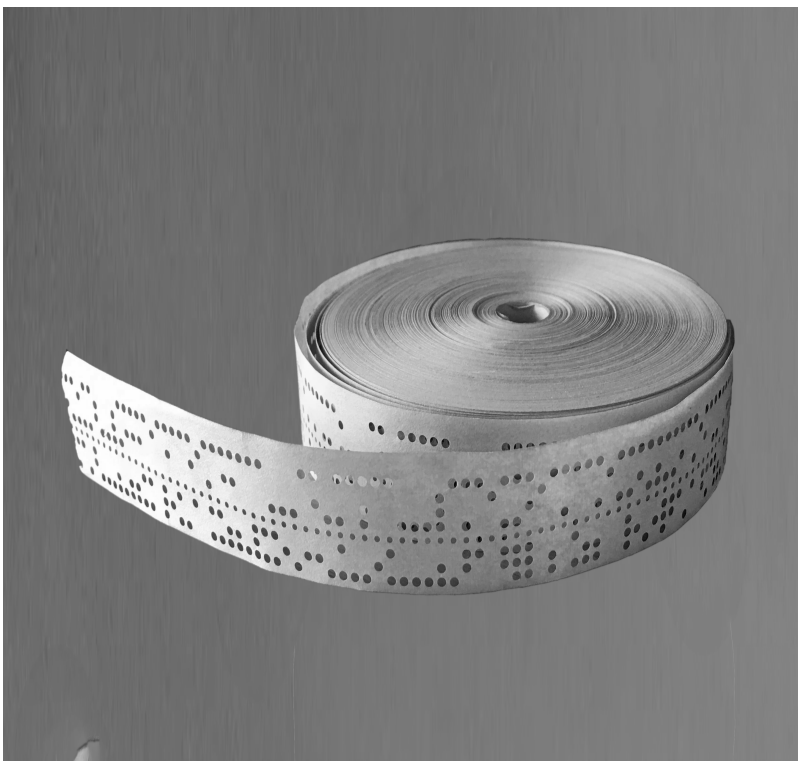
Data Storage 7



Punched-Hole Paper Tape, 1960/70s era, data storage system. Holes punched in the tape represented binary data values. This system was used as a basic storage medium until magnetic media became less expensive and more convenient toward the mid 1970s.

[96] Indeed, punched cards as a data storage medium were used throughout most of the 1900's, following Herman Hollerith's inventions of the 1890's. A punched card is shown in Figure7.1

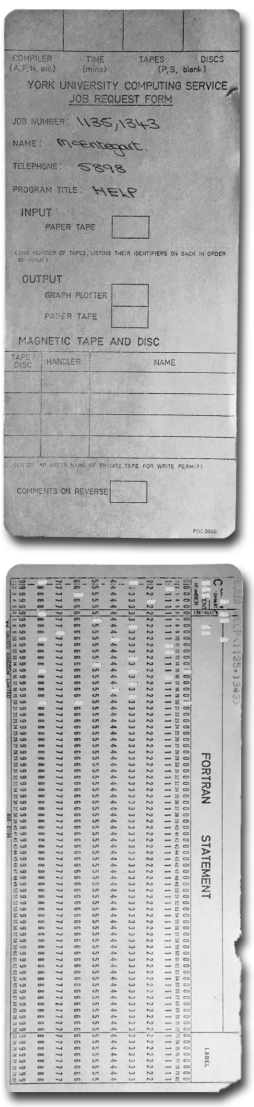


Figure 7.1: A punched card and job card from a FORTRAN program.

7.1 Storing Data

In this chapter we will examine a key additional component that is needed to make a computer system complete. This will include a significant investigation of data storage, one of the most important components not directly supported on a computer motherboard, and which comes in a number of forms.

7.2 Disks and data storage

As we have already observed, computer systems rely upon several different kinds of data memory, which we refer to as the physical memory of the computer system. However, the data storage capabilities of memory are often volatile, and rarely big enough to store all of the data the system will ever use. To overcome this limitation, computer systems have long utilised the concept of **backing store**, or in modern day terms, **disk storage**.

Computer systems of the 1960s and 1970s relied upon fairly antiquated technologies to store data in non-volatile fashion. **Punched cards** and then **punched paper tapes** were used to store data using holes in paper and card to represent storage symbols, which could then be read by a paper tape or punched card reader[96]. Later, **magnetic tapes** become available, and the natural next step was to replace holes in paper tape with magnetically manipulated data signals. Some time later, the concept of **magnetic disks** became feasible, and until recently this has been the basis for most non-volatile but rewritable data storage systems.

7.2.1 From tapes to disks

The original magnetic tape systems, which you may only have seen in old sci-fi films running in the background in some 1960's science lab, made use of the concept of a **magnetic recording head**, not that different to those used for recording audio tapes. In a fairly well developed tape storage system, the tape was wide enough to accommodate multiple recording heads across its width, as shown in Figure 7.2, in such a manner that a number of bits could be recorded at each tape position. These adjacent bits were known as **tracks**. Then, as the tape was moved (**spooled**) through the **read/write head assembly**, data could be read or written. There is an obvious problem with this system, and

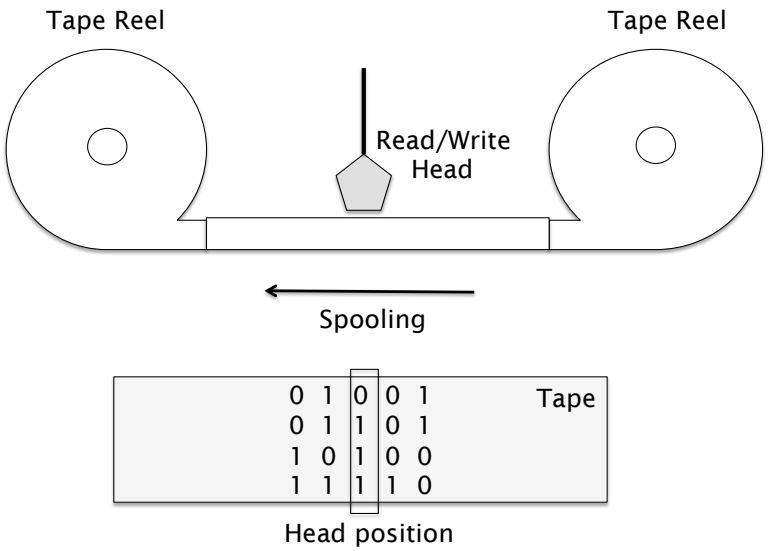


Figure 7.2: Tape System Conceptual operation. The tape reel allows tape to spool from one reel to another, in either direction. The read-write head typically has multiple positions creating multiple bits per tape region. These bits are then read or written in parallel.

that is the issue of **serialisation** of data storage. If we were to start at the beginning of a tape, and wanted to access the data at the other end, then the tape machine would have to spool the tape at high speed until we reached the point on the tape we wanted to access. This often takes far longer than actually reading the data. Therefore, there are two time factors for a tape storage system, the **seek time**, and the read (or write) **data transfer time**.

This observation is quite important: data transfer rates can be quite high with tape systems, once the data location has been reached by spooling to the correct place, but there is a disproportionate delay in reaching that position.

Magnetic tapes (in cartridge form) can still be used for emergency data backups due to their large capacity and robustness. However, it soon became apparent that significant speed gains could be achieved by rearranging the magnetic surface into a two-dimensional storage medium (as opposed to tape which is as good as one dimensional).

Although tape systems are seen as somewhat of an old technology, they have not disappeared. Tape systems have continued to evolve, primarily for the purposes of providing high density back-up storage for large quantities of data, for data security, or archiving[97]. Indeed, companies such as SONY, IBM, and FujiFilm Corp, and others, are continuing to push the limits of tape storage using advanced materials and magnetics

[97] For example, an automated robotic tape storage library can store hundreds of petabytes of data.

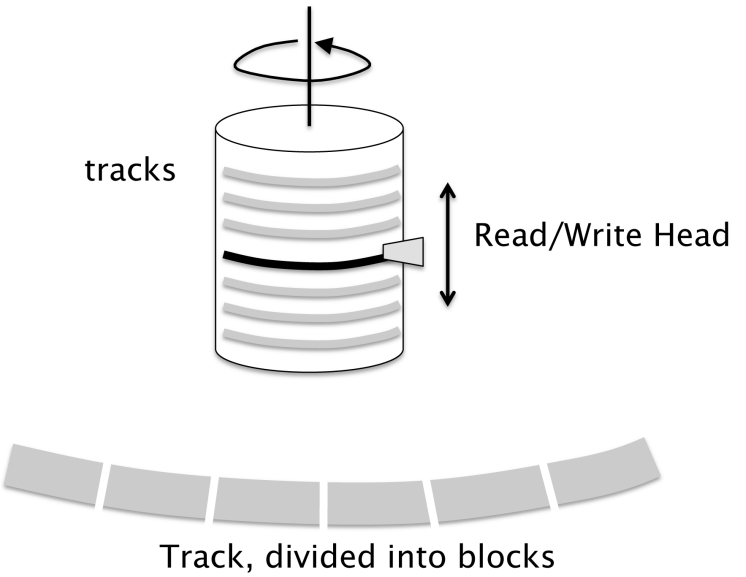


Figure 7.3: Drum Storage concept. The drum is divided into single tracks, like many short sections of tape. Instead of spooling a very long tape, a drum can jump rapidly between tracks (seek) to get to the data of interest, making access time faster.

science. Typical tape capacities have grown by about 40 times in the past ten years, and are likely to continue to improve well into the 2020's.

The first system to move to a true two-dimensional storage medium used a rotating **drum**, which was simply a cylinder upon which magnetic material was deposited to create a recording surface. Such a system is shown in Figure 7.3. Conceptually it was little more than a very wide piece of tape formed into a loop. The advantage of this system was that the drum effectively had hundreds of tracks, and rather than having to spool through every piece of data, the drum reader could jump to the track of interest and then rotate the cylinder by a small amount to get to the particular data block. Thus the seek time was much faster than tape. Later, for reasons of compactness, drums were replaced with **disks**.

A disk (or disc) is also a 2-dimensional magnetic recording surface, and just like the drum, it can rotate, and the read-write head can jump quickly to the track of interest and then rotate a little way until the data within the track is found.

7.2.2 Modern disks in detail

Since disk storage is the only major storage medium used almost universally in computer systems at present we will focus upon this system in detail. We will find that the same issues of data transfer rate and seek time still apply, though they are much more efficient in a modern disk system. At this point it would help to look at a disk unit conceptual diagram, and get to know some of the key terminology. This is shown in Figure 7.4.

As we can see in the diagram, a modern **Hard Disk Drive** (**HDD**)[98] often consists of multiple disks, known as **platters**. These can also be double-sided, allowing data to be stored on both surfaces of each platter.

In the example of Figure 7.4, there are three platters, so there could be up to six recording surfaces[99] . Each surface can be accessed via a read-write head. In most systems these are mounted on a single **armature**, but some systems have been designed with heads that can move independently of each other[100].

In order to locate data, the disk unit control circuit translates the request for a particular data block into a platter location[101], and then a particular track. The drive then locates a particular sector of that track, within which the data of interest can be found. In a disk unit the time taken for an armature to move to the required track is the **seek time**. Finding the sector requires rotation, and hence incurs **rotational latency**.

This description highlights an important point about disk storage systems: the smallest data block that can be read or written is based on the **sector size**. Sector size might, for example, be 512 bytes. As a result, a file[102], which can be an entirely arbitrary size, has to fit into an integer number of sectors. For example, a small text file might be 300 bytes long but it will still occupy (or at perhaps more precisely 'reserve') an entire sector, since we cannot store data in smaller blocks. A larger file, say 5000 bytes, needs ten sectors, and so on.

You may notice, then, that it is very unlikely that a file ever occupies exactly 'n' number of sectors, and there is always some wasted space on the disk for a given file. If the sector size is 512 bytes, then the worst case **sector wastage** per file must be 511 bytes, and the best case wastage is zero. So on average, each file wastes about 0.5 sectors of storage space (256 bytes in the case of a 512 byte sector).

[98] The Hard disk is a term used to refer to internal drives that rely upon metal platters, often stacked as mentioned. Floppy disks, on the other hand, are single platter removable disks, where the platter is made of a plastic material. These are low storage density storage disks designed to be removed and transferred from one computer to another. These days, floppy disks are rare, as USB stick drives are far more convenient.

[99] However, one of these is usually reserved for special purposes within the drive system, for example to help align heads accurately, so in actual fact, three platters translates to five usable data surfaces.

[100] This idea has recently seen a resurgence of interest as drives have become much larger in capacity whilst disk caches have not kept pace. Drive manufacturer SEAGATE is a leader in this area currently.

[101] This results in **controller delay**. However, this is a very small time period compared to other factors, and is generally ignored.

[102] A file is simply a block of data of a particular size, stored in memory or on disk.

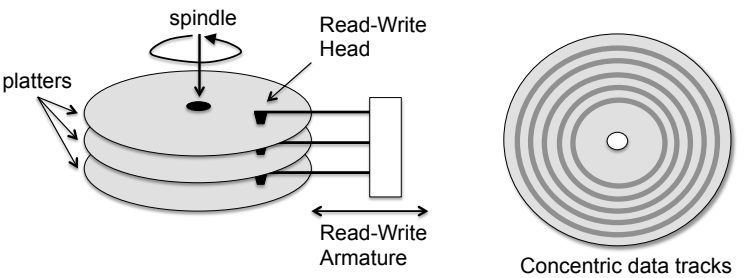


Figure 7.4: Disk Unit Conceptual Diagram. Showing the circular disk layout, and an arrangement with multiple platters on a spindle.

Now, it may not matter much if a file wastes a few hundred bytes, if the files are typically multi-megabytes in size. However, systems with large sector sizes are not efficient for applications where large numbers of small files are to be used, because the wastage (0.5 x sector size) is significant compared to a small file. This should not be overlooked when trying to determine file storage requirements. Indeed, when copying data between two disks of the same size, but different sector sizes, it is entirely possible that the data will not fit on the second disk: a somewhat confusing situation if the reason is not appreciated!

Disk systems are organised to try and minimise seek time, maximise data transfer time, and achieve the highest possible data storage capacities, whilst remaining reliable. This is a balancing act between a number of parameters. Seek time can be reduced by having more tracks per disk, and by having more platters to choose from, but this demands read-write heads to work on smaller and smaller scales (storage density is increased). Modern disk units use very high precision head positioning systems to achieve this goal.[103] Likewise, packing more data bits into a sector requires physically smaller read-write heads, and smaller magnetic areas on the disk representing individual bits (known as domains).

[103] Techniques have included **high-precision servos**, **stepper motors**, and **voice-coils**

At the same time, the **rotational speed** of the disk, measured in revolutions per minute or RPM, dictates how fast the required bits can be rotated into position and read/written. Typical speeds are of the order of 5400rpm to 7200rpm in drive technologies at the present time (2019), but higher speeds can lead to shorter drive lifetimes, more power consumption, and so on. Nonetheless, some high performance disk systems use rotational speeds up to 15000 rpm, reducing seek time, but at a cost of higher power consumption. The impact on seek time contributed by rotational latency (the time taken for the platter to perform a maximum of one rotation) is highlighted in Table 7.1, where it is clear that low rpm leads to large contributions to seek time.

Rotations per minute	Rotations per Second	Time for 1 rotation
5400	90	11.1 milliseconds
7200	120	8.3 milliseconds
15000	250	4.0 milliseconds

Table 7.1: Examples of rotational speed and contribution to total access time. Faster rotations mean lower access latencies.

When the rotational latency is added to the time taken for the head armature to move to the correct track, that is the seek time, then we arrive at a figure for the total access time for a data access. Rotational latency is also related to the **sustainable data rate** achievable by a disk unit. Whilst it represents a delay in terms of access time, because of the time taken to rotate to the start of the sector of interest, it also represents the time taken to read one whole track once the data is in the correct position. The faster that the disk rotates, the faster that data can be read, and the more densely the bits can be packed into the tracks.

If we know the rotational latency, the number of sectors per track, and the number of bytes per sector, then we can calculate data rate per read-head:

Data rate Example

Suppose a track contains 100 sectors, and each sector contains 1Kbyte of data. Let us also assume that we have an 8.3 ms rotational latency at 7200 RPM (60s/7200rpm=8.3ms). Thus, 100 sectors can be read in **8.3 ms**, assuming a single read head.

Therefore, if 100 sectors can be read in **8.3 milliseconds**, then 1 sector can be read in **83 microseconds**.

If one sector contains 1Kbyte, (1024 bytes) then we can read 1024 bytes in **83 us**.

Therefore, we can read one byte in 83x10-6 / 1024 = 81 nano seconds. And, if one byte requires **81ns**, then:

Data rate = 1 / 81x10-9 =**12.3 million bytes/s =(11.8 MB/s)**[104].

A complication here is that modern drives use the concept of **zoned bit recording**, such that the number of sectors on the outermost tracks is higher than the number of sectors on the innermost track. This is

[104] We need to be careful here, as one million bytes in decimal is not the same as a binary megabyte (1024*1024 bytes). Our decimal megabyte figure would be around 12.3 Mbytes/sec, but in binary megabytes it is 11.8 million bytes/sec. This illustrates an important point: sometimes a figure is driven by an underlying binary (base-2) property, such as sector size in this case. Therefore, we need to take care with unit definitions.

because the outer tracks are longer and can accommodate more bits at near-optimal bit-density. Rather than making sectors have different numbers of bits, ZBR allocates more fixed-size sectors per track as we move from inner to outer tracks. Data transfer rates on the outer tracks may also be higher than those for the inner tracks, since the same period of rotation delivers more bits to the head. Therefore, a very fast rotating disk with very high storage density per square millimetre of disk surface, will deliver high sustainable data read rates.

We should be able to realise from the above example that the data transfer rate can be increased simply by doing one or more of the following:

- Rotating the disk faster,
- Packing more bytes into a sector,
- Having more sectors per track.
- Reading tracks in parallel (multiple read heads)

It is obvious then, that manufacturers will benefit from developing new technologies that allow more and more bits to be stored on a disk per square millimetre of magnetic surface area, and indeed this is exactly what has happened for the past 20-30 years.

The science of miniaturising the recording of data bits magnetically on a disk surface has become quite complex, relying upon very fine precision engineering, and some fairly deep exploitation of the physics of materials. We will not go into all of those details here, but there are many articles about this quest for ever larger and faster storage systems for your own research to follow.

7.2.3 A disk performance example.

At this point, we shall try to put some of our understanding of disk mechanics to good use, and attempt to predict realistic performance expectations. We will use rpm to determine average, best, and worst case access time, data transfer rate based on sectors per track, sector size, and rpm, and calculate data read time for a 3 KiloByte block with an example.

In this case we will use some arbitrary figures for drive performance, not far from a reasonable set of assumptions. For a specific hard-disk you would of course be able to obtain the technical data from the manufacturer's web-site. Datasheets are supplied for most components in computer systems, and give a lot of insight into capabilities and limitations of particular components.

An example: estimating file access time. A drive has a rotational speed of 5400 rpm, a data read rate of 100 Megabytes/sec, and a sector size of 2 KB. What are the worst, best, and average times to access our 3 KB data?

Let us assume that deciding which platter, and which track to access is almost instantaneous (it is an internal control circuit operation taking microseconds at most).

The drive now needs to move the head to the correct track. Let us assume this seek time requires a maximum of 2 milliseconds (this would be known from the datasheets provided by the manufacturer[105]).

The disk rotates at 5400rpm, or 90 times per second, so it completes one rotation in 11.1 ms. A 3Kbyte file occupies 2 sectors of 2 Kbyte, so the disk unit needs to read 4 Kbytes in order to complete its read activity for that block of data[106]. At 100 Megabytes per sec, this requires about 40 microseconds.[107]. Now let's try some performance estimates:

Worst case:

Head movement (seek time) 2 ms, plus one full rotation, 11.1 ms,

Total access time therefore = 11.1+2ms **= 13.1ms**

Then a further 40us to read the data (transfer time).

total read time = **13.14ms**.

This implies a **peak data transfer rate** of about **0.3 Megabytes/sec**, if we assume two sectors provides 4096 bytes of data, but only 0.23 MegaBytes/sec for the actual 3 KByte file size in question.

Best Case

Head Movement 0 ms (head by chance is in just the right place),

no rotation required (by chance, the disk is in just the right rotational position),

Total access time = 0ms

Then read data 40us, and total read time = **0.04ms**.

This implies a **peak data transfer rate** of **97.7 Megabytes/sec**. Again, for the actual 3 KiloByte file it is actually lower, at around 73 Megabytes/sec.

[105] We can actually estimate this, if we know the rotational latency of the drive and the manufacturer quotes total worst-case access time, then seek time is the difference between rotational latency and total access time.

[106] Unfortunately we cannot assume that the disk controller will release the data before reading an entire sector, even if only part of it is used. For example, error checking, and other low-level considerations, may operate on a sector as an indivisible unit.

[107] To be exact, 0.039 milliseconds, though in reality the true data transfer rate might depend upon the location of the track (inner, middle, outer, etc), therefore we use an average figure here.

Average Case

From the worst and best cases, we can determine a simple average case: (13.14 + 0.04) / 2 = **6.59 ms**.

This implies a **data transfer rate** of around **0.60 Megabytes/sec**.

Note, it is important to appreciate that this only estimates the time taken for the process of getting the data from the magnetic surface into the memory buffer of the disk unit. The transfer from disk to CPU memory potentially requires further time and effort, and depends upon the bus interface used to connect the disk unit.

In this example we also observe that small files are dominated by the physics of the disk access mechanics, whereas larger files would be less impacted, and therefore are potentially able to deliver higher effective data transfer rates than smaller ones. Indeed, the actual useful data read in each case was only 3 Kilobytes, so we could have used that figure to calculate data rates and got an even more pessimistic figure. For the case evaluated, using a very small file test case, we see that average performance is about 150 times lower than the best case performance of the drive.

Whilst a simple average performance can be obtained by assuming that files are spread equally over the disk (as we did in this example), the reality is that files may be clustered together around a heavily used portion of the disk, whilst other areas might be unused or lightly used. This means that average performance in live systems might be better than this estimate. What those systems cannot do is **guarantee** that performance **will** be better, not unless they deliberately arrange files on the disk in a way that enhances this effect[108].

[108] This is of course possible: with the right file system policies, or by using suitable utilities to reorganise the disk content to optimise this behaviour with varying degrees of sophistication.
However, since this behaviour is often a run-time characteristic, it may need to be evaluated through profiling of real systems during their use in order to derive suitable optimisation insights.

It is also important to appreciate what this means for effective data transfer rates. The disk manufacture may quote a data transfer, let us say 500 Megabytes/sec, meaning the ability to transfer data between disk and CPU system bus. However, this does not mean that we can read data off the disk at that rate in a realistic file system, or indeed at the rotational speed of that particular disk. A more useful figure quoted by drive manufacturers is **maximum sustained transfer rate**. Yet we see in our example that even that figure is not telling the full story.

If we know how big our files are typically, then we can make fairly accurate estimates of how much data a hard disk can deliver under peak load. We could alternatively use a benchmarking tool (which tests the disk with various file sizes and combinations) to give a reasonable measure.

Importantly, you should appreciate that the physical disk unit impacts upon wider performance. If we had a system that operated quite nicely with a particular disk unit, and then this was upgraded to a larger disk, it may actually reduce the system performance, due to different disk mechanics, and other aspects of its internal organisation. This could be a serious issue. Consider, for example, a situation where a back-office server system is handling enquires from a customer ordering system at an airline office, just about coping with the workload, and then suddenly the system appears to be taking longer to do everyday tasks because perhaps the RPM of the new disk is lower after an upgrade.

7.2.4 The cost of fragmentation

If we were to repeat the previous example, but assume a much larger file size, then the data read rate and read time would eventually become dominant. However, this perhaps reveals another over-optimistic view of how files are stored on disk. In reality, the possibility of finding many sectors of a file next to each other on the same track is actually not 100%. Instead, file data is often spread across the disk, with sectors scattered here and there. This is known as **fragmentation**.

Imagine you work in a shop, and the customer asks for five items, so you go to the store room, and the first shelf you come to has the exact same five items sitting right next to each other on the shelf. That's great: it takes you no time at all to access the items and bring them to the customer. Another customer comes in, and also asks for five items, but this time, each item is on a different shelf, and in a different position on the shelf. It will take much longer to do the fetching this time.

This is the principle of file fragmentation. When all of the sectors of a file just happen to be on the same track, and in just the right order, this is known as a **contiguous file** (it has no gaps, and appears like a continuous series of data blocks). When a file does not have all of its sectors on the same track, in just the right order, and with no gaps, then this is known as a **fragmented file**.

A fragmented file with **n** sectors could have up to **n** separate fragments. On the other hand, it may have none, or only have one or two discontinuities in its series of sectors. In general then we could say that any file will have between zero and n fragmentations. But is this a problem?

Let us consider our previous file access example again. We had a 3 Kilobyte file split into two 2 Kilobyte sectors. In our previous calculations we assumed (without realising it) that the file was contiguous. However,

if those two sectors were in different places on the disk, then things get a lot worse. Consider two cases, one being our worst case read time from the previous example (the contiguous case), and the other being the same case but with a fragmented file.

Fragmentation impact estimate

Contiguous Worst Case:
2ms + 11.1ms +(2 x 0.02 ms) = 13.14 ms

Fragmented Worst Case: (with one fragment)
(2ms + 11.1ms + 0.02ms) + (2ms +11.1ms +0.02ms) = 26.24 ms

So what has happened is that the access time for head and rotation mechanics has to be duplicated for each incidence of a sector that is fragmented. In general, every fragmentation of the file could result in a worst case access time per sector read. The best case could still be close to zero, so the average is now around 13ms. Clearly if the file has lots of fragmentations, then the read time will become quite slow, even to the degree that it is noticeable to the user to an extent that is undesirable.

One solution to this problem is to use a **defragmentation utility**, which is a program that iteratively reads and re-writes the entire disk content in such a way that it attempts to make every file contiguous: all sectors placed next to each other on the same or immediately neighbouring track, and in the right order, for example. This significantly impacts on system performance, theoretically maximising the transfer rate for each file.

7.3 Disk cache

There are additional tricks that will improve disk performance, and our old friend, the cache, pops up again as a solution. Most modern disk units have an inbuilt cache memory module, which acts as a data buffer and as an optimising function. Consider the following examples that might be relevant:

- A computer program reads the same file repeatedly at intervals.
- A computer program begins to read data from a file, stops part way through and then resumes reading
- A computer program reads multiple files in an intermixed fashion (jumping from one file to another, known as interleaving)
- Files are fragmented.

We can see here, if we understand how cache works, that these very common cases can be helped by cache. Where the same file is read repeatedly, then keeping that data in a cache means we no longer have to go through the long-winded process of accessing data on the disk. We eliminate all of the mechanical access time costs associated with that file.

Where a program stops and starts during file reading, each resumption of reading would require a new access process, potentially adding up to a full 13ms each time, as calculated in our example. If, on the other hand, the cache is able to read the whole file in an uninterrupted flow, it can appear as if the next part of the file is instantly available when the program resumes reading, with zero apparent delay.

Here the cache can potentially read the whole file in one operation, with the expectation that it will be needed later. Of course if this turns out to be a bad assumption, then electrical power has been wasted reading the extra data only for it to be ignored, and this means there is a cost-penalty as well as a benefit.

Where a program is jumping around between multiple files, the ability to cache some or all of these files will also eliminate the undesirable **interleaved seek operations** that this entails. Again, performance will potentially be much better when this happens.

Finally, in the case of file fragmentation, a badly fragmented file, once read into cache, no longer suffers the consequences of fragmentation.

Note that whilst disk cache is a valuable enhancement to general performance, it cannot make the initial file access faster, only those successive accesses to the file. Also, just like memory cache, there is limited capacity and content can be discarded from the cache to make way for new data, so we cannot assume that a file once read will always be in disk cache.

For more on this topic, see CASE-STUDY **??**.

7.4 Solid State Drives (SSD)

A relatively recent development in data storage is the **Solid State Drive**, or **SSD**. This technology is still developing, and so is relatively costly compared to the much more mature state of magnetic disk system hardware. Even so, the SSD is much faster.

We have already encountered the idea of non-volatile memory in Section 5.2, and we learned that certain memory systems, such as **flash memory**, allow data to be read and written electronically, yet retained after power off. This is essentially a storage device in the same way that an HDD is a non-volatile storage device.

In theory, therefore, we could use flash memory in place of disks. This is essentially what an SSD does. They contain large amounts of flash memory, organised alongside a memory controller that converts disk access commands into memory reads and writes. The result is that the SSD looks identical to an HDD as far as the CPU and operating system is concerned. However, the SSD has no mechanics, and therefore none of the complications of head seek, sector rotation, or magnetic read/write constraints.

All of this makes SSD very fast, even when files are non-contiguous, and even where programs interleave between different files frequently. Indeed, whereas a typical HDD might manage a 100 Megabytes/sec sustained data transfer during a file read, an SSD could potentially achieve 20 times this performance[109] , and without any mechanical access time delaying the start of data transfer and without making small files inefficient. The only negative is that write speed is a lot lower, due to the way flash memory has to be programmed. Flash reads are very fast but flash writes are fairly slow.

[109] Indeed, the main restriction on SSD read performance is often the system bus interface rather than the SSD memory chips themselves, which can be organised in many ways to boost data read rates if desired.

Because SSD's are still multiple times the price of magnetic storage for a given capacity, they have to be used sparingly, unless money is no object, and if the purpose is biased heavily toward read performance then it maximises the benefit. A good example would be to store operating system and application code to make everyday systems run faster and more smoothly. The operating system and installed software packages do not change often but they are read heavily during use. Likewise, a video server could deliver semi-permanent data content at high speed to many viewers at the same time, and the time taken to load new videos onto the SSD would not be a big issue as it would likely be done during a quiet period, such as the middle of the night.

SSD's can also be used for accelerating disk swapping in virtual memory, or for acting as a temporary work space in very file intensive data processing tasks. However, costs will reduce in time, and in all likelihood, we will see SSD become the standard system component over the next decade, particularly if faster write speeds can be achieved. HDD systems are then likely to be phased out, just as magnetic tape was.

Interestingly, at the time of writing the first edition (2019) a recent trend in hybrid drive technology (SSHD) had begun to be seen, with very large magnetic disks combined with fairly small SSD memory technology to create a more cost effective but large format drive technology. As with all of these disk technologies, there are niche areas where each may perform well, but they are not perfect in all settings.

7.4.1 SSD memory wear

In all current flash-based memories, the ability to overwrite data cells in the structure is limited to a certain number of reliable write cycles over its operable lifetime. When a memory cell is near its write-limit, it may begin to lose data integrity; typically this happens over periods of years. Read data capacities, on the other hand, are effectively unlimited.

An SSD bit-cell can also theoretically become entirely non-functional with excessive use[110] . In order to avoid this becoming an issue, a drive will attempt to write to different blocks of physical memory each time a new block is stored. This distributes the write activity across the whole SSD memory space rather than creating excessive use in small areas. This method is known as **wear-levelling**.

[110] For example, the SAMSUNG EVO 850 Pro series SSD has a wear rating of 300TBW (MZ-7KE1T0, 1,024 GB), and 450 TBW (MZ-7KE2T0 2,048 GB) respectively, where TBW is TeraByte Writes. Interestingly, the doubling of drive size doesn't yield a doubling in write capacity in this example.

The wear issue may not be as problematic as it might seem: A 2 Terabyte drive, rated at 300 Terabyte total guaranteed writes, doesn't sound like much. However, the problem does not affect read cycles, and files are rarely overwritten repeatedly in their entirety at high frequency. Rather, only small parts of files are usually updated. Secondly, remember that drives also have inbuilt cache, and if a write-back cache is used, then the vast majority of repeated writes will hit cache, and much more rarely the actual SSD storage flash array.

Even if we consider a case where new files are being generated daily, the wear-out of bit cells may not be quite the worry it might appear to be.

Consider a 2 Terabyte (2 TB) SSD unit, where a user fully updates 10 Gigabytes of data files every day (quite a heavy user by typical standards): How long should this drive last if the storage cells are guaranteed for a total write capacity of 300 TBW ?

Wear Capacity Example (using decimal megabytes)

At 10 Gbyte per day, the user would write **3.65 Terabyte per year**.

At this rate it would take **300/3.65 = 82 years** to wear out the drive.

Most SSD units are only guaranteed for 3 to 5 years anyway, so there is no prospect of this user wearing out the drive during its planned lifetime, even at ten times the estimated use.

Now consider a few other examples of user data-write requirements:

- **Operating System:** maybe 10 Gigabytes of storage, perhaps 10% of files updated every 3 months, impact 4 Gigabyte/year
- **Computer Games:** typical game 10 Gigabyte, install one new game per month : 120 Gigabyte/year
- **Document Editing:** typical document 5 Megabyte, change per day 5%, impact 0.25 Megabyte/day, about 65 Megabyte per year for a 5-day working week.

So unless a user is doing something very demanding, the problem of wear is unlikely to be encountered[111] . In such extreme cases, the use of a highly optimised HDD configuration would be preferred. Meanwhile, if a choice of drives comes down to paying a high premium for a relatively marginal increase in write-wear rating, think carefully about the extra money: will it really deliver a justifiable benefit?

[111] A little care is needed here however. If most of the SSD is occupied by static file content, in other, words files that are rarely/never updated, then any other frequently changing files will focus their wear effects on a smaller portion of the SSD, causing wear-out to occur more quickly. Fortunately, this is a recognised problem and 'static wear-levelling' has been developed to counter this by occasionally moving static file content around the SSD sectors, to rebalance the wear effects. This is somewhat analogous to rotating tyre locations on a vehicle, to even out tread wear.

7.4.2 Storage density

Storage density is a measure of how much data is able to be stored in a given space. In the past 40 years, magnetic storage devices have changed from modules the size of a washing-machine, to something smaller than a cigarette packet in the case of so-called 2.5 inch drives.

Meanwhile the capacity of these systems has grown from perhaps 10 Megabytes to a point approaching up to 10 Terabytes. Solid state memories are likely to take this an order of magnitude further; indeed, SSD units typically started out following the same form-factor as 2.5 inch HDDs, but even this is already being surpassed by newer, even more compact, SSD design styles. Figure 7.5 shows examples of recent HDD and SSD modules for size comparison.

Figure 7.5: Modern HDD and SSD units compared. Showing a 3.5 inch 80GB HDD (left) a 2.5 inch 500GB HDD (middle), and a 500GB SSD (right).

Drive	Sustained Read Mbytes/Sec	Sustained Write Mbytes/Sec	Price
CRUCIAL MX500 1TB SSD	483	429	£95
CRUCIAL P1 3D NVMe PCIe M.2 1TB SSD	1419	1338	£99
TRANSCEND SSD370 1TB SSD	473	392	£352
SEAGATE Barracuda 1TB HDD (2016)	170	146	£32
SEAGATE SkyHawk 1TB HDD(2016)	140	138	£48

Table 7.2: A few example HDD and SSD disk storage options (data as reported from hdd.userbench.com 2019)

7.4.3 Buyer beware

When considering suitable components for a system, it is important to realise that not all components are created equal. Performance and prices vary for components quite widely. This isn't just a question of up-front cost, speed, and capacity, but might impact upon power consumption, and reliability.

This is certainly the case for storage HDDs and SSDs. A few examples are given in Table 7.2. Although these drives may be of different release dates, this serves to illustrate that performance is not always directly linked to price. There may be other reasons why a drive with apparently lower read/write performance is more expensive. The key thing is to ensure that, when specifying system components, plenty of effort is put into researching the options. Small increases in price might yield big gains for system performance, and indirectly that may actually mean the deployment of that system is more cost effective.

7.5 Storage reliability and fault tolerance

When it comes to disk data storage, there are many applications where data integrity is critically important. This could include such applications as financial record keeping, medical data scans, safety critical systems, and so on. In order to avoid errors in data storage, systems have been developed to deal with these situations.

There are actually several faults that may arise, which would be undesirable. The first is a **mechanical failure** of an HDD during use, where the HDD just stops working altogether, and thus makes all of its data unavailable. Partial recovery of data on such a disk is possible, with the assistance of a specialist, but it cannot be done in timescales of hours, more likely days or weeks. SSDs do not have mechanical parts, and therefore offer one way to avoid such problems. That is not to say that an SSD is entirely failure proof however.

The second scenario is **file corruption**, caused by a number of things, and all are generally unpredictable.

Typically, the probability of bit errors in read operations on disk units is vanishing small - of the order of one-hundred trillion to one according to some drive manufacturers. For most applications, almost too low to be a major concern, though not if your HDD is controlling a nuclear power station perhaps.... Because there are different levels of severity of consequences, there are also different levels of dealing with these problems:

For simple bit errors in the storage system, data blocks can be stored with additional information, called an error correction code, or ECC[112]. This can help to identify errors, and in some cases correct them. At the very least, the software will be able to know that the file it is reading is corrupted, and take appropriate action.

[112] There are a variety of methods, CRC: cyclic redundancy check, for instance. Of course these are not cost free - the codes use some of the available storage space, making files larger by a degree that is related to how substantial the ECC scheme is. There is also a small cost in terms of CPU or disk-controller effort and time required to perform the ECC functions.

For disk unit failures, we can use **redundant storage arrays**. In simple terms, two or more disks can store the same data, and if one fails, another is still available. However, a less costly option may be just to frequently back up the HDD content so that a recent version of the data is always recoverable. This very much depends upon how quickly it is required to get back to an operational state: days, hours, or minutes?

For high degrees of data integrity, we can use more complex disk system configurations, involving duplicate resources and data storage. The **RAID** disk system (redundant array of independent disks) provides a number of different versions of disk duplication (redundancy) to provide

for various scenarios and levels of resilience against data loss. RAID can also be used to improve performance, by combining multiple disks to make them appear like one faster disk. The full range of possibilities defined by the RAID standards (such as RAID0, RAID1, RAID2 ...) are rather complicated, and best left to the motivated reader to investigate. A couple of brief examples are worthwhile however:

RAID0: simply combines multiple drives, using a technique known as **disk striping**, to appear like one faster disk (because they can be read in parallel to multiply the sustained data read rate). The downside of this is that there are more drives to fail, so failure rate may be higher: we will see more about this in Section 14.8.

RAID1: uses **disk mirroring**. In this case, two or more drives have identical content. The advantage is that (a) if one drive fails, the others remain accessible with the same data, and (b) the drive that can read the data first will always deliver the data ahead of the others, and therefore there is a probability that average access time of the whole RAID system will be reduced, meaning faster data access and improved average data transfer rates.

So, we have observed that where data integrity is important, there are methods available to improve data resilience, and these systems can (at a higher cost) be quite complex and sophisticated in their methods of preventing data loss.

7.6 Remote storage

Another form of data storage that is increasingly popular is the idea of **remote storage**. To fully understand this concept we will of course need to understand the idea of networks, which we will examine in more detail in Chapter 13.

Let us assume for now that a typically computer system has a network connection of some kind, allowing the computer to communicate with the internet. Somewhere else in the world, another computer also has a network connection to the internet. Consequently, the two computers can communicate and share data if they wish to do so.

This brings about the concept of **remote data storage**, including the idea of **cloud storage** and **cloud computing**. Cloud computing is another way of saying that a computational task is performed on some remote computer (the **compute server**), usually for a fee, and the results

will subsequently be sent for local display on the user's own computer (the **client machine**). Cloud data storage is simply a variation of this where the server is primarily providing stored data on request rather than computed results.

Modern operating systems support the idea of a **network drive** or remote drive, and allow users to view files that are stored on a remote server, and in a way which looks almost the same as a local set of files. The user may see little or no difference, and may not even be aware that their files are stored remotely.

With the correct connectivity to the internet, the delay in sending a request to a server via a network connection can be of the order of milliseconds, and perhaps much less for local office networks. The additional time involved in accessing a file on a remote server (**server latency**) is often small, provided the data transfer rates of the network are also high. This works particularly well for relatively small files where transfer time is not noticeably long, or for data that needs to be accessed in a streamed[113] fashion such as video, where all of the data doesn't need to arrive as a whole in a short period.

[113] File streaming, particularly for audio or video, involves sending a file across a network in sections, which are collected and buffered at the receiving computer. When done efficiently, the data arrives just quickly enough to keep the buffer modestly full of data, and therefore not using more network bandwidth than immediately necessary. When data doesn't arrive quickly enough, video or audio will freeze momentarily, which can happen when a network is being heavily used by too many users at once.

Increasingly, **enterprise systems** - that is systems deployed across large organisations - are based upon servers and cloud data and cloud computing resources. It is often cheaper to lease time on servers than to buy and maintain systems in-house. All of the benefits of rapid response to failures, automatic data back-up, and the ability to suddenly increase capacity for short periods, or to scale up as a business grows, make these solutions highly desirable.

7.7 Summary

In this chapter we have reviewed the evolution of data storage devices, from early primitive systems, to modern disk based HDD equivalents, and finally through to the latest solid-state memory-based mass storage devices such as the SSD. We have highlighted the performance characteristics of these devices, and explained how these parameters relate to the mechanisms of their systems, and understood the implications for system performance.

Meanwhile, we explored the idea of fragmentation, and data organisation on disk, and the impact this has on performance, along with techniques such as disk caching to hide these effects and boost performance.

Looking toward the future, SSD technology is beginning to make significant inroads into general system design, and in high performance computing, if used carefully in context. The wear characteristics of SSDs, and their capacity versus cost, will undoubtedly continue to improve, and it is possible that HDD technology will be phased out in all but some highly specialised applications in the next ten years. Who can guess what the next storage technology will be? It is probable that SSD will not be the last step in the storage revolution.

7.8 Terminology introduced in this chapter

Armature	Backing store
Client machine	Cloud computing
Cloud storage	Compute server
Contiguous file	Controller delay
Data transfer time	Defragmentation
Disk	Disk mirroring
Disk striping	Disk track
Enterprise system	Error correction code (HDD)
File access time	File corruption
Flash memory	Fragmentation
Hard disk drive	HDD
Interleaved seek	Magnetic disk
Magnetic recording head	Magnetic tape
Network drive	Paper tape
Platter	Punched card
RAID	Read/write head
Rotational latency	Sector
Sector size	Sector wastage
Seek time	Server latency
Solid state drive	Spooling
Stepper motor	Storage density
Sustainable data rate	Sustained transfer rate (disk)
Wear-levelling	Zoned bit recording

These terms are defined in the glossary, Appendix A.1.