APPENDIX B

Disk Drive Module Implementation Details

Secondary storage performance is often dominated by disk drive behavior. This appendix describes the simulator's disk drive module, which accurately models mechanical positioning latencies, zoned recording, spare regions, defect slipping and reallocation, on-board buffers and caches, prefetch, fast write, bus delays, command queueing and request processing overheads. The first section of this appendix describes modern disk drives, providing both a short tutorial on the subject and insights into how the various components of a disk can be modeled. The second section describes the organization and operation of the disk drive module.

B.1 Modern Disk Drives

This section describes the main performance characteristics of modern disk drives. In many cases, approaches to modeling specific disk attributes are also discussed. The description focuses on common characteristics of disk drives conforming to the popular SCSI bus protocol [SCSI93]. Much of the discussion also applies to drives using other peripheral interfaces.

A disk drive can be viewed as two distinct parts, one physical and one logical. The physical part consists of the media recording and accessing components. The logical part consists of the on-board control logic that interprets requests, manages internal resources and improves the efficiency of media access.

B.1.1 Media Recording and Accessing

Mechanical Components

A disk drive (see figure B.1) consists of a set of **platters** rotating on a common axis. Both **surfaces** of each platter are coated with magnetic media. Each surface has an associated **read/write head**. In most drives, only one read/write head is active at any given time. The heads are mounted on disk **arms** that are ganged together on a common **actuator**.

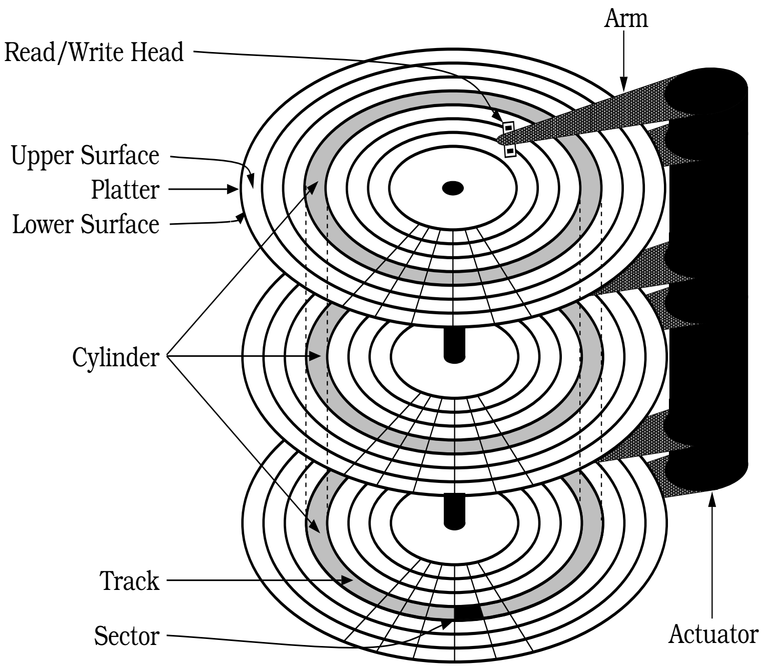


Figure B.1: Disk Drive Internals.

Data Layout

The minimum unit of data storage is a **sector**, which typically holds 512 bytes of data. Sectors are arranged in concentric circles, called **tracks**, on each surface.[1] A **cylinder** is a set of tracks (one from each surface) equidistant[2] from the center of the disk.

To exploit the larger circumference of outer tracks, a modern disk usually partitions its set of cylinders into multiple **zones**, or bands. The number of sectors per track increases with the radius of the tracks in the zone (see figure B.2). This increases both the capacity per unit volume and the mean and peak media transfer rates. However, it requires more powerful electronics to manage the varied recording/accessing activity. Also, additional on-board logic resources are needed to handle the more complex data layout.

Read/Write Head Positioning

Mechanical positioning delays often dominate disk request service times. To access a specific media location, a disk actuator must first **seek** to the target cylinder and activate the appropriate read/write head, possibly requiring a **head switch**. The read/write head then waits for the target sectors to rotate into position. After this **rotational latency**, the media transfer occurs as the target sectors rotate under the read/write head. Additional mechanical delays (e.g., head switches and/or seeks) result when multiple tracks or cylinders must be accessed.

---

[1] This data layout differs from the spiral layout used with compact disks and vinyl records.

[2] "Equidistant," in this case, refers to the per-surface track numbering. Thermal variations and other physical phenomenon may cause the Nth track on two surfaces to not be exactly the same distance from the center.

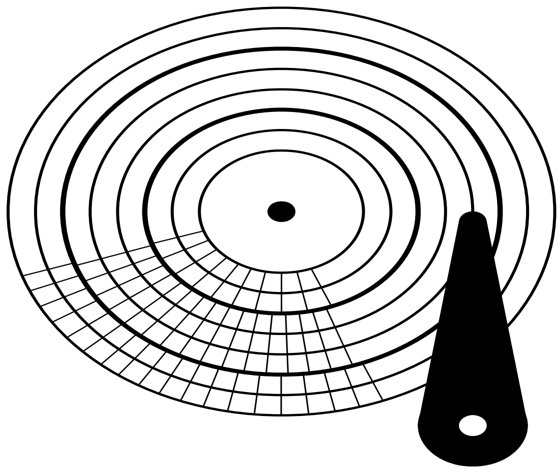


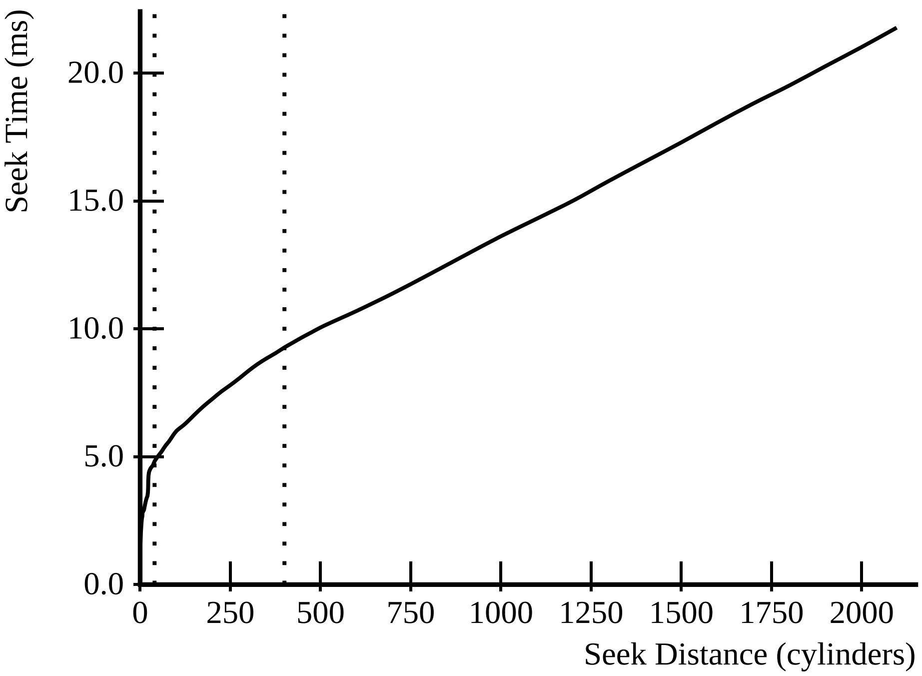
Figure B.2: Top View of a Disk Surface with 3 Zones. The innermost and outermost zone each contain 2 cylinders. The middle zone contains three.

A seek is comprised of four phases: (1) acceleration, (2) coast, (3) deceleration, and (4) settle. The disk arm accelerates until either the maximum velocity is reached or half of the seek distance has been traversed. If necessary, the disk arm coasts for some distance at its maximum velocity. The disk arm then decelerates, coming to rest near the destination track. Finally, the actuator electronics make fine adjustments to the disk arm position to **settle** precisely onto the track.[3] Very short seeks may be handled entirely by the settling electronics, without utilizing the full power of the actuator motor. Before writing data to the media, the electronics will often take additional **write settle time** to more accurately position to the center of the track. Data can be read with the read/write head slightly misaligned but writing data off-center invites cross-talk between tracks, complicating or even preventing later access.

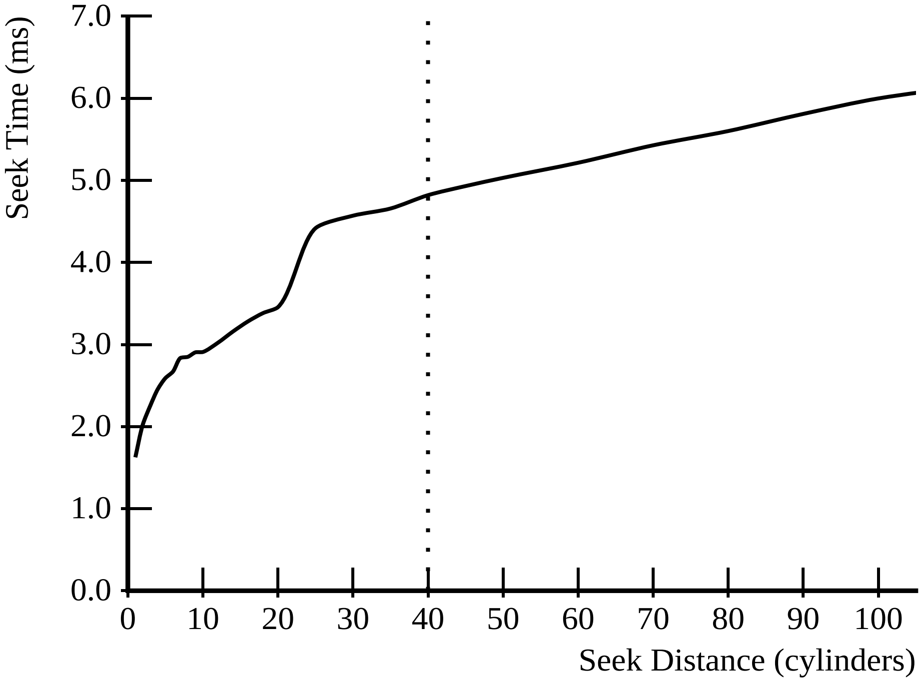
A **seek curve** maps seek distances to actuator positioning times. In reality, the seek time between two tracks is a complex function of many variables, including the positions of the cylinders (i.e., their distances from the spindle), the direction of the seek (inward or outward), various environmental factors (e.g., vibration and thermal variation), the type of servo information (e.g., dedicated, embedded or hybrid), the mass and flexibility of the head/arm assembly and the current available to the actuator. However, a single curve can effectively reflect mean values for seek times. Typical seek curves (e.g., see figure B.3) consist of an irregular initial region (caused by variations in how the actuator electronics optimize specific seek distances), a middle region approximated by a square root function (caused by the acceleration/deceleration phases), and a large linear region for long seeks. A simulator can use a seek curve directly or in the form of some fitted function. The additional write settle time is added when appropriate.

A head switch consists of electronically disabling the input/output signals of the active read/write head, enabling those of another and settling onto the new track. The settling is required because the radii of the two corresponding tracks may not match exactly (e.g., due

[3]As the platters rotate, the same track-following electronics are used to constantly fine-tune the read/write head position.



(a) Full seek curve



(b) Expanded view of short seeks

Figure B.3: Measured seek curve for a Seagate ST41601N disk drive.

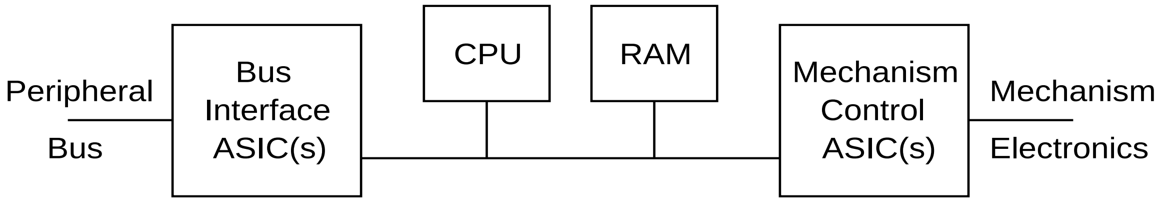


Figure B.4: On-Board Disk Drive Logic.

to thermal variation). The head switch time depends largely on the type of servo information utilized by the drive. Also, write settle time is required before writing data. When both a seek and a head switch are necessary, they usually occur concurrently. So, the combined delay is the maximum of the two.

The rotation speed, usually measured in rotations per minute (RPM), directly affects both the rotational latency and the media transfer rate. Generally, disk specifications indicate that the rotation speed varies slightly (e.g., $\pm 0.5\%$) both among disks in a line of drives and over time for a given disk. Both forms of variation are generally small enough that they can be safely ignored in a disk model. However, the initial rotational offset of each drive in a simulation should be determined separately (e.g., by a uniform distribution), unless the the drives are intended to be rotationally synchronized (e.g., synchronous disk interleaving [Kim86]).

B.1.2 On-Board Control Logic

Figure B.4 shows the main on-board logic components used in modern disk drives. The CPU, which is often an embedded version of a previous generation commodity microprocessor (e.g., Intel 80386 or Motorola 68030), executes firmware to manage the disk's resources and handle requests. The firmware is often a simple operating system that provides process structures, interrupt service routines and simple memory and lock management. Like main memory in a computer system, the on-board memory is logically (and sometimes physically) partitioned into working space for the firmware and a staging area for the disk blocks moving to/from the media. Mechanism control ASIC(s) interact directly with the mechanical and magnetic disk components discussed above. Bus interface ASIC(s) interact directly with the peripheral bus, handling the details of arbitration, signal generation and interpretation, parity generation/checking, and so forth. The on-board logic components communicate via some interconnection, which is not necessarily a common bus.

The firmware executing on the CPU manages the other components. The various policies and functions of the firmware are discussed below.

External Interface Management

Most modern disk drives conform to a standardized, high-level peripheral bus[4] protocol (e.g., SCSI or IPI). A host or intermediate controller issues a request to a disk drive in terms of a starting **logical block number** (LBN) and a total request size. The details of the subsequent media access are hidden from the host or controller, offloading the management overhead associated with actual data storage.

The bus transfer rate and bus control delays are functions of the disk drive, the interconnect and the other participant in the communication (e.g., intermediate controller). A detailed disk performance model should account for the timings of control message exchange and data transfers to/from external components. This requires that the interconnect and other subsystem components be modeled appropriately.

From a performance standpoint, the four main bus interactions are request arrival, request completion, data transfer, and disconnect/reconnect. To issue a request to a disk drive, the controller establishes a connection to the disk and sends the appropriate message. When a request has been serviced, the disk establishes a connection and informs the controller. When the disk is ready, it initiates the data transfer to/from the controller.[5] Disconnect/reconnect actions can dramatically improve the effective utilization of a shared interconnect. With the SCSI protocol, components arbitrate for bus ownership and must explicitly give it up. Without disconnect/reconnect, the bus would be held for the duration of each request's service, preventing concurrent activity by other disks on the same bus. When a disk does not need the bus (e.g., during mechanical positioning) it can **disconnect** from the bus, allowing other components to use it. The disk later **reconnects** (i.e., re-arbitrates for bus ownership and re-establishes communication) when it needs to communicate with the controller.

In normal circumstances, a disk that has disconnected will reconnect for three reasons: (1) An outstanding write request has been serviced and the disk needs to report completion. (2) The disk is ready to transfer read request data to the controller. (3) The disk is ready to receive write request data from the controller. For SCSI disk drives, the decision as to when a disk is "ready" for bus data transfers (the latter two reasons above) is governed by two watermark values. The *Buffer Full Ratio* determines how much buffered (cached) read data should be available before reconnecting. The *Buffer Empty Ratio* determines how much dirty write data should remain before reconnecting (if additional write data need to be transferred). These ratios are usually specified as fractions of either the cache segment size (see below) or the request size. Poor settings can hurt performance by causing too many disconnect/reconnect cycles (there is overhead involved with each) or reducing the overlap between bus and media transfers. Some disks dynamically adjust the buffer full and empty ratios based on observed media and bus rates to avoid these problems.

---

[4]The peripheral interconnect may not necessarily be a shared bus. For example, some computer systems employ point-to-point serial connections. The physical aspects of the interconnect are not particularly important to performance modeling of disk drives, so long as the various delays and transfer rates are properly emulated by the bus module. The topology of the interconnect is an important but external issue (from the disk drive's point of view).

[5]Usually, the data are transferred in logically ascending order. However, some protocols allow blocks of data to be transferred in non-ascending order when it is more convenient for the disk.

Request Processing

The request processing firmware for a modern disk drive is large and complex. From a performance modeling standpoint, however, the two main aspects of this firmware are the policies and the processing times. The various policies are described under the other headings in this subsection. The request processing times can be an important part of the service time, even though they are generally dominated by mechanical positioning delays.

Initial request processing overheads (i.e., when the request arrives at the disk) can depend on several factors, including the request type (e.g., read or write), the request size, the state of the on-board cache (e.g., hit or miss) and the immediately previous request's characteristics[6]. Much of this initial overhead is visible externally because the controller passes bus ownership (implicitly) to the disk and the disk does not respond (e.g., by disconnecting or transferring data) until after some initial request processing. If the request is a read hit or a write, the data will usually be transferred to/from the controller immediately (i.e., without disconnecting). There are also externally visible processing delays associated with disconnect, reconnect and completion actions. Externally visible delays are particularly important when the bus is shared by more than one disk, since bus contention may occur. Request processing overheads that are not externally visible are also important, but can often be grouped with other service time components, such as seek times.

Logical-to-Physical Data Mappings

The traditional mapping of logical block numbers to physical media locations (i.e., cylinder, surface and sector) starts with logical block zero on the outermost (or innermost) cylinder. Logical block numbers are assigned around each track of the cylinder before moving to the next adjoining cylinder. This process repeats until the entire disk's capacity has been mapped. Several policies used in modern disk drive firmware disrupt the **LBN-to-PBN** mapping by withholding parts of the physical media for other purposes or shifting the mapping for performance purposes.

Many disk drives reserve a small fraction of the storage capacity for private use by the firmware. Generally, this reserved space is located at the beginning or end of a zone.

Because switching between surfaces and seeking between adjacent cylinders requires nonzero time, the first logical block of each track or cylinder is offset from the first logical block of the previous track or cylinder. This prevents a request that crosses a track or cylinder boundary from "just missing" the next logical block and waiting almost a full rotation for it to come around again. The **track skew** is the offset used when moving to the next track within a cylinder. The **cylinder skew** is the offset used when moving from the last track of one cylinder to the first track of the next. These values are specified as an integral number of sectors and therefore differ from zone to zone.

Disk drives can survive the loss of hundred of sectors or tracks to media corruption. In fact, a new drive usually contains a list of defects identified at the factory. Defects **grown** during a disk's lifetime are added to this list and removed from the LBN-to-PBN mapping. The firmware reserves **spare regions** at the end of certain tracks, cylinders or

[6] From empirical observation, it appears that disk drive firmware will sometimes "prime" itself to reduce command processing delays for common sequences of request types (e.g., sequential reads).

zones to replace defective media locations. The defect management firmware compensates for corrupt media location in one of two ways, slipping or relocation. Sector or track **slipping** occurs only during disk format. In the case of sector slipping, the format process skips each defective sector and adjusts the LBN-to-PBN mapping so that the sectors on either side are logically sequential. Track slipping follows the same routine, except that an entire track is skipped if it contains any defective media. With sector or track slipping, each defect changes the mapping of all subsequent logical blocks up to the next spare region. Sector or track **reallocation** occurs when a disk discovers defective media during normal use. The defect management firmware immediately remaps the corresponding LBN(s) to a spare region and directs subsequent accesses to the new location(s).

Because several layout and mapping characteristics vary from zone to zone, it is useful to partition the LBN-to-PBN mapping process into two parts. First, the correct zone is identified. Then, the physical location within the zone is identified. The first step can be realized with knowledge of the first logical block number of each zone. The second requires several zone-specific values, including the first cylinder number, the number of sectors per track, the track and cylinder skews, the amount and location of the reserved space and the physical sector that is assigned the first logical block number[7]. Given these values and information about spare regions and defect management, the mapping follows the traditional approach.

On-Board Cache

As mentioned above, part of the on-board memory serves as a staging area for disk blocks. In modern disk drives, this staging area acts both as a speed-matching buffer (between the bus and the media) and as a disk block cache. As a speed-matching buffer, the memory allows the firmware to overlap the transfers over the bus and to or from the media. As a disk block cache, the memory can improve effective disk service times. A request that can be satisfied by the cache avoids lengthy mechanical positioning delays. The service time for such a request can be more than an order of magnitude less than if media access were required. Modern disk drives use a wide variety of cache management policies, many of which are described below. Although the policies used by a given disk are rarely outlined in publically available specifications, there are approaches to identifying them empirically [Worthington95].

Most disks partition the available memory into multiple cache lines, referred to as **segments**, to better service multiple streams of sequential requests. Commonly, each segment is the same size. In some disks, this size matches the number of sectors per track. In most disks, however, there is no such relationship. The most aggressive implementations analyze request patterns and dynamically modify the number of cache segments (and thereby the segment size) to improve hit rates.

During service, each request is attached to a cache segment. When a new request arrives, the cache is scanned for any overlap. For read requests, an overlap indicates a partial or full

[7] In some disk drives, the physical sector that is assigned the first logical block varies from zone to zone in non-intuitive ways. Certainly, a **zone skew** would be useful for requests that cross zone boundaries, particularly when there is reserved space at the beginning or end of adjacent zones. Empirically, however, these values do not always correlate with the inter-zone seek times.

hit. For a full read hit, the data blocks are simply transferred from the cache to the controller. For a partial read hit, the firmware may utilize the data sectors or ignore them. For write requests, overlapping data blocks (possibly only the overlaps, possibly entire segments) are invalidated in the cache. The exception to this occurs with write-back caches (see below). For a cache miss, the firmware selects one of the segments, invalidate its contents and attaches it to the request. The segment replacement policy is often *Least-Recently-Used*. Some disks only allow write requests to use a subset of the segments and may prevent read requests from using that subset.

To improve the cache hit rate for sequential access patterns, most modern disk drives prefetch data after read requests. Specifically, the firmware continues reading sequentially numbered media blocks beyond the end of a read request. There are several possible stop conditions. Most disks discontinue prefetch to service a new request arrival or another pending request (for command queued disks; see below). Although the prefetch may be stopped when it reaches the end of a track or cylinder, most disks aggressively prefetch beyond these boundaries even though a new request might have to wait for the head switch or seek to complete. Some disks are configured with a maximum prefetch size. Finally, prefetch stops when the cache segment is full.

Disks with **read-on-arrival** and/or **write-on-arrival** capability can begin transferring data from or to the media as soon as any seek or head switch activity completes. That is, media sectors are transferred from or to the media in the order that the read/write head encounters them rather than in strictly ascending LBN order. This essentially eliminates rotational latency for full track requests,[8] but is less effective for smaller or larger requests. Read-on-arrival can also increase the effectiveness of prefetch for heavy workloads. Most modern disk drives do not currently support these options.

Some disks support the ability to signal write request completion as soon as all data blocks reach the on-board cache. The firmware moves the data from the cache to the media at some later point. Because the on-board memory of most disk drives is volatile (i.e., the contents are lost whenever power fails), such **fast writes** reduce the reliability of disk drive storage.[9]. Modern disks can employ three options to reduce the probability of data corruption due to improper update ordering.[10] One option is to disable fast write entirely. A second option, used in HP-UX [Ruemmler93], is to selectively disable fast write for each request that may require ordering with respect to a later request. This requires that the software executing on host systems recognizes that a disk uses fast write and tags each request appropriately. A third option is to always perform fast writes in arrival order. Many disks do not use fast write for a second consecutive write request until the first's data have been written to the disk, unless the new request is immediately sequential. If it is sequential, then it is combined with the previous request and reported as complete.

---

[8] For this reason, read-on-arrival and write-on-arrival are sometimes referred to as **zero-latency** read and write.

[9] In the near future, disk drives are likely to be equipped with non-volatile cache memory to eliminate this problem.

[10] Some applications and system software components rely on careful ordering of permanent storage updates to protect the data integrity. For example, databases add entries to a write-ahead log to achieve an atomic update to multiple records. Also, many file systems require ordered updates to maintain metadata integrity [Ganger94].

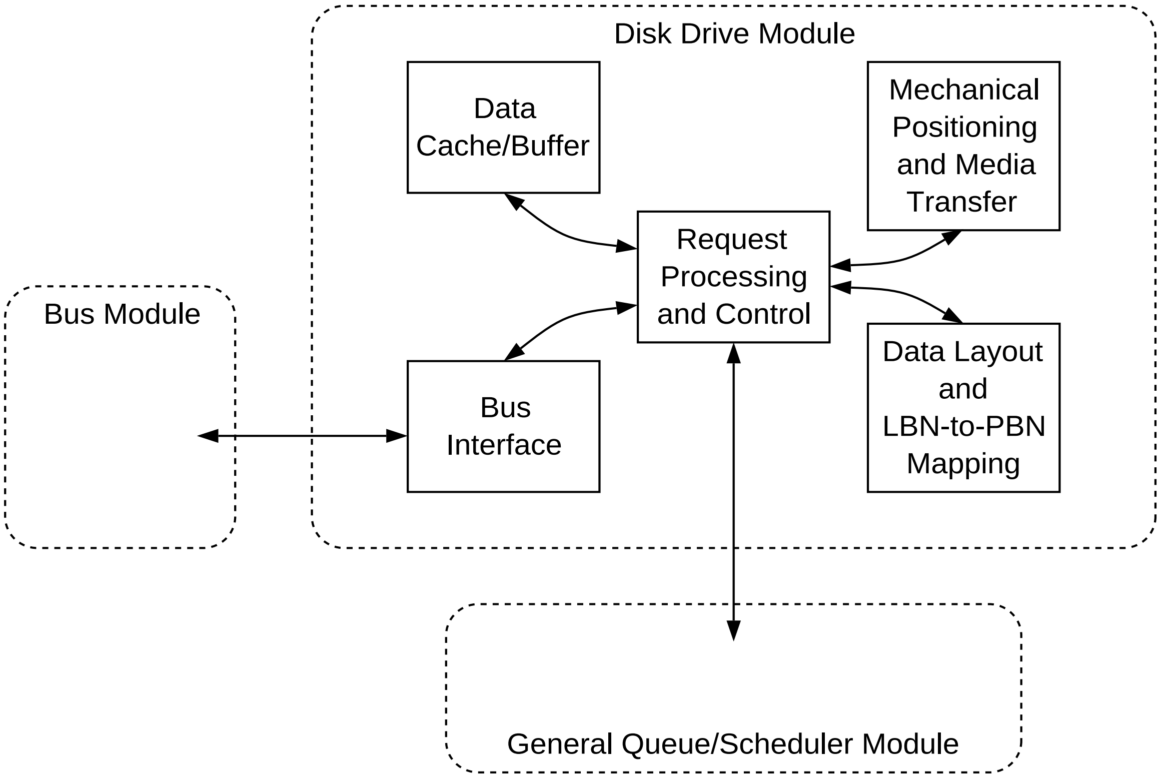


Figure B.5: Disk Drive Module.

Command Queueing

Most modern disk drives support **command queueing**, whereby some number (e.g., 2-64) of requests can be outstanding at a time. Command queueing has several benefits. First, an on-board scheduler can have an advantage over external schedulers because it generally has more accurate information about the current read/write head position and various positioning delays. The SCSI protocol provides for optional ordering restrictions, such as *Head-Of-Queue* and *Ordered* requests [SCSI93]. Second, the firmware can exploit inter-request concurrency by decoupling the command processing, bus transfer, and media access components of request service. Of course, there are dependencies between these components (e.g., write data cannot be transferred to the media before it is received from the host or controller), but the disk can still exploit concurrency between separate requests [Worthington95a].

B.2 Disk Drive Module

The disk drive module models all of the aspects of modern disk drives described in section B.1. It is organized as five submodules (see figure B.5) and uses instances of the general queue/scheduler module (see section A.1.2). This section describes the five submodules and their interactions.

Request Processing and Control

This submodule controls the flow of work in the disk module. It uses the other four submodules to accomplish this task. All of the disk service time components are decided here and performed using the general infrastructure described in section A.1.1. Any request concurrency (e.g., overlapping bus and media transfers) is also determined and handled by this submodule. Instances of the general queue/scheduler module are used to maintain the list of outstanding requests and make scheduling decisions. Most of the policies, for both this submodule and the instances of the queue/scheduler module, are configured by input parameters.

Data Cache/Buffer

This submodule maintains the state of the on-board cache segment(s). The control submodule uses it to determine whether requests hit (partially or entirely) or miss in the cache and to select a segment for staging a request's data. The control submodule makes all decisions regarding how to deal with cache hits and transfers into and out of a segment, simply informing the cache/buffer submodule when the contents change. The segment selection and replacement policies are configured by initialization parameters.

Mechanical Positioning and Media Transfer

This submodule maintains state regarding the position of the active read/write head. Also, the control submodule uses this submodule to determine how long any given mechanical positioning delay or media transfer will take. When appropriate, the control submodule informs this submodule that the current position should be updated. The current position state consists of the current cylinder, the current surface, the last known rotational offset and the time at which the offset held. This state is sufficient to determine the exact read/write head position at any point in simulated time (assuming that the rotation speed is constant). The rotation speed and various mechanical delay components are all input parameters.

Data Layout and LBN-to-PBN Mapping

This submodule simply translates logical block numbers to physical media locations (or the reverse). All of the various data layout and mapping characteristics are input parameters.

Bus Interface

This submodule interacts with the bus module. The control submodule uses this submodule to communicate with external components. The control submodule determines when ownership should be established/relinquished and when/which messages and data should be transferred.