Hard Disk Drives

Hello, and welcome to this lesson dedicated to hard disk drives. Hard disk drives are largely mechanical devices, and therefore suffer from very high service latencies. In fact disk access latencies are so high that they to a great extent dominate the overall system performance.

We will start by looking how the hard disk drives are structured, and what needs to be done to get to access or modify the data stored on them.

Once we understand how hard disk drives operate, we can build an approximate performance model of such drives.

With a performance model in hand, we will see if the performance of the disk system can be improved by judicious scheduling of disk operations.

At the end of this lesson you will understand what affects disk performance, and how the operation of disks affects the overall system performance. This will then lay the foundation to understand some of the decisions in the design of file systems later.

Let’s begin.

Disk Structure

Here we see an illustration of hard disk that has been opened so as to provide a view of its internals.

The data is stored on the magnetic surface of a stack of platters, which spin together around a spindle.

The data is read or written using disk heads, one for each platter. Sometimes both sides of the platter are used, with two heads per platter.

The head is swept along the radius of a disk using a disk arm.

In a disk with multiple platters we have an entire rake of arms, which all move in and out together.

The arm is actuated by a motor, which in turn is controlled by appropriate circuitry.

[Slide 3]

If we look at a platter, or a “surface” if we use both surfaces of the platter, we have the spindle in the middle.

We have an arm, which rotates around its own axis.

At the tip of the arm is the head, which is close enough to the surface that it reads or affects the magnetic field under the head. In this way data is read or written from or to the disk.

By rotating around its own spindle, the arm can sweep across the entire width of the surface. Combined with the rotation of the disk, this movement of the arm allows the head to cover the entire area of the disk surface.

[Slide 4]

Once the disk arm moves into a given position, the concentric area covered by the head as the disk spins is called a “track”.

Blocks of data are clustered in sections of a track, called “sectors”.

As the head moves, it covers different tracks.

[Slide 5]

Let’s look how this surface fits into the overall disk drive

[Slide 6]

<No text here. This is an intermediary slide.>

[Slide 7]

We remember, that in a real disk there are more than one platter rotating around the spindle.

These platters are stacked on top of each other.

Since the heads are at the tips of a rake of arms, all the tracks that can be covered without moving the heads are arranged in form of a cylinder, one track stacked on top of the other.

All the sectors in a cylinder can be accessed without moving the head at all.

Disk Performance Model

<No text here. This is an intermediary slide.>

[Slide 9]

This gives rise to a simple approximate performance model for hard disks.

[Slide 10]

Let’s get a feeling for how long it takes to read a block of data stored on the disk.

First, the head has to move from the current track to the track that contains the sector with the requested block.

The time to move the head is called the seek latency for this operation.

Once the head is positioned, we have to wait for the disk to spin until the sector is rotated under the head.

The time to do that is called (5) the “rotation latency” of for this operation.

Finally, we need to wait until the data is transferred from the sector to main memory.

This is called the “transfer latency” for this disk operation.

The total disk access time in this simple model is the sum of the seek latency, the rotation latency, and the transfer latency.

Keep in mind that this is a very simple model of a disk, as it does not take into consideration many optimizations that are designed into today’s disks. Most importantly, we don’t consider caching. The cost of accessing data stored on disk sectors is so expensive that the addition of even a small memory cache significantly improves the disk’s performance.

[Slide 11]

Let’s look at these latencies in a bit more detail.

The seek latency is defined by how long the arm takes to move from the current to the new track. We assume that the head needs to accelerate when it starts moving and then decelerate when it reaches the new track. We represent the time taken for this by the constant S. Once the head has accelerated, it moves at constant speed M tracks per time unit. If the head has to traverse N tracks, the time to do so is therefore the number of tracks times M plus S.

Once the head is positioned on the right track, we have to wait for the sector to pass under the head. Since we don’t know where the sector is when the head reaches the track, we have to make a guess, and having to wait for half a rotation is as good a guess as any. We therefore have to wait for the disk to make a half a rotation in average for the sector to pass under the head. If the disk rotates at R revolutions per time unit, the rotational latency is 1 over 2 R.

The transfer latency is bound by the smaller of two parameters, the speed of the disk and the speed of the IO bus. Let’s assume that the disk is the limiting factor here. In this case, the transfer latency is defined by how quickly the block of data passes by the head to be read. If the block contains B bytes, and the total number of bytes stored on the track is N, then it takes B over N times a rotation to transfer the data from the disk surface to memory. The transfer time is therefore B over N times 1 over R.

As we pointed out earlier, the disk access latency, or disk access time, in our simple model is the sum of the Seek Latency, the Rotational Latency, and the Transfer Latency.

Disk Scheduling

What can we use this simple model for? It is actually quite helpful.

Assume for example that you are designing the disk IO system for your operating system.

In your system you have threads, some of which are getting blocked waiting for the disk to handle read and write operations.

We say that these threads are blocked, each with one request for disk IO in a simple case, on a so-called disk queue.

As the requests are handled by the disk, they are dequeued, and the thread resumes execution.

Given the cost of these disk operation, one may wonder, whether it would make sense to explicitly schedule these disk requests and forward them to disk in a fashion that would optimize the disk performance.

One could think of this as integrating the request queue maintained in the device driver of the disk with the disk queue of waiting requests.

The idea sounds intriguing, but it needs to be carefully studied before proceeding with an implementation, of course. We will do our evaluation by comparing the total latency to handle a given sequence of requests for a number of scheduling policies. We will see that, given our simple model, we need to know only the cylinder for each request. The specific track or even the sector on the track makes no difference.

Evaluation of Disk Scheduling

Let’s look at the model again, and let’s see what we actually need to measure to compare different scheduling algorithms.

Since we are comparing the total disk access time for a given set of requests, the number of requests is the same for all algorithms.

We first notice that the rotational latency, which is incurred for each request, only depends on a disk parameter, R. We can therefore ignore it because it will be the same for all scheduling algorithms.

Similarly, the transfer latency is the same for all scheduling algorithms.

If we look at the disk access time per request, we notice that the only latency that may be influenced by disk scheduling is the seek latency.

So let’s have a closer look.

The start-up time is incurred for each request, and is therefore same for all scheduling algorithms, since it is a disk parameter.

The track traversal time is a disk parameter as well, but the number of traversed tracks may depend on how requests are scheduled.

Let’s explore whether the total number of tracks to be traversed can be reduced by judiciously scheduling the disk requests.

FCFS Scheduling

Let’s start with First-come-first-served, or “no scheduling at all”.

We have a simple disk with 200 tracks, and we plot the head movement, similarly to a seismogram.

The request sequence that we will use throughout this evaluation is given here, with the first request being for a sector on track 98, and the last request for a sector on track 67.

We observe the head movement as the queued requests are being served.

We serve the requests in the order they came in, following a first-come-first-serve policy. We observe how the head moves quite a lot, serving request on one end of the disk alternating with requests on the other end.

After this request sequence, the disk head has traversed 640 tracks.

Clearly, the benefits of FCFS is that it is simple, and that it is fair. Nobody is unduly delayed or even starved.

As we will see in a minute, the main problem with FCFS for disk scheduling is that it leads to high average disk access latencies.

Shortest-Seek-Time-First (SSTF)

Let’s try something different. Rather than simply serving the next requests we serve that request in the queue that causes the minimum head movement. In other words, we serve the “closest” request.

This algorithm is called shorted seek time first.

Let’s see whether this algorithm reduces the head movement. We immediately notice that the head moves less.

Significantly less in fact, with a total of only 236 tracks traversed. If we compare this to the 640 tracks for FCFS, this is quite a reduction.

The advantage of SSTF is therefore clearly that it reduces the head movement, quite dramatically.

The problem with this algorithm is that it easily leads to starvation. If the head is at one end of the disk, and requests for this end keep coming in, requests at the other end of the disk may get starved. This could be improved a bit by the addition of an aging process, similarly to what we encountered in CPU scheduling. This would still lead to significant variance in disk request latencies, and there are better algorithms to deal with this problem.

Elevator Algorithm (SCAN)

One such algorithm is the SCAN algorithm, also called Elevator algorithm.

In this algorithm the disk arm simply scans the disk surface from left to right and back and serves all requests that it finds as it traverses the disk.

In this particular case the arm is picking up the first few requests as it moves from right to left. Once it reaches the end, it reverses the direction and handles the requests as it traversed the disk from left to right.

In this particular example, the head movement is also very small, with a total of 236 tracks traversed.

The advantage of this algorithm is certainly the reduced head movement compared to FCFS.

The problem with this approach is somewhat subtle. If we observe what happened when the head turns around at the end, we notice that there are no or very few requests until the head moves towards the middle of the disk. After all, we just handled all requests on the previous scan, just before we turned around. This has two problems: First, whenever the head turns around, it will encounter quite a stretch of tracks that have no or very few requests. Second, the request in the middle tracks are handled very regularly, while the requests at the end are handled very unevenly, with the head passing twice in a row very quickly before and after it makes the turn, and then not for a long time, as it moves to the other end and turns there.

Circular SCAN (C-SCAN)

An algorithm that addresses this is called the Circular Scan algorithm.

This algorithm is similar to the elevator or SCAN algorithm, except that it scans always in the same direction.

[--] Once it reaches the end, it snaps back to the beginning and starts over with the scan.

This approach reduces the service time variance that causes difficulties with the elevator algorithm.

One problem with these SCAN based algorithms is that they always scan to the end of the disk. If there are no requests left, this is a waste of time.

LOOK, C-LOOK

Two variations of the SCAN and circular SCAN algorithm, called LOOK and Circular LOOK, address this issue by looking ahead if there are more requests to satisfy.

If there are none, the SCAN turns around and the Circular LOOK snaps back, this time to the first request, and not to the first track.

Hard Disk Drives Conclusion

With this we have come to the conclusion of this lesson on hard disk drives.

We have looked at the structure of hard disk drives, and how head movement and the spinning of platters allows for the entire disk to be read and written to.

This leads us to develop a simple performance model for hard disk drives. In order to reflect disk drives more realistically, the model would have to include other aspects of disk operation, such as caches for example.

We put our little performance model to good use to make a comparative evaluation of disk scheduling policies. We saw that judicious scheduling of disk requests can significantly reduce the head movement and therefore significantly increase the disk performance.

One may think that such a thorough discussion of hard disks is irrelevant today, as disks are being phased out and replaced by various forms of solid-state storage, such as NAND Flash storage. These new technologies have their own interesting performance issues, however. NAND Flash, for example, is very slow at erasing data in order to over-write it. When analyzing the performance of Flash memory, therefore, this quirk must be included in the underlying model, leading to a discussion that is very similar to what we did in this lesson.

We sincerely hope that you enjoyed this lesson on hard disk drives.

The type of reasoning that we learned in this lesson has been driving the design of file systems and the secondary storage back-end of virtual memory systems for decades, as we will learn in future lessons on file systems.

Thank you for watching.