¡le

new CHAPTER2, file i2dbd.tex; June 29, 1997

Chapter 2

Hardware and Its Parameters

On a clean disk you can seek forever

Thomas B. Steel jr.
at SHARE DBMS *Conference, July 1973*

2-0 INTRODUCTION

This chapter summarizes the hardware used to store data files: mainly tapes and disks. These descriptions will provide background for the understanding of their operation and performance; in other words, the concern here is what the devices do rather than how they do it. After this chapter all analysis refering to hardware will be made using a limited number of parameters and concepts. Six parameters are adequate to describe the performance of all types of current hardware with sufficient precision.

This chapter contains two major interrelated sections: the first section contains a review of the types of hardware available, and the second section discusses the parameters used to describe hardware. Along with basic performance data, cost estimates are provided since the benefits of good performance cannot be separated from the expenses associated with powerful devices. Section 2-3 presents hardware-related programming concepts, and the chapter concludes with a summary of the parameters for recall and reference.

Chapter 13 provides some further discussion of storage system architecture. The remainder of the book needs to make very few direct references to hardware in this book, due to the parameterization. New or improved storage devices are continually being developed, and any description reflects the state at the time of writing. The reduction of hardware descriptions to the small set of parameters makes it possible to use the analysis and design procedures in the remainder of the book also for devices which are not described or even not yet invented.

Typical values of parameters for the types of hardware discussed are given in Table 2-1. The use of recent devices will make the problems in subsequent exercises more interesting. The abbreviations used in formulas can be found on the inside covers and in Appendix B.

2-1 HARDWARE CHOICES

Since this section surveys the varieties of hardware available for files and Sec. 2-2 defines the parameters needed to describe their performance, some cross referencing may be necessary when encountering new terms in the section below.

2-1-1 Mechanical Storage

Some use is still made of mechanical storage for data, using storage media which were already developed before the turn of the century:

One such medium is the Hollerith or IBM *card*. Cards store data by having holes punched to encode a character per column. Cards with 80 columns of 12 bit positions each are the most common, other cards exist with 90×6 positions, and special cards are in use for attaching to inventory items.

Paper tape is still used at times to record raw data from measuring instruments. Holes, punched into eight *tracks* across according to the ASCII code (see Fig. ¡11-1¿), record data and a *check* bit.

The paper storage media can be used only once for the recording of data. The punching devices, being mechanical, are slower yet than the readers, which typically use light-sensitive sensors.

2-1-2 Magnetic Tape Storage

In traditional data-processing operations magnetic tape was predominant. Two categories, typified by cassettes and reels of tape, offer distinguishable capabilities. Tapes and all other magnetic surface devices have identical performance characteristics when reading or writing. Figure 2-1 illustrates the principle of magnetic digital recording.

Limited amounts of data can be stored by the use of *cassette* or *cartridge tape*. The magnetic tape in these packages is 0.150 inch (3.8 mm) or 0.250 inch (6.4 mm) wide and typically 300 feet (100 m) long. A single track of data bits is recorded along the tape, and a parallel track may be used for timing or addressing information. The low cost of purchase and storage of tapes makes their use attractive for the storage requirements on workstations. Attempts to standardize these types of types have resulted in a plethora of incompatible "standards", so that data interchange to workstations of different types is commonly performed over communication lines, creating distributed systems.

144pt Figure space

was Fig.2-1

Figure 2-1 Digital-tape recording.

Standards have been developed for large, high capacity tapes. The physical format has remainded stable since before 1960, but the recording density has changed in 5 steps by a factor of 30. Such *industry-compatible magnetic tape* is $\frac{1}{2}$ inch (12.7 mm) wide, comes mainly in lengths of 2400 feet (732 m) on large ($10\frac{1}{2}$-inch, 267-mm) reels, and contains seven or nine tracks across its width.

The linear density of bits in each track is measured in *bits-per-inch* or *bpi*. Some values are given in Table 2-2. A character is recorded by placing a bit in each track. The tape carries that many characters per inch of its length. On 9-track tape, a character is encoded into 8 bits, providing for $2^8 = 256$ distinct symbols. The 8-bit unit, whether used to represent characters or a binary number, is usually called a *byte*. The ninth track is used for error-checking information.

Table 2-1 assumes 9-track tape on 2400 foot reels, having a density of 800 bits-per-inch (bpi) (31.5 bits-per-mm), which uses *non-return-to-zero* encoding, as illustrated in Fig. 2-1. A *phase-encoding* method permits 1600 bpi (63 bits-per-mm). A gap of 0.6 inch (15 mm) separates blocks of data to permit starting and stopping of the tape between read or write operations. The tape units move the tape at 75 in/s (1.9 m/s) to 225 in/s (5.7 m/s) past their read-write heads.

Tape units which record and read at densities of 6250 bpi (246 bits-per-mm) have a smaller gap and provide the highest performance currently commercially available. The tape format which supports 6250 bpi use a group code transformation, using five recorded bits for four data bits and an error-correcting byte in every eighth position, so that the actual recording density is about 9000 bpi (355 bits-per-mm). Error-checking and -correcting techniques are decscribed in Sec. ¡11-3¿.

Tape units have very poor random-access qualities. The time to read through an entire tape is on the order of 4 min for reels containing 30M* characters each. Many tape units allow tapes to be read in both forward and reverse direction.

* We will use the convention where the letter M stands for *Mega* or one million (10^6), K for *Kilo* or one thousand (10^3), and G denotes *Giga* or 10^9.

174pt Figure space

was Fig.2-2

Figure 2-2 Tape-recording methods.

Techniques from video recording are influencing the design of high-capacity tapes for specialized hardware. The high storage capacity is achieved by writing data tracks *transversely* across the tape. The distance between tracks is now determined by the relationship of the head assembly speed across the tape and the forward speed of the tape itself, rather than by the spacing of the parallel heads, as shown in Fig. 2-2. Searching can be performed using the audio and synchronization tracks along the tape edges. Search speeds of 1000 inches per second (in/s) (25.4 m/s) are being used.

207pt Figure space

was fig. 2-19

Figure 2-3 Picker mechanism of an IBM 3850 tape-cartridge system.

Helical scanning, as used in home TV recorders, is employed by the tape library shown in Fig. 2-2. The tape-cartridges are used to provide backup for disk storage. Each cartridge contains 700 feet of 2.7-inch-wide tape. Data are recorded in helical format (see Fig. 2-2) with each diagonal track containing 4096 bytes across the tape. The tape is moved in steps from track to track.

Such *automated-tape libraries* provide large, but slow to access on-line storage capacity. These are available both for conventional $\frac{1}{2}$-inch-tape reels and for the extra-wide-tape strip cartridges shown above. A mechanical picker mechanism selects the tape, carries it to the tape unit, and puts the reel on the tape unit so that it can be read conventionally.

2-1-3 Rotating Magnetic Storage

The primary type of storage devices to provide large quantities of easily accessible storage for a computer system is the *magnetic disk*. It, and related *direct-access devices*, dominate database storage. We have the choice of removable media (floppy disks and disk packs) and nonremovable media. The disks may be read and written either by a movable access mechanisms or, less commonly, by magnetic heads in fixed positions.

Many varieties of the basic design of a disk drive exist. Some of the alternatives give either higher performance or lower cost. In order to increase performance, there are disk units that have multiple moving heads and there are disks that have both fixed and moving heads.

In many modern devices, referred to as WINCHESTER DRIVES, the head and disk assembly is enclosed in an airtight and dust-free capsule. The clean environment permits closer tolerances and higher recording densities. Disk drives are available which can store 2500M characters per pack while in 1970 28M was the highest capacity available in a disk unit of the same physical size and design.

A generic name for all these devices is *direct-access storage drive* or DASD, but the term disk drive seems more convenient and adequate. Even though disk packs and disks are removable, it often happens that the disks containing an active database are never removed unless hardware maintenance is required.

FLOPPY DISKS Most popular of all rotating storage devices are the *floppy disks*. They are written and read by one (single-sided) or two heads (double-sided). The larger of these flexible disks are circles of 7.8 inch (19.8 cm) diameter, cut out of coated MYLAR material, and kept within an 8.0 inch square (20.8 cm) envelope. The 5.25 inch (13.3 cm) size is used on personal computers, but not as well standardized. Smaller sizes exist as well, and are common in portable computers.

Holes in the envelope permit access for spinning, writing and reading, and sensing the position of the disk. After the disk is inserted into the drive unit the disk may be spun via a pressure clutch. The disk is spun only while in use. The reading head touches the flexible disk during access, which permits a relatively high recording density but increases wear.

DISKPACK DRIVES In a *diskpack drive* a set of disks, numbering from 1 to 11, is packaged together in a stack. Multiple disks are separated by spacers from other disks in the stack. The disks are covered with a ferromagnetic oxide, similar to that deposited on tape, and these surfaces are hardened to withstand accidents and the heavy duty expected. The pack of disks can be placed on a spindle and each surface can be read by a recording head mounted on one of a set of arms that moves in or out, relative to the center of the disks, to find a specific track. The heads fly above the surface. The disks rotate continuously once started, and the heads can be switched to read or write information on the disk surfaces. Figure 2-4 presents a sketch of one platter mechanism of a two-platter disk drive. The entire unit has four heads, one for the top and one for the bottom surface of each disk. The drive listed in Table 2-1, a DEC RS-60 type unit, has a stack of ¡6¿ disks rotating together. The top and bottom surfaces of the stack are not used, so that ¡10¿ recording surfaces are available for data. On some types of disk drives one of the recording surfaces is used to provide timing or position information.

NON-REMOVABLE DISKS We use the term “non-removable disk” when the disk cannot be removed from the disk unit, other than for repair. These units may be equipped with moving arms, as removable disks are, or may have *fixed* heads.

288pt Figure space

was 2-3

Figure 2-4 Part of a HP-7900 two-platter disk drive. *(Courtesy of Hewlett-Packard Co.)*

Only non-removable disks will have fixed heads, since now closer alignment tolerances are possible. Units with fixed heads will have multiple heads, one per track, on each surface.

CYLINDERS In a moving head disk all recording heads of the disk drive move in or out together, but only one head will actually transfer data at any one time. No further physical movement of the arm is required to reach any one of the sets of tracks (¡10¿ in the RS-60) that are radially equidistant from the center but on different surfaces. For these tracks the time required to switch from track to track is only due to electronic switching delays and is negligible. This set of tracks forms a hypothetical cylindrical surface at right angles to the physical access directions. Since we often use such a set of tracks together, we will use the term *cylinder* to describe the tracks which we can use without motion of the access mechanism. A disk unit will have as many cylinders as one of its disk surfaces has tracks. Data written sequentially will fill all the tracks of one cylinder before the mechanism will be stepped to the next cylinder.

HEAD-PER-TRACK DEVICES In another type of fixed disk design, the disk units have actually one head per track, so that there is no access arm to be moved and hence no delay required to move the heads to a specific cylinder position. The heads will be staggered over the surface since tracks will be closer to each other than the minimum possible separation of the magnetizing elements within the magnetic heads. Geometrically differently arranged, but logically identical to head-per-track fixed disks, are *magnetic drums* where the heads are mounted facing a cylindrical magnetic surface which is rotated past them. This head arrangement is sketched in Fig. 2-5.

144pt Figure space

was 2-4

Figure 2-5 Diagram of a magnetic drum.

Head-per-track devices have very good access-time characteristics but provide relatively little storage capacity since moving-head devices will typically access more tracks. We will use the term magnetic drum to refer any head-per-track storage units, disregarding whether they have disk or drum shaped recording surfaces.

2-1-4 Large-Capacity Storage Devices

To provide mass storage at low cost, two approaches are followed:

1 Increased storage capacity per unit to reduce storage cost per character

2 Reduction of dependence on manual operation and mechanical parts to achieve more reliability and compactness

Magnetic disks are nearing their limits of storage capacity because of limits in the size of the magnetizable area, which is determined by the surface coating and by the physical dimensions of the read-write heads. Mechanical limitations are due to the positioning of head assemblies over tracks, even when feedback controls are used which align heads dynamically over the track being read. At the same time the great manufacturing experience with these disks makes their production costs low, and hard to match by other devices.

Increased storage capacity can be achieved by increasing the size of the units, but this adds mechanical delays to the storage systems. In order to increase data-storage capabilities without also increasing bulk, the space required to store each single bit of data has to be reduced.

OPTICAL TECHNOLOGY The use of light rather than magnetism to store data has the advantage that the required area per bit is much smaller than the area per bit required for magnetic recording. The availability of the technology from the entertainment industry is spurring the development of optical disc storage.

The discs used for audio and video entertainment are *read-only*. For data-processing we wish to be able to write data as well. Both *write-once* and *write-multiple-times* discs are under development, and write-once discs are entering the office-system market. The high capacity of optical discs means that write-once disks are often adequate.

A write-once disc has the potential advantage that all past *versions* of the data remain available, providing an excellent historical audit trail. The file organizations available from magnetic disks are not directly applicable to write-once optical discs; and this is slowing their acceptance. Write-multiple-times discs are likely to have a lower density, and hence less capacity. The applications will eventually determine the tradeoff to be made between capacity and retention of versions versus ease of reuse of storage area.

A bit is recorded on optical media by the heat of a laser changing a miniscule spot in the coating of the disc. To read the data, a beam of light at a lower level of energy is guided to the bit position. Changes in reflection will be recognized if a "1-bit" has been recorded. Semiconductor lasers of adequately high power make economical optical discs feasible.

The capacity of announced optical write-once discs varies from 300Mbytes to 100Gbytes, with 10Gbytes becoming a reasonable value for small units. Their access times are comparable to those of magnetic disks, but because of their greater capacity an exhaustive search over their content will become even more infeasible.

¡¡Space for optical disk figure, Ricoh? ¿¿

OTHER OPTICAL DEVICES Other optical recording devices have been manufactured using photosensitive material which is written directly by intense light. The chemical changes are then read using either transmission or reflection of light. Holographic techniques have been tried as well. The information is then distributed in an interference pattern over a large area. This approach seems attractive because of its ability to ignore local surface irregularities.

Some large storage devices which use photographic media and are based on microfiche technology have been used for some time. Their initial cost and the maintenance required for the automated developing process discourages widespread use.

2-1-5 Semiconductor Technology

The rapid development of integrated circuitry has made semiconductor technology attractive. We see applications that were based on secondary storage handled now completely in primary memory, and thus leaving the realm of databases. At the same time, new applications for secondary storage devices become feasible as their capacity increases.

SEMICONDUCTOR STORAGE Semiconductors can be organized in ways that differ from random access memory in order to serve specific database storage requirements economically. Very large arrays of storage cells may, for instance, be accessed only from the edges. The data will be shifted to the edges where amplifiers and connectors extract the data. Another alternative considered is to have the data processed within the arrays so that only results will have to be transferred out of the semiconductor chips. Economies can accrue because the space required for access paths to the data and for control of individual bits of storage can be minimized. One specific architecture of this type, *associative memory*, is presented in Chap. 12.

Semiconductor technology proposed for secondary storage includes common MOS memory, charge-coupled devices, and shift registers. All forms of electronic storage have the advantage that data transmission can be stopped and reinitiated without loss of data or time, whereas rotating devices incur delays when interrupted due to mechanical inertia.

A development which originated at BELL Telephone Laboratories is based on electrical control of small magnetic charge domains on ferrite surfaces. Under the influence of a magnetic field the surface charges, which are otherwise randomly distributed, will form electromagnetic *bubbles* a few thousandths of an inch in size. These can be rapidly moved along pathways by pulsed electric currents in conductors which have been deposited on these surfaces. These bubbles can be generated and detected by semiconductors. There are some operational similarities with disk drives, but the surfaces and the reading and writing mechanisms stay fixed while strings of data-carrying bubbles are moved along the surfaces between the sensing and the generating mechanisms.

MEMORY BASED FILES The reduction in cost of memory has made it possible to keep substantial files in memory for processing. Read access is no longer subject to mechanical delays. To achieve persistence of data a permanent copy is still kept on disk, and any updates are reflected on disk as well as in memory. For modest memory sizes reliability of persistence can be improved by battery power backup.

Searching through large data quantities in memory still requires time, but many more choices exist for effective memory-based-file organizations. We will not adress these in this book.

2-1-6 Memory

Essential for processing is memory. It contains the active, manipulable data structures used for computing. As shown in Fig. 1-7, all data moves through memory between persistent storage and the users with their programs Memory also contains the instructions of the programs while they are being used.

Memory is randomly addressable at a level of one or a few characters at a time. These units, *words*, are of limited size to allow the processor to manipulate all bits simultaneously through the use of parallel circuits. This type of memory also is relatively fast, so that it can serve to tie other devices, which operate at different rates, together into one data-processing system. Data-storage units which depend on mechanical motion have considerable inertia so that they are not started, stopped, accelerated, or decelerated to suit processing needs. They operate asynchronously, and when they have completed a data transfer to memory, other devices and processes can use the results. Data obtained from storage devices are kept temporarily in buffer areas allocated in core memory. We will discuss buffering in Sec. 2-3-¡4¿.

Memory is assumed to be volatile, i.e. removal of power on purpose or by accident will cause a loss of memory contents. Some memories can actually retain data and instructions, but most operating systems do not dare to count on retention of critical information.

VIRTUAL MEMORY On many large computer systems the range of addresses available to refer to memory is adequate for many file requirements. Typical addressing limits in large computers range from $5 \cdot 2^{18} = 1.280\mathsf{M}$ to $2^{32} = 4096\mathsf{M}$ characters. Only part of the memory space is real; the remainder is virtual and brought in by paging from backup storage when referenced, as reviewed in Sec. 1-4-2. The use of *virtual memory* to support files is analyzed in Sec. ¡8-5¿.

Virtual memory, although to the user undistinguishable in function from real memory, cannot be treated from a performance point-of view as if it were real. We find that the fundamental performance analyses developed here for general file organizations continue to hold, although the programmer will have little control over the actions of a file system based on virtual memory facilities.

2-1-7 The Cost of Storage

The performance of a system cannot be considered independently of its cost. We consider this aspect primarily in Chap. 13. In Sec. ¡13-3¿ storage costs are assessed. Sec. ¡13-4¿ considers the entire storage architecture, including the devices which connect the disks, etc., to the processors, are presented.

When the volume of data is large, infrequently used data may be kept *off-line*, on tapes or disks that are mounted by operating personnel when needed. In all other cases we will want to have the data available without manual intervention, i.e., *on-line*.

The cost of keeping data on-line shown in Table 2-1 consists of the cost of the storage medium, the cost of the device that can read and write the data, and the estimated cost of its connection to the computer's processing unit. The cost of keeping data off-line includes the cost of the materials themselves as well as the costs for properly conditioned space for one year. The values in the table combine all costs and are reduced to a cost per year using normal lease and maintenance rates. It is obvious that these values are only estimates.

In recent years hardware costs of storage devices and media have fallen by about 20% per year. The purchase cost for disk storage has decreased from $65/Mbyte in 1981 to $17 in 1985, and may be $5 in 1989. At the same time personnel and maintenance costs continue to rise. A design for a new system should include a formal projection of all of these costs for its lifetime.

Table 2-1 goes here with data for

Tape Floppy PC-hard disk RS-60 IBM3380 Optical disk Memory
IBM 3330 as well?

Table 2-1 Performance and Cost Parameters of Some Storage Devices

2-2 HARDWARE PARAMETERS

We now develop quantitative measures for the hardware used for file storage. This will provide parameters to use when evaluating alternate file organizations. The parameters used in this book are oriented mainly toward disk type devices, but equivalent parameters can be obtained for other types of storage hardware. We will capture the salient features of a wide variety of hardware and hardware oriented system choices with half a dozen parameters, summarized on page ¡63¿ in Table ¡2-5¿. Some assumptions about the operation and use of the devices are made to simplify those parameters. These have to be verified before building actual systems. Different types of storage hardware need different assumptions.

RANDOM-ACCESS TIME To access some item of data, located at some known storage position, requires operations which cause an access delay. This delay is called the *random-access time*. It is based on the assumptions that the position of the data to be accessed and the initial position of the access mechanism are randomly distributed over the disk. The assumption of randomness permits the estimation of the average time to reach a specific known position. The random-access time is the most critical parameter of storage device performance. The term is not particularly felicitous, since the access to data is certainly not random. The apparent randomness occurs because data are retrieved or stored in a sequence which is not determined by the device.

The random-access time is broken down into two constituents, *seek time* and *latency*. The former refers to the positioning delay, for instance, the movement of the head mechanism of a disk, and latency is the rotational delay incurred until the data can be read. The time required for the subsequent reading or writing depends on the size of the *block* of data being transferred and on the data-transfer rate of the hardware. So, before proceeding with an analysis of the time needed to get the data, we will present how the hardware organizes the data into blocks.

BLOCKS AND SECTORS As can be seen in Table 2-1, the capacity of a track can be quite large. Copying such a large quantity of data into memory can take a long time and much memory space. It is hence common to divide a track into a number of *blocks*. A block becomes the unit of data being transferred. The division of a track into blocks may be implemented completely by hardware; such hardware units are termed *sectors*. There will be a fixed number of sectors, typically 10 or 16 per track, each capable of holding a fixed number of characters. In other disk units a track can be divided into sectors by a software. Formatting software, provided with an operating system, writes markers onto the disk to define the block boundaries. These markers are recognized by the hardware during normal read or write operations.

This division of a track into fixed blocks by formatting is sometimes called *soft sectoring*, and the alternative devices are said to use *hard sectoring*. If the size of a hard sector is inconveniently small, it is best to routinely use a number of sectors together as one block. If soft sectoring is not supported by hardware, system software may yet divide data received from a track into smaller units, so that a reasonable block size is obtained. In any case the block, be it composed of a sector, a number of sectors, a formatted (or *soft*)-sector, a track, or a software-defined portion of a track, will be a convenient fixed size unit for data transfer.

A block then is a collection of coded characters or values, of a fixed size within a computer system or part of a system, which is moved as a unit between the storage devices and the memory for processing. Once the right block is found, it will be read completely in order to obtain the error checking-and-correction information for the block. A block is the prime hardware data unit considered throughout this book. The data units that programs manipulate, and submit to the file system for reading or writing, are called *records*. Section 2-2-4 will discuss the transformation of blocks to records.

BLOCK SIZE Most files and many databases use one single, predefined block size. Use of the same block size on different devices with differing track sizes can cause loss of storage space, but a careful selection of a block size which fits well into all devices can minimize this loss. Typical block sizes range from a a few hundred to ten-thousand bytes. The fixed block size reduces complexity in the device-oriented programs. The user programs will still have to deal with a variety of fixed and variable records.

The selection of the optimal size of a block depends on many factors, and in turn the size affects file system performance in a critical fashion. This chapter will discuss the factors which directly relate hardware performance to block size, but throughout the first half of the book the block-size parameter B will appear.

Records, the data units required for processing, vary in size from a few to thousands of characters. A large block size causes more irrelevant data to be moved around when a record is needed and requires more memory. A small block size means that more separate block-access requests have to be made to collect the data required for a task. The trade-off of these inefficiencies depends on the needs of the application and the design of the transaction programs. Section ¡7-2-4¿ presents an optimization analysis for the block size for indexes.

We can now discuss hardware performance in terms of accessing blocks within tracks. Tracks and cylinders are partitions of storage determined only by the design of disk unit hardware.

2-2-1 Seek Time

The *seek time* s is the time required to position the access mechanism over the proper track. Figure 2-6 shows the seek times required to move over various track distances for a moving-head disk, here an old model of the IBM 2314. For this device the relationship of distance traveled and time taken for this travel is not at all linear.

On many modern disks the seek times can be approximated by a linear relationship of the form $s_c + \delta i$, where s_c is an initial startup time, δ is the time to move between adjacent tracks, and i is the distance traveled in terms of intertrack spaces traversed. Most disk units are designed so that the time to move to the nearest track, $s_1 = s_c + \delta$, is less than the time for one revolution. This minimizes the delay for reading cylinders sequentially.

192pt Figure space

was fig.2-6

Figure 2-6 Seek times for an IBM 2314 model 1.

To avoid the pain of actually evaluating travel distances for individual disk accesses we will use an average seek time s whenever a seek is encountered. Often an average value for the seek time is provided by the manufacturer. It should be based on a uniform access distribution over all tracks. A lower average seek time can be obtained if important and highly active files are placed on a few adjoining cylinders. In such cases it can be useful to calculate the corresponding expected average seek time.

198pt Figure space

was fig.2-7

Figure 2-7 Travel possibilities for disk heads.

DERIVATION OF AN AVERAGE SEEK TIME To demonstrate the computation of the average seek time, the assumption of random distribution of track access will be used here, but other distributions can be handled similarly. In the case of a uniform random distribution, the likelihood of travel is equal from any cylinder to any cylinder. The distribution of distances to be traveled will not be equally likely.

Starting from the extreme cylinders, the heads may have to traverse all of the disk, whereas from the center track, they never have to travel more than half the distance. Figure 2-7 illustrates this effect. It shows for some of the 200 possible starting cylinders cyl the distances that the reading head may travel to the left `(l)` or to the right `(r)`.

A seek to the left of distance i, $i = 1 \ldots 199$, is possible from initial positions $cyl \geq i + 1$; and a seek to the right of distance i is possible from $cyl \leq 200 - i$.

For a disk with j cylinders, a distance i, left and right, can be traveled from positions cyl if $cyl \geq i + 1$ and also if $cyl \leq j - i$. No travel will occur if the starting point is equal to the destination cylinder.

We will compute the probability of each distance to be traveled by counting the event, left and right, for each cylinder position. There are a total of $j(j-1)$ actual travel combinations between the j cylinders, and we are assuming that they are all equally likely. We note that the probability of no travel is $pd_0 = 1/j$.

201pt Figure space

was fig.2-8

Figure 2-8 (a) Probability distribution of seek distances from random cylinder to random cylinder. (b) Probability distribution of random seek times.

For a distance i, $i = 1 \ldots j-1$, the probability of travel of this distance pd_i is

$$pd_i = \sum_{cyl=i+1}^{j} \frac{1^{\dagger}}{j(j-1)} + \sum_{cyl=1}^{j-i} \frac{1^{\dagger}}{j(j-1)} \quad = \quad 2\frac{j-i}{j(j-1)} \qquad i = 1 \ldots j-1 \qquad \text{2-1}$$

This probability distribution is shown in Fig. 2-8a. The product of these seek distance probabilities and the time required to travel any of these distances gives the expected seek time distribution. The expected average seek time s for this device, with the assumption of random access, is

$$s = \sum_{i=1}^{j-1} s_i \, pd_i \qquad \text{2-2}$$

where s_i is the seek time for distance i. A result is shown in Fig. 2-8b for the disk head travel times s_i presented in Fig. 2-6.

Devices with arbitrary seek delays can be evaluated using Eq. 2-2. The case of only using fewer, adjoining cylinders can be computed by limiting the range of i to the number of cylinders that are in use. A nonuniform travel distribution is handled by replacing the nominators(†) in Eq. 2-1 with functions of cyl and i which define the usage pattern for a particular situation.

201pt Figure space

was fig.2-8

Figure 2-8 (a) Probability distribution of seek distances from random cylinder to random cylinder. (b) Probability distribution of random seek times.

CONTROLLING SEEK TIMES The value of s for a file may be effectively reduced by certain allocation techniques. We indicated already that when a file occupies a limited number of consecutive cylinders the average movement will be less as long as there is no other file activity on the same unit. For a large file this can be aided by fragmenting the file over multiple disk units. *Fragmentation* will also provide overall access overlap between the two access mechanisms.

An alternating use of two files on the same disk will result in average seek times that are a function of the distance in terms of cylinders between these files. This could lead to substantially larger average seek times. In multiprogrammed and timeshared computer systems the probability of multiple use of a disk unit at any time appears large, especially when there are many users relative to the number of available disk drives. Transaction systems attempt to minimize this effect.

Whenever voluminous data can be accessed without interruption and in the same sequence that they are stored on the tracks, a seek needs to occur only once per cylinder.

This seek will also require only minimal motion. The cost of seeks per disk access will then be a fraction of the average seek time. We will quantify that effect in Sec. 2-2-5 as a factor in the *bulk transfer rate*.

After the desired track has been located by the seek, it is still necessary to determine where the head is relative to the actual block on the track, since that is the unit we wish to transfer. The begin point for the tracks or of the actual blocks on the disk will be sensed by the access mechanism. There will be a rotational delay before the data can be read or written.

2-2-2 Rotational Latency

After the reading and writing mechanism of a disk drive or similar mechanism is positioned at the correct track, a further delay is incurred to reach the desired block on the track. The various delays between the completion of the seek and the actual transfer of data are referred to as the *rotational latency* r of storage units.

One type of delay occurs because the reading of the recorded data cannot commence at an arbitrary point. This is because we cannot synchronize the single stream of bits coming from the disk with any desired point in memory, until some identification marker is found. Sometimes a track begin point must be reached first, and then it still is necessary to proceed to the desired block. In most devices with fixed sectors a sector count is maintained so that the next complete sector can be transferred, if it is the desired one. If the sector was just missed, most of a revolution will pass before data transfer will take place. In software-controlled formats a block-identifying area or count area is read to determine which of the blocks that pass by is the desired one. A track-begin identifying area (*home address*) is used only by formatting programs when a disk is changed to have new block sizes.

We will first consider the case where a block can be recognized at its own begin point, so that as soon as the desired block is below the reading heads, the data transfer can commence. Then the average value of the rotational latency r is one half of the time required for one rotation of the disk, or

$$r = \frac{1}{2}\frac{60 \cdot 1000}{\text{rpm}} \qquad 2\text{-}3$$

where rpm is the number of disk revolutions per minute, and the latency r is specified in milliseconds. Typical rotational speeds are 2400 and 3600 rpm, leading to values of r of 12.5 and 8.33 ms.

LATENCY WITH READING OF TRACK IDENTIFIERS Some disks have at the beginning of each track a field which permits verification that the positioning mechanism is at the desired track or cylinder. Then the above estimate has to be modified to account for the additional delay before data blocks can be transferred. The expected distance between the track begin point and the data is dependent on the number of blocks per track. The average time required to reach the track identifier is r, and the average distance from the track begin to the beginning of the blocks, given b blocks per track, is $\frac{1}{2}(b-1)/b$ of the track length. In terms of the previous value of r we find a new total latency:

$$r' = r + \frac{1}{2}\,\frac{b-1}{b}\,2r = r\left(2 - \frac{1}{b}\right) \qquad \text{2-4}$$

If there is only one block per track, the block is found at the beginning of the track and the latency remains r. For many and small blocks this value becomes nearly double the standard latency. This value of the latency is always applicable when writing into unpreformatted areas, since then no block recognition is possible. This occurs often in FORTRAN-based systems, since the standard for this language does not include file–definition capabilities, so that sector sizes are not defined prior to first use.

SEEK AND LATENCY COMBINED Sometimes the values provided by computer manufacturers have rotational delays factored into the seek times given. Such data, while being essentially honest, tend to blur the correct evaluation of specific system performance. Figure 2-9a portrays the combined effect of the two components, seek and rotational latency, for the IBM 2314 ¡¡Change¿¿ shown, with a one-block-per-track allocation. We expect a delay of r to find a block on the current cylinder and, since in this case a new record coincides with the beginning of a track, additional delays of entire revolutions due to seeks from other cylinders. The expected combined seek and rotational latency time provides only a few discrete choices, here r (no seek), $3r$, $5r$, $7r$, $9r$, or $11r$. By combining this fact again with the previous probability distribution for seek distances, we can obtain for a total expected access time an aggregate as shown in Figure 2-9b.

234pt Figure space

was Fig.2-9

Figure 2-9(a) Combined effect of rotational latency and seek. (b) Probability distribution of access times.

Certain disk units allow the seek process to continue automatically until a specified sector on the desired cylinder, included as part of the seek destination address, is found.

The published seek time then may include the average rotational latency. Such *sector addressing* can reduce significantly the load on the channel which connects the disk unit and the main processor. In this chapter we ignore this type of design. In Chap. 13, where the total load of an application on a computer system is discussed, we will encounter such disk control mechanisms as class 5 devices.

WRITE VERIFICATION To verify that information just written has been correctly recorded, an immediate reread may be performed after a write operation. A reread operation will carry out a comparison of the data from memory and the data just written and indicate mismatches. The operation causes the disk unit to be busy for one more revolution.

In current practice verification by rereading is performed infrequently. Errors on disks tend to be more often caused by poor handling of disks subsequent to the writing process than by writing failures or poor quality of the disks. We will ignore the effect of rereading in our analyses. Its effect can be easily inserted when rereading of written information is done.

2-2-3 Track Length and Track Capacity

The amount of data that can be read or written with one access determines the effectiveness of the random-access operation. If the entire track is used to hold a single block, then the *track length* is equal to the largest possible block. Otherwise the track length is equal to the product of the number and the size of the blocks provided per track. The length of a track may be given in terms of bits, characters, or words. The entries in Table 2-1 are standardized by using characters, generally of seven or eight bits in length. On many disk units with hard sectors multiple sectors may be read as one block. Blocks will always have to fit within a single track, so that the track length places an upper limit on the block size. If yet larger units of data are required by an application, multiple blocks have to be used.

INTER-BLOCK GAPS We recall that a track is frequently divided, by hard- or soft-formatting, into a number of blocks of length B each. At every break between two blocks there is a *gap* to permit the head mechanism to prepare for the next operation. The space required for this interblock gap, G, reduces the actual storage capacity, as is shown in Table 2-2. With high recording densities the gaps displace hundreds of characters. Small block sizes increase the number of gaps, causing significant amounts of wasted disk or tape storage. Large block sizes do make heavy demands on memory-capacity and on transmission capability while transferring unused, but adjacent, data.

Gap size The size of gaps between blocks of tape or disk depend on the devices used. The older six-data-track tape units used 0.75 in (19 mm) and at their highest recording density, 800 bpi, had a value of G of 600 characters. Eight track tapes require 0.6 in (15 mm), or 960 characters at 1600 bpi; and the 6250 bpi tapes have a gap of only 0.3 in (7.6 mm), but this leads to a gap G of 1875 characters.

Disks units can have complex gaps. The gap on an IBM 3330-type disk consists of the equivalent of

1 Space equivalent to about 135 characters

2 A variable-length block-identification field, which may be considered useful space

3 A space between this identification field and the actual block of about 58 characters in length. This space allows for switching from reading – for the identification – to writing of data.

The total gap size G is hence about 200 characters for this device; more values appear in Table 2-1. The block-identification field is used by software to keep a name for the block.

Capacity for	Characters per block B				
Device	1†	80	200	1000	3000
Tape 800/6	37.9K	2.67M	7.58M	14.2 M	18.9 M
Tape 1600/8	47.4K	3.25M	10.1 M	22.3 M	33.7 M
Tape 6250/8	94.5K	6.75M	22.1 M	60.5 M	107.0 M
Floppy disk (8 in.)	45.6K	1.06M	1.29M	1.386M	1.386M
Disk 6 (RS-60)	¡ 118K	4.16M	5.40M		¿
Disk 11(3330)	730K	37.1 M	68.4 M	83.6 M	91.2 M
Disk giant(3380)	79.6 M	2 119.75M	2 272.68M	2 442.6 M	2 389.5 M

† Blocks of one character length are hypothetical for many devices. A minimum number of characters is often required to generate proper redundancy check information. The USASI standard for tape specifies a minimum block size of 18 characters for information interchange.

Table 2-2 Actual storage capacity of tapes and disks for various formatted block sizes in terms of total characters

BLOCK SIZE AND TRACK CAPACITY The reduction in storage capacity shown in Table 2-2 is caused mainly by the gaps between the blocks, and to some extent by the loss when fitting these blocks into the track dimensions of the disks. In order to reduce the loss associated with a poor fit of blocks into the track size — as seen in the last column for disks — block sizes which make optimal use of the track capacity for the device may be used. An unusual block size will, however, create problems when different devices are used within one system and whenever storage units are replaced by newer types. The desire to use identical block sizes on different devices will cause a compromise to be made. Optimizing space utilization for a variety of storage devices leads to small block sizes. Systems with a single block size for all devices may use blocks of only 128 or 256 characters.

BLOCK POINTERS In order to refer to a block, we have to construct an address identifying the device and the position of the block on the device. Such a reference address is known as a *block pointer*. A block pointer provides a unique name for every block in a system. We will use block pointers in the remainder of this section but will discuss the implementation of block pointers in Sec. 2-3-3. The block identification field seen on soft-sectored disks is used by file-system software to keep a name for the block, and hence is a form of block pointer. The size of block pointers will be denoted by the symbol P.

2-2-4 Records and Blocking

Records are the actual units for data storage on the logical or file level. The fitting of records into blocks is referred to as *blocking*. Records may be of a fixed size, or may be variable in length, as required by the application. We denote the size of a record by R.

Since an evaluation of files for an application has to be concerned with performance in terms of records rather than in terms of blocks, we will now analyze blocking and derive some parameters which relate hardware and block based parameters to records. The major parameter is the *blocking factor*, denoted by Bfr, which gives the number of records expected within a block. Three important methods of blocking are discussed next. When the methods require that all records fit within blocks, we call them *unspanned*. Unspanned blocking applies to fixed- and to variable-length records.

383pt Figure space

was fig.2-10

Figure 2-10 Blocking methods.

Fixed blocking An integer number of fixed-length records is placed within a block as shown in Fig. 2-10. We observe that for unspanned fixed blocking the blocking factor *Bfr* is an integer constant. Any unusable space in a block is wasted.

$$Bfr = \lfloor B/R \rfloor \qquad \langle \text{fixed}\dagger \rangle \; 2\text{-}5$$

Variable-length spanned blocking Records are packed into sequential blocks, and are broken at block boundaries. The continuation is indicated by a pointer to

† We use ⟨condition⟩ to distinguish cases where a variable is derived in more than one equation.

the successor block. *Spanning* is difficult to implement. Records that actually span two blocks are expensive to search and the resultant files are difficult to update.

Variable-length unspanned blocking   Only entire records are placed within a block. There is wasted space in most blocks because of the inability to use the remainder of a block if the next record is larger than the remaining unused space. The number of records per block varies if the records are of varying lengths. Records cannot be longer than the blocksize.

The value of *Bfr* for both types of variable-length blocking will be also affected by the marking scheme used to separate records, as discussed below.

RECORD MARKS   When manipulating records, it is necessary to know where records begin and end within the blocks. For fixed blocking, only the (constant) record length R has to be known to to locate records within a block. When records of variable length are packed into blocks, data for marking the record boundaries within the block has to be added to separate the records. When spanned records bridge block boundaries, some reference to the successor block is also needed.

The external specification of the file format, obtained at the time of execution of an `OPEN` statement, will give the blocking type and marking convention in use. For fixed blocking only the record length R is given. For variable length records three techniques, shown in Fig. 2-11, are available to mark the records.

The blocking descriptor may be a separator marker between the records, which can be recognized when searching through a block. Such a marker has to be unique so that data within the record cannot be mistaken for an end-of-record marker. When data are stored in the form of character strings, standard end-of-record characters are available to delineate records within blocks. These character codes are often derived from communication technology; examples are `CR`, `GS`, `RS`, and `US` shown in Fig. 11-1 ¡¡new, but verify¿¿.

99pt Figure space

was fig.2-11

Figure 2-11   Marking of variable-length records.

Another method used to mark the record positions within blocks is the use of a length indicator preceding every record. The beginning point of the next record can be reached by skipping over the body of the current record. The third method uses a table within each block giving all the record positions in a block.

We can now compute the blocking factor *Bfr* for variable-length blocking. Each record requires one marker entry, and we assume that the size of one marker entry is about equal to the size of a block pointer P. For spanned blocking a block pointer of size P to its successor block may be included in each block, so that the pieces of a spanned record can be easily retrieved. Then

$$Bfr = \frac{B - P}{R + P} \qquad \langle\text{var-spanned}\rangle\ 2\text{-}6$$

With unspanned variable-length blocking an average of $\frac{1}{2}R$ will be wasted because of the fitting problem, but no successor pointer is required. Here

$$Bfr = \frac{B - \frac{1}{2}R}{R + P} \qquad \langle\text{var-unspanned}\rangle\ 2\text{-}7$$

WASTE Because of the gaps between the blocks, unused space within blocks, and various markers, not all the storage space provided by the devices can be used. Since the effect is often significant, we calculate now the *waste per record* W due to these factors.

The waste due to gaps W_G will equal the gap size G per block divided by the number of records per block, or *blocking factor* Bfr:

$$W_G = G/Bfr \qquad 2\text{-}8$$

There is also waste due to unused space from blocking in each block. This is also allocated to each record as W_R. With fixed blocking the record sizes are generally set to minimize the amount of waste, and the wasted space per block is certainly less than one record R. The bounds on W_R are hence $0 \leq W_R < R/Bfr$. We find fixed blocking frequently when records are small, and we recall from Sec. 2-2-3 that gaps are often large. For fixed blocking the waste per record is then

$$W = W_G + W_R \qquad \text{often} \qquad W \approx W_G = G/Bfr \qquad \langle\text{fixed}\rangle\ 2\text{-}9$$

We will now consider the cases of variable length record blocking. The waste due to record fitting and marking has already been discussed in the evaluation of the blocking factor Bfr. For spanned blocking all the space in a block is usable and the total waste per record is

$$W = P + (P + G)/Bfr \qquad \langle\text{var-spanned}\rangle\ 2\text{-}10$$

and for unspanned blocking we expected a half record loss per block, so that

$$W = P + \frac{\frac{1}{2}R + G}{Bfr} \qquad \langle\text{var-unspanned}\rangle\ 2\text{-}11$$

Since Bfr is also a function of R, we note that in this last blocking method the waste increases quadratically or $O(R^2)$ with the record size. With a small value for P, and with Eq. 2-7 substituted, $W \approx \frac{1}{2}R^2/B + R\,G/B$. The waste due to fitting is now quite significant when records are relatively long relative to the block size. This factor has to be taken into account when estimating total storage capacity and effectiveness of data transfer. In order to arrive at an accurate estimate in critical situations, the actual distribution of record sizes should be considered. A discussion of tools to perform such analyses is presented in Chapter 10.

We recall that the size of a record, when spanning is not supported, is restricted to the block size. If this size is not adequate for an application, the problem of proper blocking is passed on to the next higher level of the system. Such inadequacies of the lower-level file systems make other parts of the system more complex and violate the modularity concepts discussed in Sec. 1-4-3.

2-2-5 Transfer Rate

We have up to this point discussed the two constituents of the random access time, the seek time and the rotational latency. When the proper track and rotational position is reached, the actual data block still has to be read or written from or to the disk. This third constituent of a data-transfer operation depends on the speed of actual data transfer. The rate or speed with which data can be transferred is known as the *transfer rate* t. It is measured throughout this book in terms of characters per millisecond or, equivalently, Kcharacters/s or also Kbytes/s.

The basic transfer rate is dependent on the design of the device used. On most devices the read and write rates are identical, although exceptions exist for mechanical and some optical recording schemes. On disks the transfer rate is a function of rotational speed and recording density. Manufacturers often provide a raw transfer rate in terms of bits/second, which then has to be adjusted for the number of bits required to encode and check a block of 1000 characters.

The time to read a record of R characters is

$$T_R = R/t \qquad \text{ms} \tag{2-12}$$

and the time required to transfer an entire block is equal to B/t. Since the transfer of a block is such a frequent operation, we will write for the *block transfer time*

$$btt = B/t \tag{2-13}$$

Whereas the transfer rate is determined by the device, the block transfer time is also determined by the block size B. Since blocks can never be larger than tracks, we note that $btt \leq 2r$. Table 2-1 provides some typical values of btt.

BULK TRANSFER RATE The transfer rate t given in the literature provided by a manufacturer is the instantaneous rate during the actual transfer. When transferring large quantities of data sequentially, there are intervals when gaps and other nondata areas are being passed. At the end of each cylinder a seek will occur and during the seek time no data are transferred. In the case of continuous sequential reading or writing, the use of a *bulk transfer rate* simplifies performance analyses. We will now quantify the two factors which affect bulk transfer, and then combine them into a new parameter, the bulk transfer rate t'.

EFFECT OF GAPS AND BLOCKING ON THE TRANSFER RATE The effect of unused space due to gaps and due to blocking of records into blocks was expressed in Eqs. 2-9 to 2-11 as wasted space. We now perform a similar estimation for the transfer time. We ignore any additional gaps that some devices have at the beginning of a track and include only the gaps between blocks, as in computing W. The effect of waste on the bulk transfer rate can be evaluated by considering an entire cylinder, with k tracks and nrt records per track. Reading of an entire cylinder takes a period of $k\,nrt(R+W)/t$ ms, and during that time $k\,nrt\,R$ data characters are transferred. The actual transfer rate for a entire cylinder of data is now computable in terms of W as

$$t_{cyl} = \frac{k\,nrt\,R}{k\,nrt(R+W)/t} = \frac{R}{(R+W)/t} = t\frac{R}{R+W} \qquad \text{characters/ms} \qquad \text{2-14}$$

The value of W for various types of blocking is provided by Eqs. 2-9 to 2-11.

EFFECT OF SEEKS ON THE TRANSFER RATE In reading even larger quantities of data sequentially, we will have to cross over cylinder boundaries. One minimal seek is required once per cylinder. Even though seeks may be infrequent, the required correction may still be far from negligible because of the relatively long seek times on many units. On disks with few surfaces, seeks will be needed frequently.

In order to account for the effect of seeks, we will consider the seek frequency and seek time, and use the combination to predict an effective seek time per record s'. The assessment requires some understanding of the operating environment. In a multiprogrammed environment, competition may exist for the seek mechanism, so that a full seek time delay s may be incurred frequently, up to once per block. The effective delay per record is then $s' = s/Bfr$. But when interference is modest or can be controlled during a transaction, the expected delay beyween blocks is much less. In the case where there is no interference at all, the seek delay will occur only at the end of a cylinder. We use the number of surfaces k to determine the size of a cylinder in terms of tracks and then apply the seek delay s_1. To evaluate seeks per record, the number of records per track nrt is also needed; the entire cylinder contains $k\,nrt$ records.

In general a seek to the next track (s_1) requires less than the time of one revolution ($2r$). The desire to keep $s_1 < 2r$ caused in fact the mechanical complexity which led to the complex seek-time curve shown in Fig. 2-6. Since we continue the reading of records at the beginning of the next track of the new cylinder, we have, if $s_1 < 2r$, a seek delay per cylinder of $2r$. One revolution can again be expressed in terms of the time to transfer all records from a track, so that $2r = nrt(R+W)/t$. Combining these terms we find the minimal effect, from a track-switch delay occurring only at cylinder boundaries, to be $s' = s_1/(k\,nrt) \le 2r/(k\,nrt) = ((R+W)/t)/k$ per record.

The seek overhead per record for continuous reading s' is now bound by

$$\frac{1}{k}\frac{R+W}{t} \le s' \le \frac{s}{Bfr} \qquad \langle\text{bounds}\rangle\ \text{2-15}$$

The range of these bounds is great, easily a factor 100. As a practical estimate the seek time overhead per record, while reading sequentially, may be taken to be equal to the minimum value ($2r = nrt(R+W)/t$), but occurring at every track boundary, so that

$$s' \approx \frac{2r}{nrt} = \frac{(R+W)}{t} \qquad \langle\text{estimate}\rangle\ \text{2-16}$$

Measurements indicate that this estimate is still conservative and that in most computing systems interference is modest. If this factor is of great importance to the eventual success of the system, measurements of sequential read performance should be made on the system to be used, to assure that the estimate is met.

Any assumptions of this type, as having expectations of less than average seek times and seek frequencies less than once per block, should be well documented in any evaluation, so that effects of changes in the operating environment will not lead to unexpected and disastrous performance degradations.

CALCULATION OF THE BULK TRANSFER RATE The bulk tranfer rate includes the effects of waste W and seeks s'. We obtained in Eq. 2-14 above the bulk transfer rate due to waste within one cylinder. For multiple cylinders we add the seek delay per record s' to the transfer time per record and its waste $(R+W)/t$, and obtain the bulk transfer rate

$$t' = \frac{R}{(R+W)/t + s'} \qquad \text{characters/ms} \qquad 2\text{-}17$$

where s' is chosen to reflect the operating environment as detailed in the reasoning leading to Eqs. 2-15 and 2-16.

To combine the two factors for the case of no interference, we consider the time to read an entire cylinder with k tracks and nrt records per track. During this time $k\,nrt\,R$ characters are transferred, and during this time the access mechanism passes each of the k tracks. The lower bound in Eq. 2-15 is based on the loss of an additional revolution once per cylinder for a seek and causes an effect per record of $s' = ((R+W)/t)/k$ or per cylinder of $nrt(R+W)/t$. The time per cylinder is the sum $T_{cyl} = k\,nrt(R+W)/t \;+\; nrt(R+W)/t = (k+1)nrt(R+W)/t$. The bulk transfer rate for the entire cylinder is hence

$$t' = \frac{k\,nrt\,R}{T_{cyl}} = t\,\frac{k}{k+1}\,\frac{R}{R+W} \qquad \langle\text{no interference}\rangle\; 2\text{-}18$$

A similar reasoning can provide the bulk transfer rate for the estimated value of the seek effect given in Eq. 2-16. Now $T_{cyl} = k\,nrt(R+W)/t + k\,nrt(R+W)/t = 2k\,nrt(R+W)/t$ and

$$t' = \frac{k\,nrt\,R}{T_{cyl}} = \frac{t}{2}\,\frac{R}{R+W} \qquad \langle\text{estimate}\rangle\; 2\text{-}19$$

We can compare this estimate with the value of t_{cyl} given in Eq. 2-14 and note that the seek interference halves the estimate of the bulk transfer rate within a cylinder, and doubles the time for sequential data-transfer operations. Factors in this estimate involved the interference per block and the seek distances expected. Since larger blocks take a longer time to transfer, a change in block size will not affect the estimate directly, although the total seek delays may be less.

Transfer of data at the bulk transfer rate, as computed above, can of course be obtained only if we do not have to delay the reading in order to perform major computational tasks. We will describe in Sec. 2-3-4 the conditions for computational interference required to make this transfer rate valid. In this section we only considered the access interference by other, concurrent, tasks. In either case a system environment which creates significant interference will cause delays in the transfer operations, and such delays will in turn increase the opportunity for interference. The effect of interference on the bulk transfer rate quantifies one of the reasons for operating in a transaction-oriented operating system, as described in Sec. 1-4-3.

USE OF THE BULK TRANSFER RATE The bulk transfer rate accounts for all additional factors when reading through sequences of records. It has been obtained for the actual data record size R, so that it can be applied to the units of concern to the user programs. When we apply the bulk transfer rate to entire blocks, only the actual size of the data in a block $Bfr\,R$ has to be taken into account, since the waste due to gaps and record fitting has been accounted for in the formula for t'. Whenever the instantaneous transfer rate t is used, gaps have to be accounted for when reading from block to block.

If a single record is to be read from a random file position, an entire block of size B has to be transferred from the file. The single block is read, following a seek and latency to arrive at the block, with a transfer rate t. Gaps between blocks can be ignored. In summary:

Sequential accessing of records Use the bulk transfer rate t' applied to the record size R.

Sequential accessing of blocks Use the bulk transfer rate t' and the actual data quantity per block $Bfr\,R$ or, if neccessary, the transfer rate t and all of $B + G$ to compute the processing time per block.

Random accessing of records or blocks Since an entire block has to be read use the transfer rate t and the block size B with the random access time $s + r$ to compute the retieval time, giving $s + r + B/t = s + r + btt$.

When blocking, the fitting of records into blocks, generates little waste, we can ignore the difference between the net ($Bfr\,R$) and actual (B) block sizes. The use of B where $Bfr\,R$ would be more appropriate will lead to more conservative values when estimating performance. There is little waste with spanned blocking or when fixed records fit precisely into the blocks. In those cases, the number of records per block, the blocking factor Bfr, can always be taken to be equal to the block size divided by the record size (B/R).

2-3 BLOCKS AND BUFFERS

We already have developed the concept of a block in the preceding section and will only review the definitions before proceeding with a number of subjects related to the management of blocks.

2-3-1 Blocks

A *block* is the unit of information actually transferred between the external storage devices and a working area, the *buffer*, in the memory of the computer. The requirements imposed on buffer management to achieve good file performance will be one of the subjects to be discussed.

We presented in Fig. 2-10 three basic methods of placing records in a block:

1 Fixed blocking
2 Variable-spanned blocking
3 Variable-unspanned blocking

The blocking method used effects mainly the quantity W, the wasted space per record. If, in subsequent chapters, the blocking method is not noted, use of the variable unspanned method can be assumed for dealing with variable-length records and fixed blocking would be used where we are limited to fixed-length records.

MIXED BLOCKING TECHNIQUES Sometimes blocking methods are not kept consistent throughout a file. An example of such an inconsistency exists in some file systems in regard to updating. To avoid complex update programs, new records are written unspanned into a file which is otherwise spanned, or blocking may be avoided altogether when updating. A traditional file access method, IBM-ISAM, uses this technique. The analysis of such files becomes more complex than the methods shown in the examples in the next chapters.

A mixed blocking strategy also can be employed for other reasons. In an operating system that provides a paged, virtual memory environment, it can be desirable to use pages as the basic unit of data transfer. Since the size of records, when using unspanned blocking, is limited to the size of a block, multiple pages may actually be combined to create larger blocks for file purposes. This approach used by IBM-VSAM, where a *train* of pages is the basic working unit (see Fig. 2-12). Within the train, records may span pages, but between trains no spanning takes place. Since the entire train is read or written together the entire train acts like a single block. The increased transfer time can cause more interference, even when the pages are maintained in sequence. Multiple gaps appear now within such a block, so that the effective waste will be greater than for a block composed of a single unit. Using B now to denote the size of a train having nPt of pages leads to a blocking factor and wasted space per record

$$Bfr = \frac{B - nPt\,P - \frac{1}{2}R}{R + P} \qquad \langle\text{VSAM train}\rangle\ 2\text{-}20$$

$$W = \frac{nPt(G + P) + \frac{1}{2}R}{Bfr} + P \qquad \langle\text{VSAM train}\rangle\ 2\text{-}21$$

Other aspects of this file-access method will be discussed in a detailed example in Sec. ¡6-3¿ (also see Fig. ¡6-15¿).

144pt Figure space

was Fig.2-12

Figure 2-12 Train arrangement of multiple pages per unspanned block.

2-3-2 Density and Locality

Other considerations which can affect blocking decisions are related to the growth of a file. Updating of information in a file may require that new records have to be inserted into the file or that longer records replace earlier, shorter records. The various types of file organization discussed in Chap. 3 all have their own method of handling insertions. Some of these methods will extend blocks by linking new blocks to the old ones using block pointers. Since the successor blocks are assigned at later times, they will not be accessible by sequential reading but require access to another point of the disk, and additional seek and rotational latency time.

LOADING DENSITY If we expect many update operations, we may actually leave some free space distributed within the file for future needs. Each block that is assigned when a file is created or enlarged will initially have some unused space. The fraction of space initially utilized is called the *loading density*. If sufficient free space is available within a block, data can be added without an interspersed random access. When all the free space has been used, a new block has to be fetched and linked into the file, but this block will also have space for several more updates. A disadvantage of a low loading density is, of course, that more blocks have to be read to get the same amount of data when the file is not yet full.

EQUILIBRIUM DENSITY A system where the records themselves grow and shrink is forced to move and reallocate records within the blocks. Depending on the design of the file structure, some or much of the space for the file may become fragmented and unusable. Space no longer used can be allocated for reuse to retain the benefits of a low loading ratio. The density expected after a long period of operation is the *equilibrium density*.

LOCALITY A record will be obtained with least delay if it is placed close to its predecessor. When a series of records has to be fetched, the clustering of the series is the most important factor in performance. A similar consideration is encountered in paging systems where it is desirable that all memory references be located within a small number of pages. If this is achieved, there is strong *locality*. If serial references are far apart so that it is costly to get the next record, the locality is weak. Locality as applicable to records in a file is categorized in Table 2-3.

Strong locality

Record is in the same block and the block is available in memory.
Record is in the next available block on the same cylinder.
Record is on the same cylinder.
Record is on a current cylinder of another device.
Record is on adjoining cylinders.
Record is on a known cylinder.
Record position is unknown, computed using data in memory.
Record position is unknown, found by reading an auxiliary file.
Record is on a remote computer in a distributed network.
Record is on a device not currently on-line.

Weak locality

Table 2-3 Locality

If the data that are to be used during some transaction exhibit strong locality, we say that we are dealing with a clustered data structure. *Clustering* applies to data in one or more files, and we will encounter this term mainly in the design of databases which include many files.

In a system where there is a variety of devices, the considerations which determine strength of locality will become more complex. We will not attempt to define the strength of locality as a single, quantitative term, but we do use locality as a useful concept in file design.

2-3-3 Block Pointers

The *block pointers* that have been used to link blocks to each other require some more elaboration. We use block pointers to address a specific data field in secondary storage.

PHYSICAL DISK ADDRESSES To refer to a unit of data on a disk, a *physical aTake heath ddress* will have to specify up to six segments:

1 The number of the physical device
2 The number of the cylinder
3 The number of the surface
4 The sector or block number
5 The record number within the block
6 The field or character number within a record

The complete physical address is composed of a sequence of such segments. Use of such an address as a block pointer is both unwieldy and inadequate. Integer arithmetic applied to a segmented address will lead to erroneous values.

Another problem with a segmented address occurs because different types of physical units in a computer system will have a different number of cylinders, surfaces, and blocks. This means that multiple address formats have to be manipulated within one system.

A third problem occurs when diskpacks are exchanged on a physical device. In that case a physical address does not correspond to a specific item of data.

A final problem is that record sizes and field sizes are application dependent, so that the maximum number of these components in terms of the next higher components is not fixed. In most systems a *block* is the smallest fixed unit under control of the operating system.

RELATIVE ADDRESSES An alternative to physical addressing is the use of a *relative address* over the entire file domain. Relative block, record, and character addresses are all used in practice. A relative address is an integer ranging in value from zero (or one) to the maximum number of blocks within the domain of the system which controls the storage of data. Figure 2-13 shows a sample system. There will be a unique translation of a relative address to a physical address and vice versa, which allows access to the physical devices to be carried out by operating system routines when a relative address is given. The application of such an algorithm is shown in Table 2-4.

SYMBOLIC ADDRESSES It is also possible to assign a *symbolic address* or *block identifier* to every block or record. There is now no computable relationship between a symbolic address and its value. An address table will provide the physical or relative address for every block in use, and a look-up procedure is executed to obtain the block address whenever a block is requested.

The block identifiers can be integers which select the table entry, or can use key-to-address transformation techniques as *hashing*. Such methods are commonly used for the linkage of records in the ring-file structures presented in Sec. 5-¡2¿.

The use of an address table provides a flexible assignment of blocks to storage. Blocks or records can be moved, and the address table changed appropriately, so that references to the data which use the symbolic identifier remain valid. The address table requires memory and additional accesses to storage when it gets larger than its memory allocation.

USE OF POINTERS We described three techniques suitable to refer to blocks or records of files. We will use the term *pointer* for any value used to reference a block or record. Any one of three pointer types:

1 Symbolic block or record identifier
2 Relative block or record address
3 Physical hardware address

is adequate to identify a block or a record uniquely. Whenever a specific record is wanted, a pointer is provided to the file programs which fetch the data from storage. The alternative is to retrieve data in physical sequence. Then a pointer may be provided as an additional output to identify the record which was retrieved. When data are stored, a pointer is also provided for subsequent retrieval.

Programs which are at a level closer to the hardware tend to use addresses more; higher-level software will use symbolic references more. A single level of software should, for consistency, use only one pointer type throughout. In the illustrations we show mainly relative addresses.

243pt Figure space

was fig.2-13

Figure 2-13 Relative block addressing.

To refer from one record to another record, a field of the referencing record will be used to hold a pointer. The field size for a pointer P has to be large enough to hold a value adequate to address all possible records or blocks; in the example of Table 2-4 this requires six digits. A pointer field which is empty is denoted by Λ.

When the pointers associated with addressing techniques are used within a file to represent cross references, the resulting structure becomes complex and has to be managed with care. If blocks or records are not moved freely, relative addresses allow efficient use of pointer fields and accessing of data without requiring the look-up of a symbolic identifier. We can consider that the files have been bound with differing intensity or strength, depending on which pointer type is used.

2-3-4 Buffers

The area into which a block from the file is read is termed a *buffer*. The management of buffers has the objective of maximizing the performance or the utilization of the secondary storage systems, while at the same time keeping the demand on CPU resources tolerably low. The use of two or more buffers for a file allows the transfer of data to be overlapped with the processing of data.

Equipment in the domain of the file system

Number	Type	Cylinders per unit	Tracks per cylinder	Blocks per track	*Total*
2	Head per track disks	1	128	16	4 096
2.5	Permanent disks	400	20	8	160 000
1	Disk drive with choice of 3 packs	200	20	4	48 000
					212 096

Relative Block Address ($RBA = 0$ to $212\,095$) allocated to the devices as follows:

Allocated RBA Range		Device	Hardware address computation
$RBAbeg$	$RBAend$	type no.	
0 to	2 047	Head per track disk 1	track=$\lfloor RBA/16 \rfloor$, block=RBA mod 16
2 048 to	4 095	Head per track disk 2	track=$\lfloor (RBA - RBAbeg)/16 \rfloor$, block=$(RBA - RBAbeg)$ mod 16
4 096 to	68 095	Disk P1	cylinder=$\lfloor (RBA - RBAbeg)/(20 \cdot 8) \rfloor$, track=$\lfloor ((RBA - RBAbeg) \bmod (20 \cdot 8))/8 \rfloor$, block=$(RBA - RBAbeg)$ mod 8
68 096 to	132 095	Disk P2	etc.
132 096 to	164 095	Disk P3	
164 096 to	180 095	Disk D, pack A	cylinder=$\lfloor (RBA - RBAbeg)/(20 \cdot 4) \rfloor$, track=$\lfloor ((RBA - RBAbeg) \bmod (20 \cdot 4))/4 \rfloor$, block=$(RBA - RBAbeg)$ mod 4
180 096 to	196 095	Disk D, pack B	etc.
196 096 to	212 095	Disk D, pack C	etc.

Table 2-4 Relative Block Addressing

BUFFER REQUIREMENTS Buffers can occupy a large fraction of the memory resources of a system. Let us look at a simple medium-sized computer system where 30 users on terminals can manipulate files. A typical data-processing operation may involve three files. For each file one may use one or two buffers. Then 90 or 180 blocks will be occupying memory buffers. If each block has a length of 1000 characters, 180 000 characters of the memory are used for buffers, while the total memory size may be 250K to 1M characters. Even in systems where memory is backed up by paging storage on disk or drums, those pages which are involved in data transfer of files have to be kept resident in memory. The demand made by a file system on memory for buffers is hence an important part of the resource usage of a file system. The high cost of memory relative to disk storage, as shown in Table 2-1, makes good buffer management very important.

BUFFER MANAGEMENT In order to optimize the allocation of limited memory buffers, a buffer-scheduling algorithm assigns buffer spaces to users as needed. As shown in Fig. 2-14, blocks requested more than once will use the same buffer. Any memory not currently in use by other processes may be made available for the buffer manager; but when the requests exceed the available space, the buffer-scheduling process may have to delay a user process or deallocate buffer space of a lower priority process. Deallocation may also be done on the basis of low usage.

If buffer management is to be performed in a paged multiprogramming environment for the entire system, a separate buffer-managing program can be omitted. The paging program will have to take account of the need to keep active buffers resident. In either case, separate management or paged management, it will be useful to match the sizes and boundaries of the buffers, used by the file system, and the pages, used by the operating system, in order to minimize fragmentation of memory.

```
Q: Check if the block is already in memory using a table look-up.
   If yes, then give the process a memory pointer to the
           buffer which contains the block.
   If no, then allocate a buffer,
           initiate reading of the block into the buffer, and delay
           the process by putting it into an operating system queue.
           The process is restarted at a suitable time,
           beginning at point Q.
```

Figure 2-14 Block Request Processing by a Buffer Management Program

If only one record out of a block is needed, the remainder of the buffer space could be immediately released for other uses. Frequently it is beneficial to keep the contents of the entire block available. This will allow updating of the record and subsequent rewriting of the block with the surrounding records. If the remainder of the block has not been kept available, the block has to be read again before rewriting can take place. Also, if the process at the next request wishes to read a successor record, retention of the block can avoid the rereading from file which improves file-system performance.

Throughout the following evaluations we keep at least two buffers available in memory, as well as directory and other critical information from other blocks which have been read previously. If the algorithms require more buffer space, the descriptions will state this fact explicitly. Two buffers, when used correctly, provide high performance in a file system when reading sequentially.

DOUBLE BUFFERING The availability of two buffers permits reading into one buffer while processing the previous buffer contents. This is a prerequisite for the use of the bulk transfer rate t' for sequential reading; Fig. 2-15 illustrates the algorithm in detail. It is clear that the computational time required for processing one buffer must be less than the time used by the disk units to fill the other one. This algorithm is hence valid as long as the computational time required to process the records of one block $c_{block} = c\,Bfr$ is relatively small in comparision with the block transfer time.

Condition for continuous sequential processing

$$c\,Bfr < btt \qquad \text{or} \qquad c\,Bfr < \frac{B+G}{t} \qquad \text{or} \qquad c < \frac{R+W}{t} \qquad \text{2-22}$$

The gaps between the blocks provide some leeway, (G/t), but it should be realized that the sending of the process wake-up message and buffer switching also requires time. This time is the *overhead* caused by buffering. We prefer hence the safer condition, $c\,Bfr < btt$.

288pt Figure space

was Fig.2-14

Figure 2-15 Use of two buffers for sequential reading.

The areas indicated as "Wait" are the periods, $btt - c\,Bfr$, that the CPU is idle relative to this computation. Operating systems which provides multiprogramming will attempt to allocate this time to other processes. In order to terminate the entire process, a file system will have the capability to pass the message "`No more blocks on the file.`" to the transaction requesting sequential data.

If the condition of continuous sequential processing is violated, the previous buffer is not free to be refilled and we will lose one revolution or $2r$ prior to reading the next block. This loss is very significant, and will affect an application greatly if it happens regularly. With two buffers, and $r > c_{block} > btt$, this loss is incurred for every two buffers, and the effective bulk transfer rate is reduced to

$$t' = \frac{2B}{2r + 2btt} = \frac{B}{r + btt} \qquad \text{if} \qquad r > c\,Bfr > btt \tag{2-23}$$

In most cases this causes a reduction of processing speed by several factors.

If $c_{block} > r$, it becomes reasonable to assume a random delay based on search and latency with an average value $c + r$ for every block, so that

$$t' = \frac{B}{c + r} \qquad \text{if} \qquad c\,Bfr > r \tag{2-24}$$

The use of more buffers to hold more blocks in memory will, within the limits imposed by the computation time c, allow for a short time bulk-transfer rates higher than as shown in Eq. 2-24. Eventually all buffers will be filled and file reading will have to stop. As long as sufficient buffers are available, the rate of sequential reading or writing is then constrained by two factors, computation and device speed (see Eq. 2-14),

$$t' = \min\left(\frac{R}{c}\,,\, t\,\frac{R}{R + W}\right) \qquad \langle\text{limit}\rangle\ \text{2-25}$$

The purpose of buffering is to achieve this minimum. The assumption of unlimited buffers is rarely true but the having more than two buffers will even out the effects of computation times that vary, but that on the average are $c \approx (R + W)/t$ or smaller. Irregular computation can be caused in multiprogrammed systems by competing computations.

The requirement for two buffers becomes essential when blocks containing spanned records are to be processed, as discussed in Sec. 2-2-4. We assume that blocks involved in spanning are always sequentially allocated.

Double buffering is also beneficial when records are read serially that are not sequential. The reading time for a record will now be longer, $s + r + btt$, and the permissible computation time c will be greater. It is then easier to completly hide the cost of computation in the disk access and transfer time.

Condition for continuous random processing

$$c < s + r + btt \tag{2-26}$$

2-3-5 Updating Blocks

Updating of data contained within a block requires more time than either the read or write operation alone, since at least one read and one write is required. To append a record to the file, it has to be inserted into an existing block containing predecessor records unless that last block happens to be full. This last block may still be read to find out that it is full, unless counters, kept in the file directory in memory, keep track of the status of the last block written. Change of an existing record will always require the block to be read so that all other records, and other fields of the changed record, are available.

After the reading of a block, the new record is inserted into the memory buffer containing the block and the block is rewritten. In most instances it is necessary to rewrite the block into its original position so that the file structure remains unchanged. If the insertion of the record in the memory buffer for the block is fast, it will be possible to rewrite the block during the next disk revolution. Such a rewrite operation will take one revolution or approximately

$$T_{RW} = 2r \qquad \text{if} \qquad c_{update} \ll 2r \tag{2-27}$$

This assumption is generally made throughout the text. This condition is not as restrictive as the one required to continuously read sequential data (Eq. 2-22), since $2r = nbt\,btt \gg btt$ for multiple blocks per track ($nbt > 1$), and $c_{update} < c\,Bfr$ for one record being changed at a time.

For devices other than disks the rewrite time has to be specifically analyzed. For magnetic-tape units with the capability to *backspace* a block, the value $2r$ can be replaced by two block transfer times and two delays to account for the gaps, or $T_{RW} \approx 2(B+G)/t \approx 2B/t'$. Some typical values of T_{RW} are given in Table 2-1.

CONDITIONS FOR REWRITING IN ONE REVOLUTION A more rigorous limit on the allowable value of c for disk rewrites can be deduced by inspecting Fig. 2-17. The assumption that $T_{RW} = 2r$, for the rewriting of a single block i, is valid when the time required for computation and update c is such that the following relation will be true:

$$T_{RW} = 2r \quad \text{if} \quad c < 2r\frac{(nbt-1)(B+G)+G}{nbt\,(B+G)} \quad \text{or if} \quad c < 2r\frac{nbt-1}{nbt} \tag{2-28}$$

If there is more than one block per track ($nbt > 1$) and the system is designed so that the update has sufficient priority, this condition will generally be true.

If a rewrite is not accomplished in time, a second revolution is lost, so that $T_{RW} = 4r$. When there is a fluctuation of the computational time, sometimes one, at other times two or more, revolutions will be lost. The use of the average value of c in the analysis will give erroneous results, since the relationship between c and T_{RW} is not linear. Chapter 10 will present methods to evaluate such cases. If there is other use of the disk unit, a new search is required when the computation is completed and

$$T_{RW} = c + T_F \qquad \text{if} \qquad c \gg 2r \tag{2-29}$$

where T_F is the time to fetch a record without any benefit of prior conditions.

To be expanded to 205 pt for Figure space
was fig.2-17

Figure 2-18 Rewrite-time considerations.

2-4 SUMMARY OF FILE HARDWARE

We have, in the preceding sections, taken a variety of hardware types and a number of basic, low-level, file-software implementation choices and distilled their effect into a small number of parameters: seek time s, rotational latency r, and transfer rates t and t'. We use these parameters in the next chapters to produce measures of file-system performance. In Chap. 12 advanved concepts in hardware architecture will be surveyed and in Chap. ¡13¿ we consider the effect of parallel operations of devices on these parameters.

2-4-1 Summary of Hardware Parameters

We summarize these parameters now in the context for which they were derived, namely, the evaluation of the performance of files and databases which use the prevalent disk devices. We will then discuss the applicability of these parameters for alternate storage devices.

Parameter	Unit	Code	Function	Condition
Seek time	ms	s	Mechanical delay	Device requires positioning
Latency	ms	r	Rotational delay	Block access or 1/2 reaccess
Transfer rate	char./ms	t	Block transfer	Hardware speed
Bulk transfer rate	char./ms	t'	Sequential transfer	Requires buffering
Block size	char.	B	Fixed transfer unit	Hardware or system specified
Blocking factor		Bfr	Records per block	Blocking method and fit

Table 2-5 Primary performance parameters for disks

Estimation of the bulk transfer rate t' requires determination of the waste W due to blocking and gaps G, and consideration of a minimal seek s_1 or a latency loss $2r$ whenever a cylinder break occurs. Bulk transfer implies continuous operation which requires buffering and assuring that $c\,Bfr < btt$.

These, and other assumptions made, require periodic verification during the design of a database system and the evaluation of its performance. The sections where the parameters are derived may have to be reread to assure that the formulas remain applicable in specific situations. We do not include a table of formulas here, since that could encourage their erroneous application. The index of the book can be used to locate the page where the derivation of the parameter begins.

From these primary parameters we derive some useful secondary parameters. The time to transfer a block having B characters, after positioning and latency delays, is $btt = B/t$ ms. The random-access time to a block is then $s + r + btt$, but a block being read sequentially is transferred in a time B/t' ms. A shorthand notation for the time to reaccess a block, typically done in order to rewrite it with changed data, is $T_{RW} = 2r$.

It is clear that this simple parameterization leaves some holes uncovered in the system-design process. In the remainder of this chapter we will raise some warning signs, so that we can avoid distracting the reader from the principal subjects in the chapters ahead. Whenever the simple hardware model used previously does not hold, a system-design process has to reevaluate the results obtained in this and the following chapters. The general methodology employed, however, should remain valid.

2-4-2 Differences among Storage Devices

The relative importance of the hardware parameters differs among the various types of hardware. We emphasize in the next chapters disk-drive units, without ruling out other devices. This means that we stress the effect of the seek time.

DISK VERSUS DRUM Drums and head per track disks, as well as most semiconductor memories, do not require a seek. When the seek time s is zero, the effect of rotational latency and transfer rate becomes more important than the discussions imply. All the formulas developed will still hold when $s = 0$. The benefits of double buffering is relatively greater, since it applies to the latency.

DISK VERSUS TAPE The methods discussed also apply to the use of tape for files. The block size B for tape is specified by the system used, and can be used also where the track size is required. Most tape units can read tapes in both the forward and the reverse direction and this provides the effective random seek time. Writing may only be possible in the forward direction. The expected random seek time for reading s can be estimated based on forward or backward searching through a third of the length of tape in use. Even when using the ability to read in either direction, this operation will take more than 1 min.

For writing, or if reverse reading is not possible, other values for the seek times will have to be determined, depending on the method used to find a record. A simple approach is to always rewind the tape and then search forward. Better strategies can be used if the current position of the tape is remembered and used. This permits tape to be spaced forward or backward prior to forward search to achieve minimal seek delays.

The latency r is due to the time taken to accelerate the tape to full speed. The gap G provides the space for getting up to speed and slowing down. The extremely high value of the expected random seek times effectively removes tape as a candidate for the file organizations presented in Chaps. 4 and 5.

Some operations can be faster on tape than on disk. To find a predecessor block on a disk unit takes most of a revolution. On a tape unit with reverse reading such a block can be read immediately in reverse mode. Even without reverse reading it is possible on tape to backspace over the block and then read or write it forward. This assumption was made for the rewrite time in Sec. 2-3-5.

We can incorporate the effect of rewrites for tape to the formulas of Chap. 3 by setting the value of the latency r to the block transfer time for tape. Similarly the single cylinder seek s_1 can be kept as $\approx 2r$. These simplifications introduce errors in the value of s for the seek times, but the effect is negligible because of the long seek times for tape under random access conditions.

More complex file architectures, as introduced in Chap. 12 require further analysis. The principles underlying the parameter derivations in this chapter can be applied also in those cases. On unusual devices it will be wise to run some tests, comparing measurements with analysis results. Avoiding analyses and only relying on measurements does not help in understanding, and can easily fail to reveal what happens in extreme or boundary conditions.

2-4-3 Directions

New approaches to data storage will continue to be developed. Many of these will be driven by innovation in hardware, and will not have been included in these discussions. The ongoing reduction in the cost of primary memory will permit some applications to avoid secondary storage entirely, or to use secondary storage only for backup purposes, but larger and growing collections of data will continue to strain current and future systems.

Since few truly novel systems have yet attained operational status, little experience is available to judge system-design alternatives. Alternate system organization strategies are often tried by industry while academic institutions provide analysis and evaluation. A detailed analytic evaluation of new and complex systems is often beyond the current state of the art, although many of the engineering level techniques for performance analysis presented here, combined with simulation and fundamental experiments, can direct further work and avoid expenditure of effort in fruitless directions.

BACKGROUND AND REFERENCES

Storage technology had its beginnings in card and paper-tape filing systems. The original Hollerith card was dimensioned on the basis of the dollar bill in 1890, as documented by Billings[1892]. The dollar and its value have shrunk since then, but storage costs have decreased even faster. Half-inch digital tapes were first used in 1953. Later in the fifties, UNIVAC computers employed coated steel tapes and strong operating personnel. The recording density has increased by a factor of 100 since that time.

The first disk units were used in 1956 on RAMAC (Random Access Memory Accounting Computers), machines which required users to insert wires to specify the logic of their programs. Two read-write heads on one access arm were moved up or down, and inserted between a stack of 20 fixed disks. Seek times were often many seconds. Removable diskpacks appeared in 1962, and in 1973 WINCHESTER diskpacks with integral head assemblies became available. The capacity increase from RAMAC disks with 5M bytes to 1 200M bytes for 3380-type units exceeds the advances seen in tape technology.

Recording technology is described in White[80]. Bell[83] presents curreny issues in magnetic and optical recording. Developments in magnetic technology are covered in the IEEE *Transactions on Magnetics*; Pugh[71] provides an early overview of storage devices and their use, and Michaels[75] describes a bubble mass memory. Bobeck[75] tells all about bubbles; Doty[80] assesses them for databases. The *Bell System Technical Journal* has many original papers in this area beginning with Thiele[69].

A general reference for computer hardware architecture is Mano[82], Myers[81] presents novel storage architectures. More than 30 different types of disk units are now on the market. Surveys of available devices appear regularly in the trade literature and are also obtainable from companies specializing in market surveys. A general reference model was proposed by Miller[85]. Specific information regarding computer hardware is best obtained from manufacturers; frequently their promotional literature omits some of the parameters needed for a proper performance evaluation. A regular feature of computer publications is a projection of future hardware availability and costs; an example is Chi[82].

Prices of computing equipment can be obtained from the price schedules prepared for the General Services Administration (GSA[57,...]). Sharpe[69] provides an analysis of storage-device economics. Often obsolete equipment is used beyond its economic lifetime, GAO[80] cites examples.

An overview of storage technology (semiconductor, tape, disk, bubble, optical devices, automated libraries) is found in Hoagland[84] and projected in Hoagland[85]. Instructive descriptions of hardware-design considerations appear regularly in the IBM *Journal for Research and Development*; see, for instance, Mulvany[74] on the design of a diskpack system with an integral access mechanism. Design decisions leading to the IBM 3330 sector-addressing scheme are considered in Brown[72]. Floppy disk performance is analyzed in Pechura[83] and Dunnigan[84] reviews the small Winchester disks used in personal computers.

Papers describing new systems and technology appear regularly in the proceedings of the AFIPS conferences and IEEE COMPUTER; Johnson[75] describes the automatic-tape library which provides backup to disks and Michael[80] and Savage[85] give general specifications.

Economical optical discs are announced in Fujitani[84] and a multi-platter optical disc unit, developed for NASA is described by Ammon[85]. An issue of IEEE COMPUTER is devoted to optical storage; Copeland[82] argues that its availability will change many habits of data storage. Access software will have to undergo changes, as described by Rathmann[84], Vitter[85], and Copeland[86].

Programming for OS-360 files is taught by Chapin[68]. The relatively low-level programming and performance analysis methods presented have seen little formal publication; some are documented in IBM GC20-1649. Mass storage software is considered by O'Leary[82].

Most performance issues have been analyzed piecemeal. Knuth[73S] reviews general aspects of disk-performance analysis. Wong[80] minimizes access movement and Gaynor[74] calculates seek times. Anzelmo[71] and Inglis[73] discuss record blocking, Leung[82] presents the pros and cons of fixed length records, March[83] record fragmentation and placement, and Hakola[80] looks at waste.

The low rate of interference seen in multiprogramming is documented by Rodriguez-Rosell[76]. Effelsberg[84] analyzes buffering and Sacco[82] and Teng[84] consider buffer usage and management. Salasin[73] evaluates record fetch costs; Waters[72,75] discusses file placement to optimize access Baber[63] evaluates record access on tape and Ghosh[76] concentrates on optimal sequential placement.

Other techniques are part of programming lore, noted at best as a paragraph in some system description. Many of these are specific to certain hardware combinations and have not been presented here. On large machines users have little control of the storage system, but on smaller personal or network computers there are many opportunities to improve storage-system performance.

Measurement and tuning of systems is covered by Ferrari[83]. Locality of transaction accesses has been reviewed by Kearns[83].

EXERCISES

1 Derive the result stated in Sec. 2-1-2 for the average distance to be traveled when accessing tape randomly (one-third of the length of tape is used).

2 Why is the surface area available a concern when building a magnetic storage device? What is the advantage of many reading heads?

3 Read a manufacturer's description of a disk file or other storage device not now included and determine the parameters for Table 2-1.

4 What is the seek time for tape using the simple approach of Sec. 2-4-1? Assume that rewinding over b blocks requires $btt(5 + b/10)$.

5 Design a better method to seek tape blocks when a backspace operation, but no read backward operation, is available. Backspacing over a block takes as long as reading a block btt.

6 Compute the bulk transfer rate for an DEC ¡RS-60¿ transferring blocks of 300 chars.

7 Determine the net transfer rate seen by the user for the ¡RS-60¿ if for each block 1.0 ms of computation is required and three buffers are available.

8 Solve the same problem (by simulation?) if the computation time is evenly distributed between 0 and 3 ms.

9 Estimate the performance for someone wanting to convert a Compact Audio Disc Player for databases on a micro computer. The disc revolves at 500 rpm. One disc can contain 60 min worth of music. Transfer rate to the micro-computer CPU is limited by the CPU at 100 chars/millisec. The reading head travels at linear rate over the surface and takes 3 sec between extremes. The directories of the disc have entries for up to 99 pieces of music (files). Each disc player unit costs $400 and it costs about the same to interface one to the micro.

Estimate all the parameters as a column of Table 2-1. Write down all assumptions made.

10 Your computation takes $c_{block} = 7$ ms per block while the file is being read sequentially and double-buffered. The values of btt and r are respectively 10 and 20 ms. A new computing application on the system runs with high priority and uses 50% of the CPU. How much is your program slowed down?