Chapter 2
Storage System Environment

KEY CONCEPTS

- Host, Connectivity, and Storage
- Block-Level and File-Level Access
- File System and Volume Manager
- Storage Media and Devices
- Disk Components
- Zoned Bit Recording
- Logical Block Addressing
- Little's Law and the Utilization Law

Storage, as one of the core elements of a data center, is recognized as a distinct resource and it needs focus and specialization for its implementation and management. The data flows from an application to storage through various components collectively referred as a *storage system environment.* The three main components in this environment are the host, connectivity, and storage. These entities, along with their physical and logical components, facilitate data access.

This chapter details the storage system environment and focuses primarily on storage. It provides details on various hardware components of a disk drive, disk geometry, and the fundamental laws that govern disk performance. The connectivity between the host and storage facilitated by bus technology and interface protocols is also explained.

This chapter provides an understanding of various logical components of hosts such as file systems, volume managers, and operating systems, and their role in the storage system environment.

2.1 Components of a Storage System Environment

The three main components in a storage system environment — the host, connectivity, and storage — are described in this section.

2.1.1 Host

Users store and retrieve data through applications. The computers on which these applications run are referred to as *hosts*. Hosts can range from simple laptops to complex clusters of servers. A host consists of physical components (hardware devices) that communicate with one another using logical components (software and protocols). Access to data and the overall performance of the storage system environment depend on both the physical and logical components of a host. The logical components of the host are detailed in Section 2.5 of this chapter.

Physical Components

A host has three key physical components:

- Central processing unit (CPU)
- Storage, such as internal memory and disk devices
- Input/Output (I/O) devices

The physical components communicate with one another by using a communication pathway called a *bus*. A bus connects the CPU to other components, such as storage and I/O devices.

CPU

The CPU consists of four main components:

- **Arithmetic Logic Unit (ALU):** This is the fundamental building block of the CPU. It performs arithmetical and logical operations such as addition, subtraction, and Boolean functions (AND, OR, and NOT).
- **Control Unit:** A digital circuit that controls CPU operations and coordinates the functionality of the CPU.
- **Register:** A collection of high-speed storage locations. The registers store intermediate data that is required by the CPU to execute an instruction and provide fast access because of their proximity to the ALU. CPUs typically have a small number of registers.
- **Level 1 (L1) cache:** Found on modern day CPUs, it holds data and program instructions that are likely to be needed by the CPU in the near future. The L1 cache is slower than registers, but provides more storage space.

Storage

Memory and storage media are used to store data, either persistently or temporarily. Memory modules are implemented using semiconductor chips, whereas storage devices use either magnetic or optical media. Memory modules enable data access at a higher speed than the storage media. Generally, there are two types of memory on a host:

- **Random Access Memory (RAM):** This allows direct access to any memory location and can have data written into it or read from it. RAM is volatile; this type of memory requires a constant supply of power to maintain memory cell content. Data is erased when the system's power is turned off or interrupted.
- **Read-Only Memory (ROM):** Non-volatile and only allows data to be read from it. ROM holds data for execution of internal routines, such as system startup.

Storage devices are less expensive than semiconductor memory. Examples of storage devices are as follows:

- Hard disk (magnetic)
- CD-ROM or DVD-ROM (optical)
- Floppy disk (magnetic)
- Tape drive (magnetic)

I/O Devices

I/O devices enable sending and receiving data to and from a host. This communication may be one of the following types:

- **User to host communications:** Handled by basic I/O devices, such as the keyboard, mouse, and monitor. These devices enable users to enter data and view the results of operations.
- **Host to host communications:** Enabled using devices such as a Network Interface Card (NIC) or modem.
- **Host to storage device communications:** Handled by a *Host Bus Adaptor (HBA)*. HBA is an application-specific integrated circuit (ASIC) board that performs I/O interface functions between the host and the storage, relieving the CPU from additional I/O processing workload. HBAs also provide connectivity outlets known as *ports* to connect the host to the storage device. A host may have multiple HBAs.

2.1.2 Connectivity

Connectivity refers to the interconnection between hosts or between a host and any other peripheral devices, such as printers or storage devices. The discussion here focuses on the connectivity between the host and the storage device. The components of connectivity in a storage system environment can be classified as physical and logical. The *physical components* are the hardware elements that connect the host to storage and the *logical components* of connectivity are the protocols used for communication between the host and storage. The communication protocols are covered in Chapter 5.

Physical Components of Connectivity

The three physical components of connectivity between the host and storage are Bus, Port, and Cable (Figure 2-1).

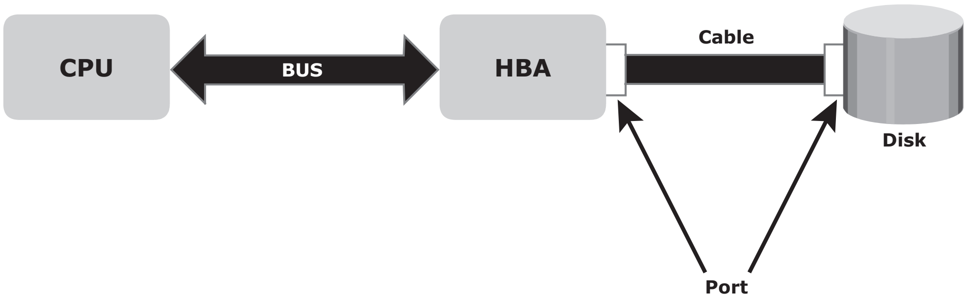

Figure 2-1: Physical components of connectivity

The *bus* is the collection of paths that facilitates data transmission from one part of a computer to another, such as from the CPU to the memory. The *port* is a specialized outlet that enables connectivity between the host and external devices. *Cables* connect hosts to internal or external devices using copper or fiber optic media.

Physical components communicate across a bus by sending bits (control, data, and address) of data between devices. These bits are transmitted through the bus in either of the following ways:

- **Serially:** Bits are transmitted sequentially along a single path. This transmission can be unidirectional or bidirectional.
- **In parallel:** Bits are transmitted along multiple paths simultaneously. Parallel can also be bidirectional.

The size of a bus, known as its width, determines the amount of data that can be transmitted through the bus at one time. The width of a bus can be compared to the number of lanes on a highway. For example, a 32-bit bus can transmit 32 bits of data and a 64-bit bus can transmit 64 bits of data simultaneously. Every bus has a clock speed measured in MHz (megahertz). These represent the data transfer rate between the end points of the bus. A fast bus allows faster transfer of data, which enables applications to run faster.

Buses, as conduits of data transfer on the computer system, can be classified as follows:

- **System bus:** The bus that carries data from the processor to memory.
- **Local or I/O bus:** A high-speed pathway that connects directly to the processor and carries data between the peripheral devices, such as storage devices and the processor.

Logical Components of Connectivity

The popular interface protocol used for the local bus to connect to a peripheral device is *peripheral component interconnect (PCI)*. The interface protocols that connect to disk systems are *Integrated Device Electronics/Advanced Technology Attachment (IDE/ATA)* and *Small Computer System Interface (SCSI)*.

PCI

PCI is a specification that standardizes how PCI expansion cards, such as network cards or modems, exchange information with the CPU. PCI provides the interconnection between the CPU and attached devices. The plug-and-play functionality of PCI enables the host to easily recognize and configure new cards and devices. The width of a PCI bus can be 32 bits or 64 bits. A 32-bit PCI bus can provide a throughput of 133 MB/s. *PCI Express* is an enhanced version of PCI bus with considerably higher throughput and clock speed.

IDE/ATA

IDE/ATA is the most popular interface protocol used on modern disks. This protocol offers excellent performance at relatively low cost. Details of IDE/ATA are provided in Chapter 5.

SCSI

SCSI has emerged as a preferred protocol in high-end computers. This interface is far less commonly used than IDE/ATA on personal computers due to its higher cost. SCSI was initially used as a parallel interface, enabling the connection of devices to a host. SCSI has been enhanced and now includes a wide variety of related technologies and standards. Chapter 5 provides details of SCSI.

2.1.3 Storage

The storage device is the most important component in the storage system environment. A storage device uses magnetic or solid state media. Disks, tapes, and diskettes use magnetic media. CD-ROM is an example of a storage device that uses optical media, and removable flash memory card is an example of solid state media.

Tapes are a popular storage media used for backup because of their relatively low cost. In the past, data centers hosted a large number of tape drives and processed several thousand reels of tape. However, tape has the following limitations:

- Data is stored on the tape linearly along the length of the tape. Search and retrieval of data is done sequentially, invariably taking several seconds to access the data. As a result, random data access is slow and time consuming. This limits tapes as a viable option for applications that require real-time, rapid access to data.
- In a shared computing environment, data stored on tape cannot be accessed by multiple applications simultaneously, restricting its use to one application at a time.
- On a tape drive, the read/write head touches the tape surface, so the tape degrades or wears out after repeated use.
- The storage and retrieval requirements of data from tape and the overhead associated with managing tape media are significant.

In spite of its limitations, tape is widely deployed for its cost effectiveness and mobility. Continued development of tape technology is resulting in high capacity medias and high speed drives. Modern tape libraries come with additional memory (cache) and / or disk drives to increase data throughput. With these and added intelligence, today's tapes are part of an end-to-end data management solution, especially as a low-cost solution for storing infrequently accessed data and as long-term data storage.

Optical disk storage is popular in small, single-user computing environments. It is frequently used by individuals to store photos or as a backup medium on personal/laptop computers. It is also used as a distribution medium for single applications, such as games, or as a means of transferring small amounts of data from one self-contained system to another. Optical disks have limited capacity and speed, which limits the use of optical media as a business data storage solution.

The capability to write once and read many (WORM) is one advantage of optical disk storage. A CD-ROM is an example of a WORM device. Optical disks, to some degree, guarantee that the content has not been altered, so they can be used as low-cost alternatives for long-term storage of relatively small amounts of fixed content that will not change after it is created. Collections of optical disks in an array, called *jukeboxes*, are still used as a fixed-content storage solution. Other forms of optical disks include CD-RW and variations of DVD.

Disk drives are the most popular storage medium used in modern computers for storing and accessing data for performance-intensive, online applications. Disks support rapid access to random data locations. This means that data can be written or retrieved quickly for a large number of simultaneous users or applications. In addition, disks have a large capacity. Disk storage arrays are configured with multiple disks to provide increased capacity and enhanced performance.

2.2 Disk Drive Components

A disk drive uses a rapidly moving arm to read and write data across a flat platter coated with magnetic particles. Data is transferred from the magnetic platter through the R/W head to the computer. Several platters are assembled together with the R/W head and controller, most commonly referred to as a *hard disk drive (HDD)*. Data can be recorded and erased on a magnetic disk any number of times. This section details the different components of the disk, the mechanism for organizing and storing data on disks, and the factors that affect disk performance.

Key components of a disk drive are *platter, spindle, read/write head, actuator arm assembly, and controller* (Figure 2-2):

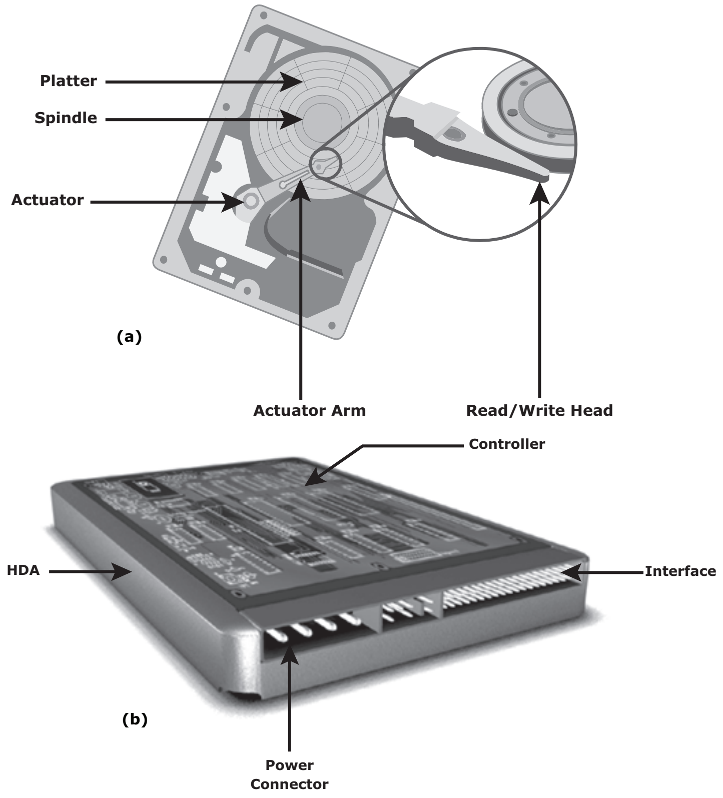

Figure 2-2: Disk Drive Components

2.2.1 Platter

A typical HDD consists of one or more flat circular disks called *platters* (Figure 2-3). The data is recorded on these platters in binary codes (0s and 1s). The set of rotating platters is sealed in a case, called a *Head Disk Assembly (HDA)*. A platter is a rigid, round disk coated with magnetic material on both surfaces (top and bottom). The data is encoded by polarizing the magnetic area, or domains, of the disk surface. Data can be written to or read from both surfaces of the platter. The number of platters and the storage capacity of each platter determine the total capacity of the drive.

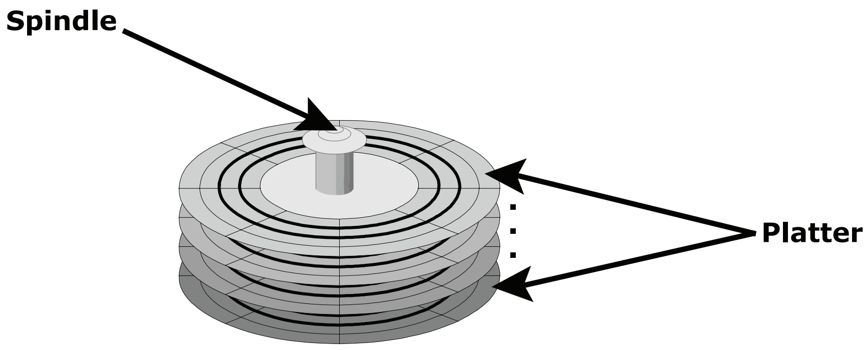

Figure 2-3: Spindle and platter

2.2.2 Spindle

A spindle connects all the platters, as shown in Figure 2-3, and is connected to a motor. The motor of the spindle rotates with a constant speed.

The disk platter spins at a speed of several thousands of revolutions per minute (rpm). Disk drives have spindle speeds of 7,200 rpm, 10,000 rpm, or 15,000 rpm. Disks used on current storage systems have a platter diameter of 3.5″ (90 mm). When the platter spins at 15,000 rpm, the outer edge is moving at around 25 percent of the speed of sound. The speed of the platter is increasing with improvements in technology, although the extent to which it can be improved is limited.

2.2.3 Read/Write Head

Read/Write (R/W) heads, shown in Figure 2-4, read and write data from or to a platter. Drives have two R/W heads per platter, one for each surface of the platter. The R/W head changes the magnetic polarization on the surface of the platter when writing data. While reading data, this head detects magnetic polarization on the surface of the platter. During reads and writes, the R/W head senses the magnetic polarization and never touches the surface of the platter. When the spindle is rotating, there is a microscopic air gap between the R/W heads and the platters, known as the *head flying height*. This air gap is removed when the spindle stops rotating and the R/W head rests on a special area on the platter near the spindle. This area is called the *landing zone*. The landing zone is coated with a lubricant to reduce friction between the head and the platter.

The logic on the disk drive ensures that heads are moved to the landing zone before they touch the surface. If the drive malfunctions and the R/W head accidentally touches the surface of the platter outside the landing zone, a *head crash* occurs. In a head crash, the magnetic coating on the platter is scratched and may cause damage to the R/W head. A head crash generally results in data loss.

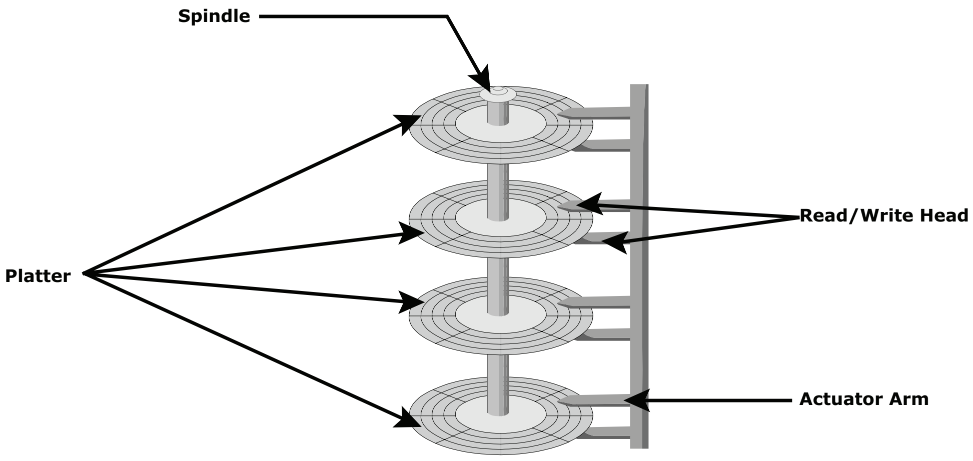

Figure 2-4: Actuator arm assembly

2.2.4 Actuator Arm Assembly

The R/W heads are mounted on the *actuator arm assembly* (refer to Figure 2-2 [a]), which positions the R/W head at the location on the platter where the data needs to be written or read. The R/W heads for all platters on a drive are attached to one actuator arm assembly and move across the platters simultaneously. Note that there are two R/W heads per platter, one for each surface, as shown in Figure 2-4.

2.2.5 Controller

The *controller* (see Figure 2-2 [b]) is a printed circuit board, mounted at the bottom of a disk drive. It consists of a microprocessor, internal memory, circuitry, and firmware. The firmware controls power to the spindle motor and the speed of the motor. It also manages communication between the drive and the host. In addition, it controls the R/W operations by moving the actuator arm and switching between different R/W heads, and performs the optimization of data access.

2.2.6 Physical Disk Structure

Data on the disk is recorded on *tracks*, which are concentric rings on the platter around the spindle, as shown in Figure 2-5. The tracks are numbered, starting from zero, from the outer edge of the platter. The number of *tracks per inch (TPI)* on the platter (or the *track density*) measures how tightly the tracks are packed on a platter.

Each track is divided into smaller units called *sectors*. A sector is the smallest, individually addressable unit of storage. The track and sector structure is written on the platter by the drive manufacturer using a formatting operation. The number of sectors per track varies according to the specific drive. The first personal computer disks had 17 sectors per track. Recent disks have a much larger number of sectors on a single track. There can be thousands of tracks on a platter, depending on the physical dimensions and recording density of the platter.

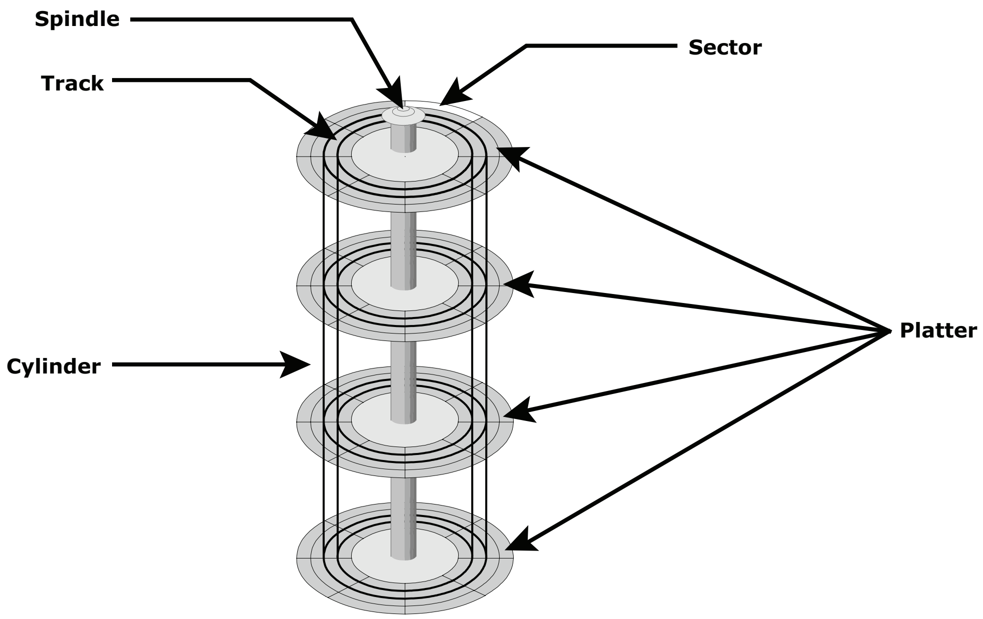

Figure 2-5: Disk structure: sectors, tracks, and cylinders

Typically, a sector holds 512 bytes of user data, although some disks can be formatted with larger sector sizes. In addition to user data, a sector also stores other information, such as sector number, head number or platter number, and track number. This information helps the controller to locate the data on the drive, but storing this information consumes space on the disk. Consequently, there is a difference between the capacity of an unformatted disk and a formatted one. Drive manufacturers generally advertise the unformatted capacity — for example, a disk advertised as being 500GB will only hold 465.7GB of user data, and the remaining 34.3GB is used for *metadata*.

A cylinder is the set of identical tracks on both surfaces of each drive platter. The location of drive heads is referred to by cylinder number, not by track number.

2.2.7 Zoned Bit Recording

Because the platters are made of concentric tracks, the outer tracks can hold more data than the inner tracks, because the outer tracks are physically longer than the inner tracks, as shown in Figure 2-6 (a). On older disk drives, the outer tracks had the same number of sectors as the inner tracks, so data density was low on the outer tracks. This was an inefficient use of available space.

Zone bit recording utilizes the disk efficiently. As shown in Figure 2-6 (b), this mechanism groups tracks into zones based on their distance from the center of the disk. The zones are numbered, with the outermost zone being zone 0. An appropriate number of sectors per track are assigned to each zone, so a zone near the center of the platter has fewer sectors per track than a zone on the outer edge. However, tracks within a particular zone have the same number of sectors.

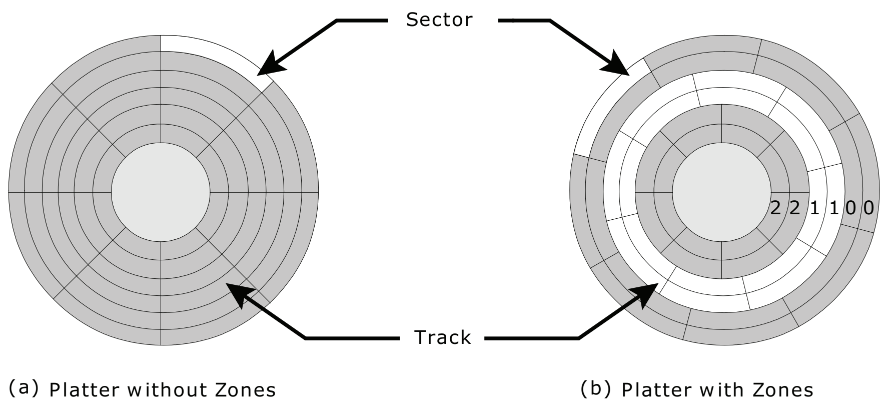

Figure 2-6: Zoned bit recording

The data transfer rate drops while accessing data from zones closer to the center of the platter. Applications that demand high performance should have their data on the outer zones of the platter.

2.2.8 Logical Block Addressing

Earlier drives used physical addresses consisting of the *cylinder, head, and sector (CHS)* number to refer to specific locations on the disk, as shown in Figure 2-7 (a), and the host operating system had to be aware of the geometry of each disk being used. *Logical block addressing (LBA)*, shown in Figure 2-7 (b), simplifies addressing by using a linear address to access physical blocks of data. The disk controller translates LBA to a CHS address, and the host only needs to know the size of the disk drive in terms of the number of blocks. The logical blocks are mapped to physical sectors on a 1:1 basis.

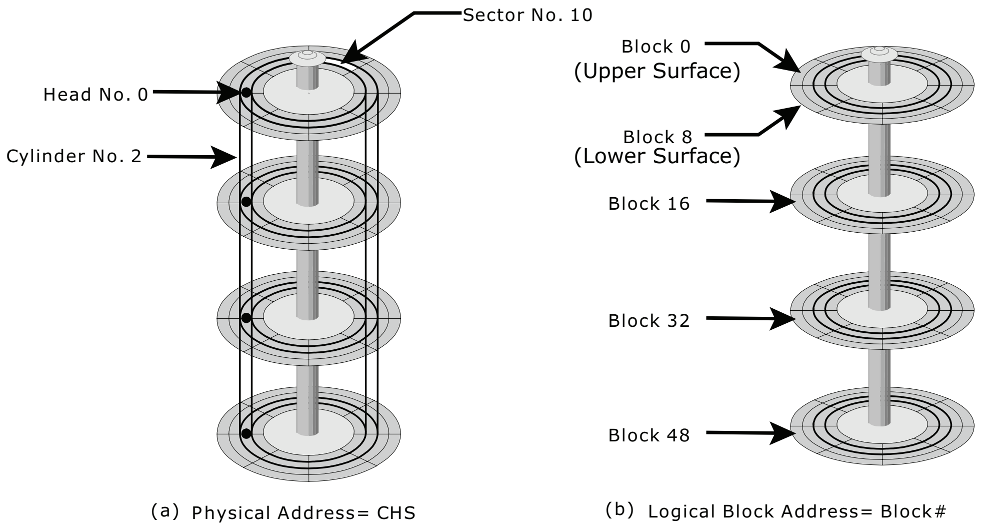

Figure 2-7: Physical address and logical block address

In Figure 2-7 (b), the drive shows eight sectors per track, eight heads, and four cylinders. This means a total of 8 × 8 × 4 = 256 blocks, so the block number ranges from 0 to 255. Each block has its own unique address. Assuming that the sector holds 512 bytes, a 500 GB drive with a formatted capacity of 465.7 GB will have in excess of 976,000,000 blocks.

2.3 Disk Drive Performance

A disk drive is an electromechanical device that governs the overall performance of the storage system environment. The various factors that affect the performance of disk drives are discussed in this section.

2.3.1 Disk Service Time

Disk service time is the time taken by a disk to complete an I/O request. Components that contribute to service time on a disk drive are *seek time, rotational latency,* and *data transfer rate.*

Seek Time

The *seek time* (also called *access time*) describes the time taken to position the R/W heads across the platter with a radial movement (moving along the radius of the platter). In other words, it is the time taken to reposition and settle the arm and the head over the correct track. The lower the seek time, the faster the I/O operation. Disk vendors publish the following seek time specifications:

- **Full Stroke:** The time taken by the R/W head to move across the entire width of the disk, from the innermost track to the outermost track.
- **Average:** The average time taken by the R/W head to move from one random track to another, normally listed as the time for one-third of a full stroke.
- **Track-to-Track:** The time taken by the R/W head to move between adjacent tracks.

Each of these specifications is measured in milliseconds. The average seek time on a modern disk is typically in the range of 3 to 15 milliseconds. Seek time has more impact on the read operation of random tracks rather than adjacent tracks. To minimize the seek time, data can be written to only a subset of the available cylinders. This results in lower usable capacity than the actual capacity of the drive. For example, a 500 GB disk drive is set up to use only the first 40 percent of the cylinders and is effectively treated as a 200 GB drive. This is known as *short-stroking* the drive.

Rotational Latency

To access data, the actuator arm moves the R/W head over the platter to a particular track while the platter spins to position the requested sector under the R/W head. The time taken by the platter to rotate and position the data under the R/W head is called *rotational latency*. This latency depends on the rotation speed of the spindle and is measured in milliseconds. The average rotational latency is one-half of the time taken for a full rotation. Similar to the seek time, rotational latency has more impact on the reading/writing of random sectors on the disk than on the same operations on adjacent sectors.

Average rotational latency is around 5.5 ms for a 5,400-rpm drive, and around 2.0 ms for a 15,000-rpm drive.

Data Transfer Rate

The *data transfer rate* (also called *transfer rate*) refers to the average amount of data per unit time that the drive can deliver to the HBA. It is important to first understand the process of read and write operations in order to calculate data transfer rates. In a *read operation*, the data first moves from disk platters to R/W heads, and then it moves to the drive's internal *buffer*. Finally, data moves from the buffer through the interface to the host HBA. In a *write operation*, the data moves from the HBA to the internal buffer of the disk drive through the drive's interface. The data then moves from the buffer to the R/W heads. Finally, it moves from the R/W heads to the platters.

The data transfer rates during the R/W operations are measured in terms of internal and external transfer rates, as shown in Figure 2-8.

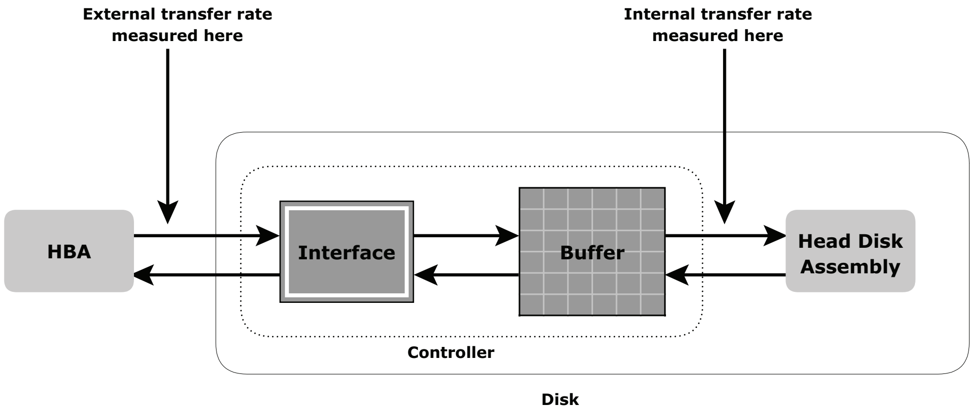
External transfer rate measured here
Internal transfer rate measured here
HBA
Interface
Buffer
Head Disk Assembly
Controller
Disk

Figure 2-8: Data transfer rate

Internal transfer rate is the speed at which data moves from a single track of a platter's surface to internal buffer (cache) of the disk. Internal transfer rate takes into account factors such as the seek time. *External transfer rate* is the rate at which data can be moved through the interface to the HBA. External transfer rate is generally the advertised speed of the interface, such as 133 MB/s for ATA. The sustained external transfer rate is lower than the interface speed.

2.4 Fundamental Laws Governing Disk Performance

To understand the laws of disk performance, a disk can be viewed as a black box consisting of two elements:

- **Queue:** The location where an I/O request waits before it is processed by the I/O controller.
- **Disk I/O Controller:** Processes I/Os that are waiting in the queue one by one.

The I/O requests arrive at the controller at the rate generated by the application. This rate is also called the *arrival rate*. These requests are held in the I/O queue, and the I/O controller processes them one by one, as shown in Figure 2-9. The I/O arrival rate, the queue length, and the time taken by the I/O controller to process each request determines the performance of the disk system, which is measured in terms of response time.

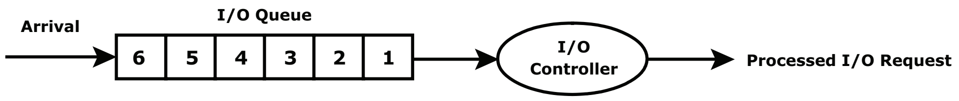

Figure 2-9: I/O processing

Little's Law is a fundamental law describing the relationship between the number of requests in a queue and the response time. The law states the following relation (numbers in parentheses indicate the equation number for cross-referencing):

$$\mathbf{N = a \times R} \quad (1)$$

where

"N" is the total number of requests in the queuing system (requests in the queue + requests in the I/O controller)

"a" is the arrival rate, or the number of I/O requests that arrive to the system per unit of time

"R" is the average response time or the turnaround time for an I/O request — the total time from arrival to departure from the system

The *utilization law* is another important law that defines the I/O controller utilization. This law states the relation:

$$\mathbf{U = a \times R_S} \quad (2)$$

where

"U" is the I/O controller utilization

"R_S" is the *service time,* or the average time spent by a request on the controller**.** $1/R_S$ is the *service rate.*

From the arrival rate "a", the average inter-arrival time, R_a, can be computed as:

$\mathbf{R_a = 1/a}$ (3)

Consequently, *utilization* can be defined as the ratio of the service time to the average inter-arrival time, and is expressed as:

$\mathbf{U = R_S / R_a}$ (4)

The value of this ratio varies between 0 and 1.

Here, it is important to realize that in a single controller system, the arrival rate must be smaller than the service rate. In other words, the service time must be smaller than the average inter-arrival time; otherwise, I/O requests will arrive into the system faster than the I/O controller can process them.

With the help of these two fundamental laws, a number of important measures of disk performance, such as average response time, average queue length, and time spent by a request in a queue can be derived.

In the following equation, the term average response rate (S) can be defined as the reciprocal of the average response time (R), and is derived as follows:

S = service rate – arrival rate

Consequently,

R = 1/ (service rate – arrival rate)

$R=1/(1/R_S - 1/R_a)$

$=1/(1/R_S-a)$ (from eq. 3)

$=R_S/(1-a \times R_S)$

$\mathbf{R = R_S/(1-U)}$ (5) (from eq. 2)

As a result,

Average response time (R) = service time/(1 – utilization)
(from equation 2)

As utilization reaches 1 — that is, as the I/O controller saturates — the response time is closer to infinity. In essence, the saturated component, or the bottleneck, forces the serialization of I/O requests, meaning each I/O request has to wait for the completion of the I/O requests that preceded it.

Utilization (U) can also be used to represent the average number of I/O requests on the controller, as shown in the following:

Number of requests in the queue (N_Q) = Number of requests in the system (N) – Number of requests on the controller or utilization (U). Number of requests in a queue is also referred to as *average queue size.*

$N_Q = N - U$

$= a \times R - U$ (from eq. 1)

$= a \times (R_S / (1 - U)) - U$ (from eq. 5)

$= (R_S / R_a) / (1 - U) - U$ (from eq. 3)

$= U / (1 - U) - U$ (from eq. 4)

$= U (1 / (1 - U) - 1)$

$\mathbf{= U^2 / (1 - U)}$ (6)

The time spent by a request in the queue is equal to the time spent by a request in the system, or the average response time minus the time spent by a request on the controller for processing:

$= R_s / (1 - U) - R_s$ (from eq. 5)

$= U \times R_S / (1 - U)$

= U × avg. response time

= Utilization × R (7)

Consider a disk I/O system in which an I/O request arrives at a rate of 100 I/Os per second. The service time, R_S, is 8 ms. The following measures of disk performance can be computed using the relationships developed above — utilization of I/O controller (U), total response time (R), average queue size [$U^2 / (1 - U)$] and total time spent by a request in a queue (U × R), as follows:

Arrival rate (a) = 100 I/O/s; consequently, the arrival time

$R_a = 1/a = 10$ ms

$R_S = 8$ ms (given)

1. Utilization (U) = $R_S / R_a = 8 / 10 = 0.8$ or 80%
2. Response time (R) = $R_S / (1 - U) = 8 / (1 - 0.8) = 40$ ms
3. Average queue size = $U^2 / (1 - U) = (0.8)^2 / (1 - 0.8) = 3.2$
4. Time spent by a request in a queue = U × R, or the total response time-service time = 32 ms

Now, if controller power is doubled, the service time is halved; consequently, $R_S = 4$ ms in this scenario.

1. Utilization (U) = 4 / 10 = 0.4 or 40%
2. Response time (R) = 4 / (1 – 0.4) = 6.67 ms
3. Average queue size = $(0.4)^2 / (1 - 0.4) = 0.26$
4. Time spent by a request in a queue = 0.4 × 6.67 = 2.67 ms

As a result, it can be concluded that by reducing the service time (the sum of seek time, latency, and internal transfer rate) or utilization by half, the response time can be reduced drastically (almost six times in the preceding example). The relationship between utilization and response time is shown in Figure 2-10.

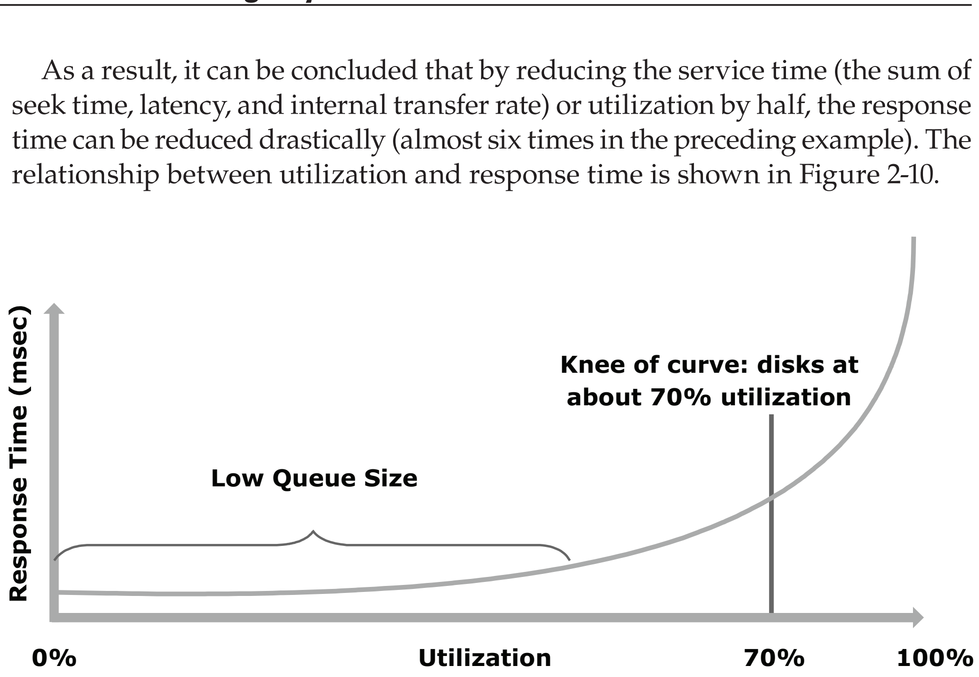

Figure 2-10: Utilization vs. Response time

Response time changes are nonlinear as utilization increases. When the average queue sizes are low, response time remains low. Response time increases slowly with added load on the queue, and increases exponentially when utilization exceeds 70 percent.

2.5 Logical Components of the Host

The logical components of a host consist of the software applications and protocols that enable data communication with the user as well as the physical components. Following are the logical components of a host:

- Operating system
- Device drivers
- Volume manager
- File system
- Application

2.5.1 Operating System

An *operating system* controls all aspects of the computing environment. It works between the application and physical components of the computer system. One of the services it provides to the application is data access. The operating system also monitors and responds to user actions and the environment. It organizes and controls hardware components and manages the allocation of hardware resources. It provides basic security for the access and usage of all managed resources. An operating system also performs basic storage management tasks while managing other underlying components, such as the file system, volume manager, and device drivers.

2.5.2 Device Driver

A *device driver* is special software that permits the operating system to interact with a specific device, such as a printer, a mouse, or a hard drive. A device driver enables the operating system to recognize the device and to use a standard interface (provided as an *application programming interface*, or *API*) to access and control devices. Device drivers are hardware dependent and operating system specific.

2.5.3 Volume Manager

In the early days, an HDD appeared to the operating system as a number of continuous disk blocks. The entire HDD would be allocated for the file system or other data entity used by the operating system or application. The disadvantage was lack of flexibility: As an HDD ran out of space, there was no easy way to extend the file system's size. As the storage capacity of the HDD increased, allocating the entire HDD for the file system often resulted in underutilization of storage capacity.

Disk partitioning was introduced to improve the flexibility and utilization of HDDs. In partitioning, an HDD is divided into logical containers called *logical volumes (LVs)* (see Figure 2-11). For example, a large physical drive can be partitioned into multiple LVs to maintain data according to the file system's and applications' requirements. The partitions are created from groups of contiguous cylinders when the hard disk is initially set up on the host. The host's file system accesses the partitions without any knowledge of partitioning and the physical structure of the disk.

Concatenation is the process of grouping several smaller physical drives and presenting them to the host as one logical drive (see Figure 2-11).

The evolution of *Logical Volume Managers (LVMs)* enabled the dynamic extension of file system capacity and efficient storage management. LVM is software that runs on the host computer and manages the logical and physical storage. LVM is an optional, intermediate layer between the file system and the physical disk. It can aggregate several smaller disks to form a larger virtual disk or to partition a larger-capacity disk into virtual, smaller-capacity disks, which are then presented to applications. The LVM provides optimized storage access and simplifies storage resource management. It hides details about the physical disk and the location of data on the disk; and it enables administrators to change the storage allocation without changing the hardware, even when the application is running.

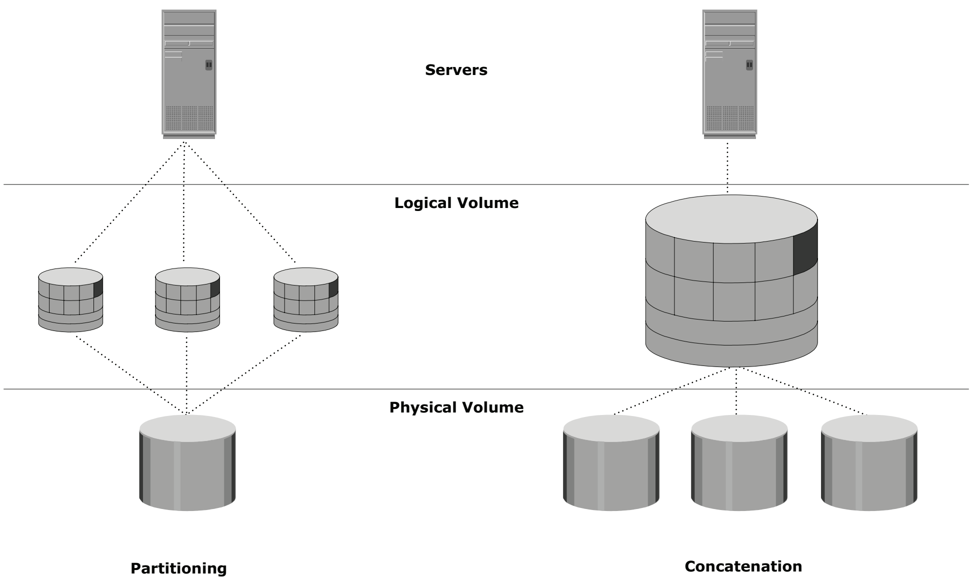

Figure 2-11: Disk partitioning and concatenation

The basic LVM components are the *physical volumes*, *volume groups*, and *logical volumes*. In LVM terminology, each physical disk connected to the host system is a *physical volume (PV)*. LVM converts the physical storage provided by the physical volumes to a logical view of storage, which is then used by the operating system and applications. A *volume group* is created by grouping together one or more physical volumes. A unique *physical volume identifier (PVID)* is assigned to each physical volume when it is initialized for use by the LVM. Physical volumes can be added or removed from a volume group dynamically. They cannot be shared between volume groups; the entire physical volume becomes part of a volume group. Each physical volume is partitioned into equal-sized data blocks called *physical extents* when the volume group is created.

Logical volumes are created within a given volume group. A logical volume can be thought of as a virtual disk partition, while the volume group itself can be thought of as a disk. A volume group can have a number of logical volumes. The size of a logical volume is based on a multiple of the physical extents.

The logical volume appears as a physical device to the operating system. A logical volume can be made up of noncontiguous physical partitions and can span multiple physical volumes. A file system can be created on a logical volume; and logical volumes can be configured for optimal performance to the application and can be mirrored to provide enhanced data availability.

2.5.4 File System

A *file* is a collection of related records or data stored as a unit with a name. A *file system* is a hierarchical structure of files. File systems enable easy access to data files residing within a disk drive, a disk partition, or a logical volume. A file system needs host-based logical structures and software routines that control access to files. It provides users with the functionality to create, modify, delete, and access files. Access to the files on the disks is controlled by the permissions given to the file by the owner, which are also maintained by the file system.

A file system organizes data in a structured hierarchical manner via the use of directories, which are containers for storing pointers to multiple files. All file systems maintain a pointer map to the directories, subdirectories, and files that are part of the file system. Some of the common file systems are as follows:

- FAT 32 (File Allocation Table) for Microsoft Windows
- NT File System (NTFS) for Microsoft Windows
- UNIX File System (UFS) for UNIX
- Extended File System (EXT2/3) for Linux

Apart from the files and directories, the file system also includes a number of other related records, which are collectively called the *metadata*. For example, metadata in a UNIX environment consists of the *superblock*, the *inodes*, and the list of data blocks free and in use. The metadata of a file system has to be consistent in order for the file system to be considered healthy. A superblock contains important information about the file system, such as the file system type, creation and modification dates, size and layout, the count of available resources (such as number of free blocks, inodes, etc.), and a flag indicating the mount status of the file system. An inode is associated with every file and directory and contains information about file length, ownership, access privileges, time of last access/modification, number of links, and the addresses for finding the location on the physical disk where the actual data is stored.

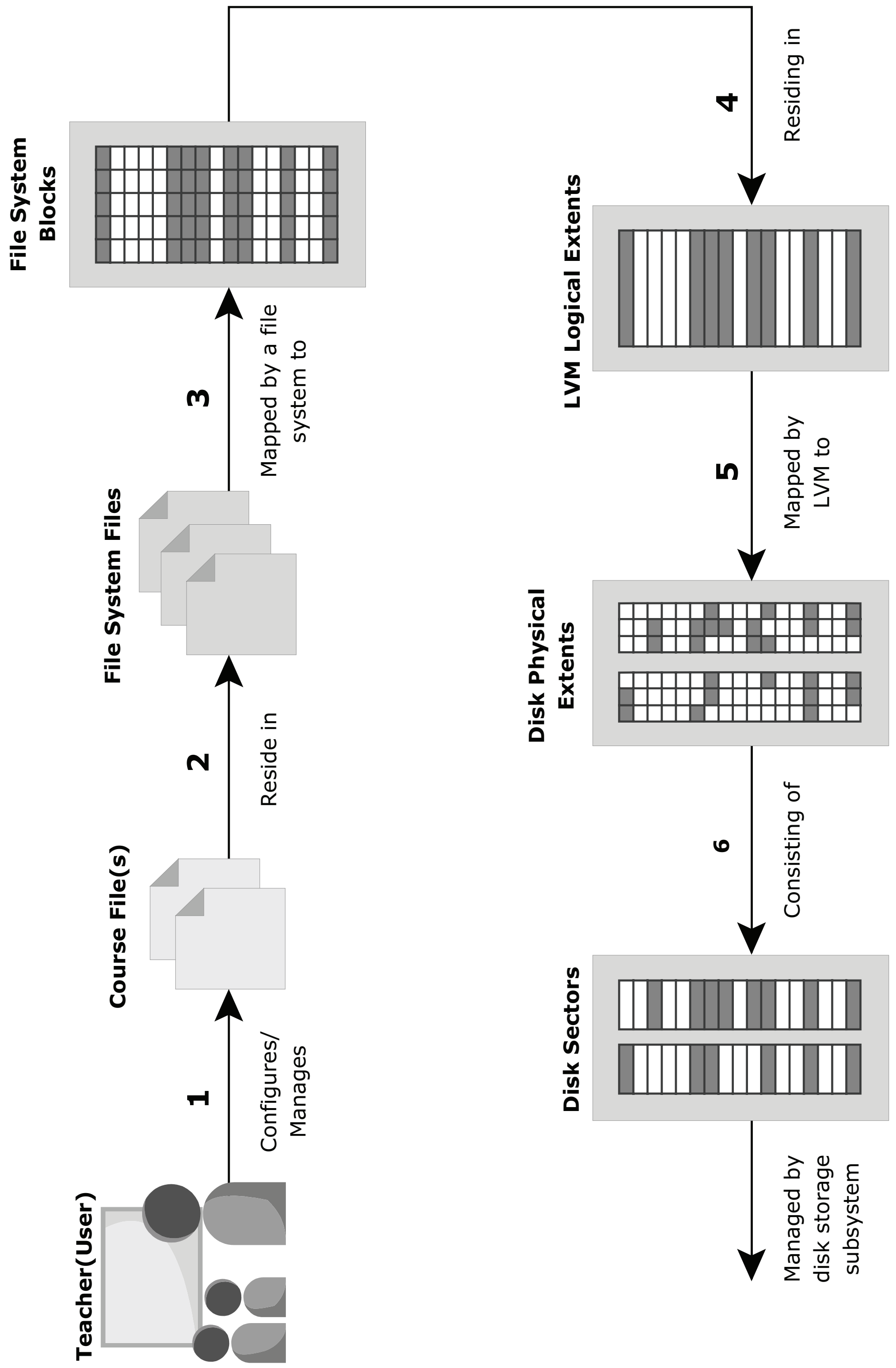

Figure 2-12: Process of mapping user files to disk storage

A file system *block* is the smallest "container" of physical disk space allocated for data. Each file system block is a contiguous area on the physical disk. The block size of a file system is fixed at the time of its creation. File system size depends on block size and the total number of blocks of data stored. A file can span multiple file system blocks because most files are larger than the predefined block size of the file system. File system blocks cease to be contiguous (i.e., become fragmented) when new blocks are added or deleted. Over time, as files grow larger, the file system becomes increasingly fragmented.

Figure 2-12 shows the following process of mapping user files to the disk storage subsystem with an LVM:

1. Files are created and managed by users and applications.
2. These files reside in the file systems.
3. The file systems are then mapped to units of data, or file system blocks.
4. The file system blocks are mapped to logical extents.
5. These in turn are mapped to disk physical extents either by the operating system or by the LVM.
6. These physical extents are mapped to the disk storage subsystem.

If there is no LVM, then there are no logical extents. Without LVM, file system blocks are directly mapped to disk sectors.

The *file system tree* starts with the *root directory*. The root directory has a number of *subdirectories*. A file system should be mounted before it can be used.

The system utility `fsck` is run to check file system consistency in a UNIX host. An example of the file system in an inconsistent state is when the file system has outstanding changes and the computer system crashes before the changes are committed to disk. At the time of booting, the `fsck` command first checks for consistency of file systems critical for a successful boot. If the critical file systems are found to be consistent, the command checks the consistency of all other file systems. If any file system is found to be inconsistent, it is not mounted. The inconsistent file system can sometimes be repaired automatically by the `fsck` command or may require user interaction for, and confirmation of, corrective actions to be taken.

A file system can be either a journaling file system or a nonjournaling file system. *Nonjournaling file systems* create a potential for lost files because they may use many separate writes to update their data and metadata. If the system crashes during the write process, metadata or data may be lost or corrupted.

When the system reboots, the file system attempts to update the metadata structures by examining and repairing them. This operation takes a long time on large file systems. If there is insufficient information to recreate the desired or original structure, files may be misplaced or lost, resulting in corrupted file systems.

A *journaling file system* uses a separate area called a *log,* or *journal*. This journal may contain all the data to be written (*physical journal*), or it may contain only the metadata to be updated (*logical journal*). Before changes are made to the file system, they are written to this separate area. Once the journal has been updated, the operation on the file system can be performed. If the system crashes during the operation, there is enough information in the log to "replay" the log record and complete the operation. Journaling results in a very quick file system check because it only looks at the active, most recently accessed parts of a large file system. In addition, because information about the pending operation is saved, the risk of files being lost is reduced.

A disadvantage of journaling file systems is that they are slower than other file systems. This slowdown is the result of the extra operations that have to be performed on the journal each time the file system is changed. However, the much shortened time for file system checks and the file system integrity provided by journaling far outweighs this disadvantage. Nearly all file system implementations today use journaling.

Dedicated file servers may be installed to manage and share a large number of files over a network. These file servers support multiple file systems and they use file sharing protocols specific to the operating system — for example, NFS and CIFS. These protocols are detailed in Chapter 7.

2.5.5 Application

An *application* is a computer program that provides the logic for computing operations. It provides an interface between the user and the host and among multiple hosts. Conventional business applications using databases have a three-tiered architecture — the application user interface forms the front-end tier; the computing logic forms, or the application itself is, the middle tier; and the underlying databases that organize the data form the back-end tier. The application sends requests to the underlying operating system to perform read/write (R/W) operations on the storage devices. Applications can be layered on the database, which in turn uses the OS services to perform R/W operations to storage devices. These R/W operations (I/O operations) enable transactions between the front-end and back-end tiers.

Data access can be classified as block-level or file-level depending on whether the application uses a logical block address or the file name and a file record identifier to read from and write to a disk.

Block-Level Access

Block-level access is the basic mechanism for disk access. In this type of access, data is stored and retrieved from disks by specifying the logical block address. The block address is derived based on the geometric configuration of the disks. Block size defines the basic unit of data storage and retrieval by an application. Databases, such as Oracle and SQL Server, define the block size for data access and the location of the data on the disk in terms of the logical block address when an I/O operation is performed.

File-Level Access

File-level access is an abstraction of block-level access. File-level access to data is provided by specifying the name and path of the file. It uses the underlying block-level access to storage and hides the complexities of logical block addressing (LBA) from the application and the DBMS.

OBJECT-LEVEL ACCESS

Object-level access is an intelligent evolution of data access whereby objects are the fundamental unit for data storage and access from storage. Data belonging to certain objects is grouped or organized and identified with a unique object identifier. Applications use this identifier to store and retrieve the data.

2.6 Application Requirements and Disk Performance

The analysis of application storage requirements conventionally begins with determining storage capacity. This can be easily estimated by the size and number of file systems and database components that will be used by applications. The application I/O size and the number of I/Os the application generates are two important measures affecting disk performance and response time.

Consequently, the storage design and layout for an application commences with the following:

1. Analyzing the number of I/Os generated at peak workload
2. Documenting the application I/O size or block size

The block size depends on the file system and database on which the application is built. Block sizes in a database environment are controlled by the underlying database engine and the environment variables set.

Consider an example of a SCSI controller (SCSI interface) with a throughput of 160 MB/s and disk service time $R_S = 0.3$ ms. The computation of the rate (1 / [R_S + Transfer time]) at which I/Os are serviced in a typical database I/O with block sizes 4 KB, 8 KB, 16 KB, 32 KB, and 64 KB are shown in Table 2-1. The rate at which the application I/Os are serviced is termed I/Os per second (IOPS).

Table 2-1: Maximum IOPS Performed by SCSI Controller

BLOCK SIZE	TRANSFER TIME (MS)	IOPS= 1/(R_S + TRANSFER TIME)
4 KB	4 KB / 160 MB = 0.025	1 / (0.3 + 0.025) = 3,076
8 KB	8 KB / 160 MB = 0.05	1 / (0.3 + 0.05) = 2,857
16 KB	16 KB / 160 MB = 0.1	1 / (0.3 + 0.1) = 2,500
32 KB	32 KB / 160 MB = 0.2	1 / (0.3 + 0.2) = 2,000
64 KB	64 KB / 160 MB = 0.4	1 / (0.3 + 0.4) = 1,428

As a result, the number of IOPS per controller depends on the I/O block size and ranges from 1,400 (for 64 KB) to 3,100 (for 4 KB).

The disk service time (R_S) is a key measure of disk performance; R_S along with disk utilization rate (U) determines the I/O response time for applications.

As shown earlier in this chapter, the total disk service time (R_S) is the sum of seek time (E), rotational latency (L), and the internal transfer time (X):

$$R_S = E + L + X$$

E is determined based on the randomness of the I/O request. L and X are measures provided by disk vendors as technical specifications of the disk. Consider an example with the following specifications provided for a disk:

- Average seek time of 5 ms in a random I/O environment, or E = 5 ms.
- Disk rotation speed of 15,000 rpm — from which rotational latency (L) can be determined, which is one half of the time taken for a full rotation or L = (0.5 / 15,000 rpm expressed in ms).
- 40 MB/s internal data transfer rate, from which the internal transfer time (X) is derived based on the block size of the I/O — for example, an I/O with a block size of 32 KB or X = 32 KB / 40 MB.

Consequently, R_S = 5 ms + (0.5 / 15,000) + 32 KB / 40 MB = 7.8 ms.

The maximum number of I/Os serviced per second or IOPS = 1/ $R_{S.}$

In other words, for an I/O with a block size of 32 KB and R_S = 7.8 ms, the maximum IOPS will be 1 / (7.8×10^{-3}) = 128 IOPS.

Table 2-2 lists the maximum IOPS that can be serviced for different block sizes.

Table 2-2: Maximum IOPS Performed by Disk Drive

BLOCK SIZE	RS = E+L+X	IOPS = 1/RS
4 KB	5 ms + (0.5 / 15,000 rpm) + 4K / 40MB = 5 + 2 + 0.1 = 7.1	140
8 KB	5 ms + (0.5 / 15,000 rpm) + 8K / 40MB = 5 + 2 + 0.2 = 7.2	139
16 KB	5 ms + (0.5 / 15,000 rpm) + 16K / 40MB = 5 + 2 + 0.4 = 7.4	135
32 KB	5 ms + (0.5 / 15,000 rpm) + 32K / 40MB = 5 + 2 + 0.8 = 7.8	128
64 KB	5 ms + (0.5 / 15,000 rpm) + 64K / 40MB = 5 + 2+ 1.6 = 8.6	116

Table 2-2 shows that the value of E, the seek time, is still the largest component of R_S. It contributes 5 ms out of 7.1 ms to 8.6 ms (58–70 percent) of the disk service time.

The IOPS ranging from 116 to 140 for different block sizes represents the IOPS that could be achieved at potentially very high levels of utilization (close to 100 percent). As shown in Section 2.4, the application response time, R, increases with an increase in utilization or the IOPS.

For the example in Table 2-2, the I/O response time (R) for an I/O with a block size of 64 KB will be approximately 215 ms when the controller works close to 96 percent utilization, as shown here:

$$\begin{aligned} R &= R_S \ / \ (1- \ U) \\ &= 8.6 \ / \ (1-0.96) \\ &= 215 \text{ ms} \end{aligned}$$

If an application demands a faster response time, then the utilization for the disks should be maintained below 70 percent, or the knee of the curve, after which the response time increases exponentially.

Considering another example from Table 2-2, with a block size of 8 KB and the utilization near 100 percent, a maximum of 139 IOPS could be achieved. However, at U = 97 percent the IOPS in the system will be approximately 134, and the response time, R, for each I/O will be 240 ms.

Similarly, the response time for 105 IOPS (U = 75%) will be 29.5 ms; for 75 IOPS (54% utilization), it will be 15.7 ms; and for 45 IOPS (32% utilization), it will be 10.6 ms.

Considering a block size of 64 KB, at U = 100 percent, the maximum IOPS that can be sustained will be 116. The application will not be able to sustain 135 IOPS for the block size and performance specifications used in this example. For 64KB I/O at 105 IOPS (U = 91%), R = 88.6 ms, and at 45 IOPS (U = 39%), R = 14.6 ms.

Storage requirements for an application are specified in terms of both capacity and the IOPS that should be met for the application. If an application needs 200GB of disk space, then this capacity could currently be provided with a single disk. However, if the application I/O demands are high, then it will result in performance degradation because one disk cannot provide the required response time for I/O operations.

The total number of disks required (N) for an application is computed as follows:

If *C* is the number of disks required to meet the capacity and *I* is the number of disks required for meeting IOPS, then

$$N = \text{Max}\ (C, I)$$

Disk vendors publish the disk potential in terms of IOPS based on the benchmark they carry out for different block sizes and application environments.

Consider an example in which the capacity requirements for an application are 1.46 TB. The peak workload generated by the application is estimated at 9,000 IOPS. The vendor specifies that a 146 GB, 15,000-rpm drive is capable of a maximum of 180 IOPS (U = 70%).

In this example, the number of disks required to meet the capacity requirements will be only 1.46 TB / 146 GB = 10 disks. To meet 9,000 IOPS, 50 disks will be required (9,000 / 180). As a result, the number of disks required to meet the application demand will be Max (10, 50) = 50 disks.

In many application environments, more disks are configured to meet the IOPS requirements than to meet the storage capacity requirements. For response-time-sensitive applications, the number of drives required is also calculated based on the IOPS that a single disk can sustain at less than 70 percent utilization level to provide a better response time.

Summary

This chapter detailed the storage system environment — the host, connectivity, and storage. The data flows from an application to storage through these components. Physical and logical components of these entities affect the overall performance of the storage system environment.

Storage is the most important component in the storage system environment. The hard disk drive (HDD) is the most popular storage device that uses magnetic media for accessing and storing data for performance-intensive applications. Logically, the HDD can be viewed in sectors, tracks, and cylinders, which form the basis of disk addressing. This chapter detailed the fundamental laws that govern HDD performance. Overall performance depends on disk response time, which consists of seek time, rotational latency, and disk service time.

Modern disk storage systems use multiple disks and techniques such as RAID to meet the capacity and performance requirements of applications, as described in the next chapter.

EXERCISES

1. What are the benefits of using multiple HBAs on a host?
2. An application specifies a requirement of 200 GB to host a database and other files. It also specifies that the storage environment should support 5,000 IOPS during its peak processing cycle. The disks available for configuration provide 66 GB of usable capacity, and the manufacturer specifies that they can support a maximum of 140 IOPS. The application is response time sensitive and disk utilization beyond 60 percent will not meet the response time requirements of the application. Compute and explain the theoretical basis for the minimum number of disks that should be configured to meet the requirements of the application.
3. Which components constitute the disk service time? Which component contributes the largest percentage of the disk service time in a random I/O operation?
4. Why do formatted disks have less capacity than unformatted disks?
5. The average I/O size of an application is 64 KB. The following specifications are available from the disk manufacturer: average seek time = 5 ms, 7,200 RPM, transfer rate = 40 MB/s. Determine the maximum IOPS that could be performed with this disk for this application. Taking this case as an example, explain the relationship between disk utilization and IOPS.
6. Consider a disk I/O system in which an I/O request arrives at the rate of 80 IOPS. The disk service time is 6 ms.
   a. Compute the following:

      Utilization of I/O controller

      Total response time

      Average queue size

      Total time spent by a request in a queue
   b. Compute the preceding parameter if the service time is halved.
7. Refer to Question 6 and plot a graph showing the response time and utilization, considering 20 percent, 40 percent, 60 percent, 80 percent, and 100 percent utilization of the I/O controller. Describe the conclusion that could be derived from the graph.
8. The Storage Networking Industry Association (SNIA) shared storage model is a simple and powerful model for describing the shared storage architecture. This model is detailed at `www.snia.org/education/storage_networking_primer/shared_storage_model/SNIA-SSM-text-2003-04-13.pdf`. Study this model and prepare a report explaining how the elements detailed in this chapter are represented in the SNIA model.