**Chapter 37: Hard Disk Drives**

Hard disk is the main form of persistent data storage in computer systems.

**37.1 The Interface**

The drive consists of a large number of sectors (512-byte blocks), each of which can be read or written. We can view the disk as an array of sectors from 0 to n-1, which is the **address space** of the drive.

Multi-sector operations are possible. Many file systems will read or write 4KB at a time. When updating the disk, the only guarantee drive manufacturers make is that a single 512-byte write is **atomic**. Thus, if a power loss occurs, only a portion of a larger write may complete (**torn write**).

There are some assumptions most clients of disk drives make, but that are not specified directly in the interface. Researchers called this the **unwritten contract** of disk drives. We usually assume that accessing two blocks near one-another within the drive’s address space will be faster than accessing two blocks that are far apart. We also assume that accessing blocks in continuous chunk is the fastest mode.

**37.2 Basic Geometry**

**Platter** is a circular hard surface on which data is stored persistently by inducing magnetic changes to it. A disk may have one or more platters. Each platter has two sides, called **surface**. Platters are made of hard material such as aluminum and then coated with a thin magnetic layer that enables the drive to persistently store bits even when the drive is powered off.

The platters are bound together around a **spindle**, which connected to a motor that spins the platters around. The **rotations per minute (RPM)** is often from 7200 to 15000.

Data is encoded on the surface in concentric circles of sectors. We call one concentric circle a **track**. A single surface contains thousands of tracks.

To read and write from the surface, we need a mechanism that allows us to either sense the magnetic patterns on the disk or to induce a change. This is accomplished by **disk head**, which is attached to a single **disk arm** that moves across the surface to position the head over the desired track.

Diagram

Description automatically generated

**37.3 A Simple Disk Drive**

Consider the below disk. In the figure above (37.2), the disk head is attached and point to sector 6.

Shape, circle

Description automatically generated

**Single-track Latency: The Rotational Delay**

Consider the track disk that is asked to read block 0, it has to wait for the desired sector to rotate under the disk head. This wait is referred to as **rotational delay**. In the above example, we have to wait R/2 where R is the full rotation delay.

**Multiple Tracks: Seek Time**

**Diagram

Description automatically generated**

Consider when we have multiple tracks. When we are at 30, to move to 11, we have to **seek**, which is the most costly disk operations.

The seek operation has many phases: first an acceleration phase as the disk arm gets moving; then coasting as the arm is moving at full speed, then deceleration as the arm slows down; finally settling as the head is carefully positioned over the correct track. The **settling time** is significant (0.5 to 2ms).

After seek, the platter rotates to find 11.

When sector 11 passes under the disk head, the final phase of I/O will take place, known as the **transfer**, where data is either read from or written to the surface. And thus, we have a complete picture of I/O time: first a seek, then waiting for the rotational delay, and finally the transfer.

**Some Other Details**

Many drives employ **track skew** to make sure that sequential reads can be properly serviced when crossing track boundaries.

Sectors are often skewed like this because when switching from one track to another, the disk needs time to reposition the head. Without such skew, the head would be moved to the next track but the desired next block would have already rotated under the head.

Diagram

Description automatically generated

Another thing is that outer tracks tend to have more sectors than the inner tracks (result of geometry). These tracks are often referred to as **multi-zoned** disk drives where the disk is organized into multiple zones, and where a zone is consecutive set of tracks on a surface. Each zone has the same number of sectors per track, and outer zones have more sectors than inner zones.

In addition, disk drives also have **cache**, sometimes called track buffer. This cache is just some small amount of memory which the drive can use to hold data read from or written to the disk.

On writes, **write back** is when the write is acknowledged after completion when it has put the data in memory. **Write through** is when it has been written to disk.

Write back is sometimes make the drive appear faster, but can be dangerous: if the file system or applications require that data be written to disk in a certain order for correctness, write-back caching can lead to problems.

**37.4 I/O Time: Doing The Math**

TI/O = Tseek + Trotation + Ttransfer

The rate of I/O can be computed using:

RI/O = Sizetransfer / TI/O

Table

Description automatically generated

Assuming we read 4KB at random location on disk. On Cheetah, the time would be:



The rotation time can be calculated since RPM is 15000, so 1 rotation is 2ms. The transfer time is the size of the transfer over the maximum transfer rate.

Text, table

Description automatically generated

There is a huge gap between random and sequential workloads. In addition, there is a large difference in performance between high-end performance drives and low-end capacity drives.

**37.5 Disk Scheduling**

Because of the high cost of I/O, the OS has to decide the order of I/Os issued to the disk. The disk scheduler examines the requests and decides which one to schedule next.

**SSTF: Shortest Seek Time First**

In this approach, it orders the queue such that it picks requests on the nearest track to complete first.

Diagram

Description automatically generated

For example, in this scenario, it would go to sector 21 and then 2. However, the drive geometry is not available to the host OS. Instead, OS can simply implement **nearest-block-first (NBF)** that schedules the request with the nearest block address next.

The second problem is **starvation**. This can happen because there can be a request that is far away from the rest, so it will never be processed.

**Elevator (a.k.a. SCAN or C-SCAN)**

This approach (SCAN) simply moves back and forth across the disk servicing requests in order across the tracks. Let’s call a single pass across the disk a **sweep**. If a request comes for a block on a track that has already been serviced on this sweep of the disk, it is not handled immediately, but rather queued until the next sweep (in the other direction).

F-SCAN, on the other hand, freezes the queue to be serviced when it is doing a sweep. This action places requests that come in during the sweep into a queue to be serviced later. Doing so avoids starvation of far-away requests, by delaying the servicing of late-arriving (but nearer by) requests.

C-SCAN sweeps both direction of the disk, i.e. outer-to-inner and then inner-to-outer.

Problem: does not adhere as closely to the principle of SJF.

**SPTF: Shortest Positioning Time First**

The problem with choosing what goes next is that it depends on the seek time and rotation time.

In modern drives, both seek and rotation are roughly equivalent, so SPTF is useful and improves performance. However, OS does not know the information in the disk, so SPTF is usually performed in a drive.

**Other Scheduling Issues**

In modern system, disks can accommodate multiple outstanding requests, and have sophisticated internal schedulers themselves. Thus, the OS scheduler usually picks what it thinks the best few requests are and issues them all to disk. The disk then uses its internal knowledge of head position and detailed track layout information to service said requests in the best possible (SPTF) order.

Another important related task performed by disk schedulers is **I/O merging**. For example, if the requests are to read block 33, 8 and then 34, then we can merge to read 33 and 34 together.

How long should the system wait before issuing an I/O to disk? One approach is **work-conserving** where the disk should immediately issue the request to the drive.

However, it is better to wait for a bit (**non-work-conserving** or **anticipatory disk scheduling**). By waiting, a new and “better” request may arrive at the disk, and thus overall efficiency is increased. Of course, deciding when to wait, and for how long, can be tricky.