PERSISTENCE: DISK SCHEDULING

Shivaram Venkataraman
CS 537, Spring 2019
Grades: Project 2b, 3, midterm grades out!
See Piazza for regrade information

Project 4a is out! Due April 4th
More details in discussion section

Out of town Monday, Tue next week.
Guest lecture on Tuesday
How do you calculate sequential and random tput of a disk?

What algorithms are used to schedule I/O requests?
RECAP
Hardware support for I/O

CPU
Memory

Memory Bus (proprietary)

General I/O Bus (e.g., PCI)
Graphics

Peripheral I/O Bus (e.g., SCSI, SATA, USB)

→ High speed
Lower speed
Greater distance
3.5 inch hard drive
SATA, keyboard
### Example Write Protocol

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

<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>

**Registers**
- Status
- COMMAND
- DATA

- Waiting
- Writing
- Copying
- Check if device is busy
- Wait for the operation
Protocol Variants

Status checks: polling vs. interrupts

PIO vs DMA

Special instructions vs. Memory mapped I/O

<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>

CPU stalls avoided with "DMA"
HARD DISK INTERFACE

Disk has a sector-addressable address space
Appears as an array of sectors

Sectors are typically 512 bytes

Main operations: reads + writes to sectors

Mechanical and slow (?)
Motor connected to spindle spins platters.

Rate of rotation: RPM

10000 RPM → single rotation is 6 ms
Heads on a moving arm can read from each surface.
READING DATA FROM DISK

Seek Time
Rotational delay

1. Th disk
2. Num surface
3. Sector size = 512 bytes
4. Head right track
5. Surface rotate right sector is below the Head
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 takes 4 - 10 ms
Average seek = 1/3 of max seek

Depends on rotations per minute (RPM)
7200 RPM is common, 15000 RPM is high end
Average rotation?
Pretty fast: depends on RPM and sector density.

100+ MB/s is typical for maximum transfer rate
What is the time for 4KB random read?

<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>
How long does an average random 4-KB read take with Cheetah?

\[ T_{\text{total}} = T_{\text{seek}} + T_{\text{rotate}} + T_{\text{transfer}} \]

\[ T_{\text{seek}} = 4 \text{ ms} \]

\[ T_{\text{rotate}} = \frac{15000 \text{ RPM}}{4 \times 60} \times 4 \text{ ms} = 6.25 \text{ ms} \]

\[ T_{\text{transfer}} = \frac{4 \text{ KB}}{125 \text{ MB/s}} = \frac{4 \times 10^3 \text{ B}}{125 \times 10^6 \text{ B/s}} = 3.2 \times 10^{-4} \text{ s} = 0.032 \text{ ms} \]

Rot per ms \( \approx 4 \text{ ms/rot} \)  \( \Rightarrow 2 \text{ ms on avg} \)
How long does an average random 4- KB read take w/ Barracuda?

\[ T_{seek} = 9 \text{ ms} \]

\[ T_{rot} = \frac{7200 \text{ RPM} \times 60 \times 1000}{60 \times 1000} = \frac{72}{60} \text{ ms} = 1.2 \text{ ms} \]

\[ T_{transfer} = \frac{4}{105} = 0.038 \text{ ms} = 132 \text{ ms} \]

<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>

Avg \( \approx 4.16 \text{ ms} \)
WORKLOAD PERFORMANCE
So…
- seeks are slow
- rotations are slow
- transfers are fast

How does the kind of workload affect performance?
Sequential: access sectors in order
Random: access sectors arbitrarily
### DISK SPEC

<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**: $4 \text{ms} + 2 \text{ms} = 6 \text{ms} \implies \frac{4 \text{KB}}{6 \text{ms}} < \ll 125 \text{MB/s}$
- **Barracuda**: $9 \text{ms} \implies \frac{105 \text{MB/s}}{9 \text{ms}} = 11.66 \text{MB/s}$ (effective throughputs are noted.)
OTHER IMPROVEMENTS

Track Skew

Zones

Cache
Imagine sequential reading, how should sectors numbers be laid out on disk?
When reading 16 after 15, the head won’t settle quick enough, so we need to do a rotation.
Track Skew
**ZBR** (Zoned bit recording): More sectors on outer tracks
OTHER IMPROVEMENTS

Track Skew

Zones

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

What advantage does caching in drive have for reads?
- Store recently read sectors. Fetch it from cache. Read ahead.

What advantage does caching in drive have for writes?
- CPU doesn't need to wait for write to finish. Acknowledge before "persistent."
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
FCFS (FIRST-COME-FIRST-SERVE)  
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

300001, 300002, 300003, 700001, 700002, 700003
SSTF (SHORTEST SEEK TIME FIRST)

Strategy: always choose request that requires least seek time (time for seeking and rotating)

Greedy algorithm (just looks for best NEXT decision)

How to implement in OS?

Disadvantages?

Starvation
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?
SPTF (SHORTEST POSITIONING TIME FIRST)

SATF (SHORTEST ACCESS TIME FIRST)
Schedulers

Where should the scheduler go?

OS

Disk

Scheduler

Scheduler

Hardware

Knows process

Fair sharing across

Software & upgrade

Sorts request

Sector location

Seek time

Rotation time

Knows disk geometry
WHAT HAPPENS?

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

```c
void reader(int fd) {
    char buf[1024];
    int rv;
    while((rv = read(buf)) != 0) {
        assert(rv);
        // takes short time, e.g., 1ms
        process(buf, rv);
    }
}
```
Work conserving schedulers always try to do work if there’s work to be done.

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

Possible improvements from I/O Merging:

- Not work conserving
- Wait for a while to “merge” or get a better sequence.
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
Rotational speed is set to 1 degree per time. Complete revolution takes 360 time

Transfer begins and ends at the halfway point between sectors. E.g., to read sector 10, the transfer begins halfway between 9 and 10, ends halfway between 10 and 11.

There are 12 sectors per track, meaning that each sector takes up 30 degrees. To read a sector, it takes 30 time units (given our default speed of rotation).

Disk head is positioned on the outside track, halfway through sector 6.
Compute the seek, rotation, and transfer times for the following sets of requests:
1. -a 7
2. -a 7,30,8
3. -a 10,11,12,13

    python disk.py -a <cmd> -G
Compare FIFO and SSTF for request stream 7,30,8

```
python disk.py -a 7,30,8 -p <SSTF|FIFO>
```
Next class: How to achieve resilience against disk errors

Project 4a in Discussion today

Guest lecture on Tuesday