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extra RAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other special-purpose chips</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

while (STATUS == BUSY)
    ; // spin
Write data to DATA register
Write command to COMMAND register
while (STATUS == BUSY)
    ; // spin

Virtualization
Concurrency
Persistente

General/Hidden

Example
xv6 → IDE/EIDE drivers

Device drivers
Software/code
flat interacts with the device

Memory → Device

In/Out
MMIO

Interrupts
DMA

Software code
Motor connected to spindle spins platters

Rate of rotation: RPM

10000 RPM $\rightarrow$ single rotation is 6 ms
Tracks are divided into numbered sectors → **512 bytes**

Analog: PAGES

24 sectors in this surface

Read / write to a sectors
Heads on a moving arm can read from each surface.
READING DATA FROM DISK

Rotational delay

→ Wait for disk to rotate
→ Once sector is under the Head, do the read/write
READ DATA FROM DISK

Rotates this way

Remaining rotation

Spindle

Seek Time

Time it takes for the arm to move to the right track

Seek

Want to read
TIME TO READ/WRITE

Three components:

Time = seek + rotation + transfer time

- to get to right track
- right sector inside track
- you want to perform the operation
SEEK, ROTATE, TRANSFER

Seek cost: Function of cylinder distance
Not purely linear cost
Must accelerate, coast, decelerate, settle
Settling alone can take 0.5 - 2 ms
Entire seeks often take 4 - 10 ms
Average seek = 1/3 of max seek

Derivation in text book

Depends on rotations per minute (RPM)
7200 RPM is common, 15000 RPM is high end
Average rotation?
Half of time for 1 rotation

Pretty fast: depends on RPM and sector density.
100+ MB/s is typical for maximum transfer rate

USB 2.0 slower
What is the time for 4KB random read with Cheetah?

\[ T_{\text{seek}} + T_{\text{rotation}} + T_{\text{transfer}} \]
\[ = 4 \text{ ms} + 2 \text{ ms} + 0.03 \text{ ms} = 6 \text{ ms} \]

- 15000 rotations in 60,000 ms
  - 1 rotation in 4 ms
  - Avg = \( \frac{4}{2} \) = 2 ms

<table>
<thead>
<tr>
<th></th>
<th>Cheetah 15K.5</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>300 GB</td>
<td>1 TB</td>
</tr>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Average Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
<tr>
<td>Platters</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cache</td>
<td>16 MB</td>
<td>16/32 MB</td>
</tr>
<tr>
<td>Connects via</td>
<td>SCSI</td>
<td>SATA</td>
</tr>
</tbody>
</table>

\[ T_{\text{transfer}} = \frac{4 \text{ KB}}{125 \text{ MB/s}} = 0.032 \text{ ms} \]
QUIZ 21

What is the time for 4KB random read with Barracuda?

\[ \text{Time} = \text{Seek} + \text{Trot} + \frac{\text{Transfer}}{\text{Max Transfer}} \]

- Seek: 9 ms
- Trot: \(4.16 \text{ ms} \times 0.038\) = \(0.16 \text{ ms}\)
- Transfer: \(\frac{4 \text{ KB}}{105 \text{ MB/s}} = 0.038 \text{ ms}\)

\[ \text{Total Time} = 9 \text{ ms} + 0.16 \text{ ms} + 0.038 \text{ ms} = 9.228 \text{ ms} \]

\[ \approx 9.23 \text{ ms} \]

Barracuda:

- Capacity: 1 TB
- RPM: 7,200
- Average Seek: 9 ms
- Max Transfer: 105 MB/s
- Connects via SATA

Cheetah 15K.5:

- Capacity: 300 GB
- RPM: 15,000
- Average Seek: 4 ms
- Max Transfer: 125 MB/s
- Connects via SCSI
WORKLOAD PERFORMANCE
WORKLOAD PERFORMANCE

So…
- seeks are slow $\rightarrow \sim 4 - 9 \text{ ms}$
- rotations are slow $\rightarrow \sim 2 - 4 \text{ ms}$
- transfers are fast $\rightarrow 0.03 \text{ ms}$

How does the kind of workload affect performance?
Sequential: access sectors in order $\rightarrow$ Better
Random: access sectors arbitrarily $\rightarrow$ Worse

Avoid seeks and rotations
<table>
<thead>
<tr>
<th></th>
<th>Cheetah</th>
<th>Barracuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>300 GB</td>
<td>1 TB</td>
</tr>
<tr>
<td>RPM</td>
<td>15,000</td>
<td>7,200</td>
</tr>
<tr>
<td>Avg Seek</td>
<td>4 ms</td>
<td>9 ms</td>
</tr>
<tr>
<td>Max Transfer</td>
<td>125 MB/s</td>
<td>105 MB/s</td>
</tr>
<tr>
<td>Platters</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cache</td>
<td>16 MB</td>
<td>32 MB</td>
</tr>
</tbody>
</table>

**Sequential workload:** what is throughput for each?

- Cheetah:
  - Time to seek and rotate: $\frac{100 \text{ MB}}{125 \text{ MB/s}} \approx 0.78 \text{ ms}$

- Barracuda:
  - Time to seek and rotate: $\frac{6 \text{ ms}}{105 \text{ MB/s}} \approx 0.057 \text{ ms}$

OTHER IMPROVEMENTS

Track Skew

Zones

Cache
Imagine sequential reading, how should sectors numbers be laid out on disk?

Rotation is always happening and is simultaneous with arm movement.
When reading 16 after 15, the head won’t settle quick enough, so we need to do a rotation.
Track Skew

Account for rotation & seek happening at the same time

Random = seek + sequential read of 4KB
ZBR (Zoned bit recording): More sectors on outer tracks
DRIVE CACHE

Drives may cache both reads and writes. (In addition to OS cache)

What advantage does caching in drive have for reads?

Avoid doing disk seek if sector is in cache

What advantage does caching in drive have for writes?

Acknowledge writes before it is "persisted"
BUFFERING

Disks contain internal memory (2MB-16MB) used as cache

Read-ahead: “Track buffer”
  – Read contents of entire track into memory during rotational delay

Write caching with volatile memory
  – Immediate reporting: Claim written to disk when not
  – Data could be lost on power failure

Tagged command queueing
  – Have multiple outstanding requests to the disk
  – Disk can reorder (schedule) requests for better performance
I/O Schedulers
I/O SCHEDULERS

Given a stream of I/O requests, in what order should they be served?

Much different than CPU scheduling

Position of disk head relative to request position matters more than length of job

```
Read 1000 Write 2000 Read 1001 ...
```

Sector number

Time
FCFS (First-Come-First-Serve) → FIFO

Assume seek+rotate = 10 ms for random request

How long (roughly) does the below workload take? Requests are given in sector numbers

300001, 700001, 300002, 700002, 300003, 700003
10ms 10ms 10ms
≈ 60 ms

300001, 300002, 300003, 700001, 700002, 700003
≈ 20 ms

Reordering requests can improve performance.
SSTF (SHORTEST SEEK TIME FIRST)

**Strategy** always choose request that requires least seek time (approximate total time with seek time)

Greedy algorithm (just looks for best NEXT decision)

How to implement in OS?
- Sort sector numbers

Disadvantages?
- Starvation

How much seek time will it take?

Sort by sector number
SCAN or Elevator Algorithm:
- Sweep back and forth, from one end of disk other, serving requests as pass that cylinder
- Sorts by cylinder number; ignores rotation delays

C-SCAN (circular scan): Only sweep in one direction

Pros/Cons?
- Scan the internal tracks get to go twice \( \rightarrow \) NOT fair
- CSCAN \( \rightarrow \) 100 \( \rightarrow \) No reads
SPTF (Shortest Positioning Time First)

In the example 8 may have lower access time! 

SATF (Shortest Access Time First)
Disk accesses: 32, 12, 33, 3, 13, 4
Rotation Time = 2ms (non-adjacent reads)
Seek Time (for adjacent track) = 2ms.

What is the time taken to using (FCFS) scheduling?

32: 2ms
12: 2ms for seek + 2ms for rot = 4
33: 4ms
3: 4ms + 2 = 6 → going to two tracks
13: 2ms + 2ms = 4
4: 2ms + 2ms = 4

Order in which requests will be serviced for Shortest Seek Time First (SSTF)?

= 24ms
Disk accesses: 32, 12, 33, 3, 13, 4
Rotation Time = 2ms (non-adjacent reads)
Seek Time (for adjacent track) = 2ms.

Order in which requests will be serviced for
Shortest Seek Time First (SSTF)?

Time Taken
Where should the scheduler go?
WHAT HAPPENS?

Assume 2 processes each calling read() with C-SCAN

```c
void reader(int fd) {
    char buf[1024];
    int rv;
    while((rv = read(fd, buf)) != 0) {
        assert(rv);
        // takes short time, e.g., 1ms
        process(buf, rv);
    }
}
```
WORK CONSERVATION

Work conserving schedulers always try to do work if there’s work to be done.

Sometimes, it’s better to wait instead if system anticipates another request will arrive.

Possible improvements from I/O Merging.
SUMMARY

Disks: Specific geometry with platters, spindle, tracks, sector

I/O Time: rotation_time + seek_time + transfer_time
Sequential throughput vs. random throughput

Advanced Techniques: Skewed layout, caching

Scheduling approaches: SSTF, SCAN, C-SCAN
Benefits of violating work conservation
NEXT STEPS

Next class: How to achieve resilience against disk errors

Project 4a in Discussion today