HEWLETT PACKARD

Modelling Disks

Chris Ruemmler, John Wilkes
Computer Systems Laboratory

HPL–93–68
July, 1993

disk drives, modelling, trace-driven simulation, SCSI disk performance

As processor speeds continue to increase rapidly, I/O is in danger of becoming a performance bottleneck. As a result, many researchers have started to look at ways to improve overall I/O performance—a large component of which is the performance of the disk itself. Many of these studies use analytical or simulation models to compare alternative approaches, and the quality of these models determines the quality of the conclusions: the wrong simplifying assumptions can even lead to erroneous conclusions. Unfortunately, the level of quality of the models used in practice varies widely—at least in part because the data needed to build a good one has not been readily available.

This paper makes three main contributions: (1) it provides a description of the current state of the art in disk drive technology and the underlying technology trends, with particular emphasis on those features that are likely to be of value to researchers constructing simulations; (2) it shows how to construct an accurate disk simulation model; and (3) it provides quantitative information on how the different portions of the model contribute to its accuracy. The primary audience for the paper is researchers engaged in I/O system studies, for whom it provides a wealth of information to assist them in producing more accurate simulation models with which to test out their ideas. It should also be of interest to anybody who would like to understand modern disk drive technology, and trends in its evolution.

Also published as Operating Systems Research Department report HPL–OSR–93–29.

1 Introduction

Modern microprocessors technology is advancing at an incredible rate: speedups of 40–60% compounded annually have become the norm. Although disk storage densities are also improving at a similarly impressive rate (60–80% compounded annually), performance improvements have been occurring rather more slowly (about 7–10% compounded annually over the last decade). As a result, the performance of the disk system is fast becoming a dominant factor in overall system behavior.

One consequence is that many researchers have turned to this area searching for improved algorithms, policies, and approaches to speed up computer systems. Recent examples include disk arrays (RAIDs), file systems (LFS, EFS, FFS), disk scheduling algorithms, and database designs. Almost all of these studies used modelling at some stage in their development; and the quality of those models has had a significant effect on the correctness (and believability) of the results. In one extreme case, the better disk models of [Holland92] reversed some of the findings derived from a simpler one [Muntz90].

Despite this, there has been little work in developing or describing accurate disk models, and—perhaps as a result—the use of simple, relatively inaccurate models is surprisingly commonplace. We show here that this is unnecessary by demonstrating and describing a calibrated, high-quality disk model with 14 times smaller errors than a simple first-order model. The improvement resulting from each of the model's components is described separately, so that an informed trade-off can be made between effort and accuracy by other model builders. In addition, detailed disk drive characteristics are provided for a pair of disk drives, as well as a brief description of a simulation environment that uses the disk model.

Our approach should be useful to a wide range of researchers using simulations to examine the effectiveness of new algorithms in virtual memory, file system, device driver, caching, and other I/O system-related areas.

Paper outline. The remainder of the paper is organized as follows. We begin with a survey of the design and properties of modern disk drives. We then introduce our model, and provide information on how its accuracy improves as a function of each additional component, followed by a survey of recent disk modelling work. We end with a summary of our results, together with some recommendations for other model builders.

2 Characteristics of modern disk drives

A necessary prelude to modelling disks is to understand how they behave. This section of the paper provides an overview of the current state of the art in magnetic disk drives. Our emphasis is on disks that are most likely to be encountered in practice: the increasing competition for market share by disk manufacturers is being pursued within the context of a narrowing range of standard disk interfaces, of which the most common today is probably the Small Computer Systems Interconnect, or SCSI. For simplicity, we restrict our discussion here to Winchester disk drives with embedded SCSI controllers, since these are likely to be the *de facto* standard on most computer systems for the next several years.

Disks contain a *mechanism* and a *controller*. The mechanism is made up of the *recording components* (the rotating disks, the heads that read from and write to them), and the *positioning components* (an arm assembly that moves the heads into the correct position together with a servo system that keeps it in place). The disk controller contains a microprocessor, some buffer memory, and the interface to the SCSI bus. The controller manages the storage and retrieval of data to and from the mechanism, and performs mappings between incoming logical addresses and the physical disk sectors that store the information.

The following sections provide a more detailed overview of each of these elements in turn, with particular emphasis on those features that need to be considered when creating a disk model. Not all these features are equally important to the accuracy of the model, as will become clear later.

2.1 The recording components

Modern Winchester disks range in size from 1.3" to 8" diameter, with 2.5", 3.5" and 5.25" being the most common sizes today. Smaller disks have less surface area, which means that they can store less data than their larger counterparts, but they are cheaper—and easier—to manufacture, can be spun faster, have smaller seek distances, and thus smaller seek times. Historically, as storage densities increase to the point where 2–3GB can be placed onto a single disk, the next-smaller diameter in the series becomes the most cost-effective—and hence preferred—storage device. Right now, 2.5" disks are the size of choice for most battery-powered applications, and 3.5" disks are in the process of replacing 5.25" disks for mains-powered applications.

Increased storage density results from two improvements. The first is better linear density, which is determined by maximum rate of flux changes that can be recorded and read back—current values are around 50000 bits per inch (2000mm^{-1}), rising to around double this value by the end of the decade.[1] The second is making packing the separate tracks of data more closely together—which is where most of the improvements are occurring. Current values are about 2500 tracks per inch (100mm^{-1}), rising to perhaps 20000 tpi by the end of the decade. The product of these two factors will probably sustain a growth rate of over 60% per year to the end of the decade.

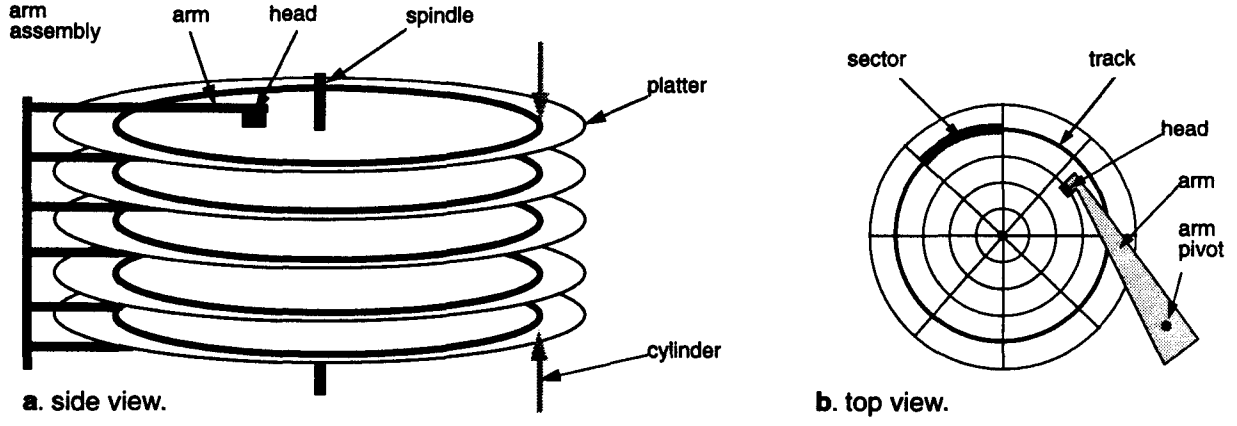


Figure 1: the mechanical components of a disk drive.

1. The disk industry seems to think in terms of inches. Translations to more standard units are provided where appropriate.

A single disk contains one, two, or as many as a dozen *platters*, stacked one above one another, as shown in Figure 1. The platters are very flat, thin disks, usually of aluminum (although glass has been used recently [HPKittyhawk92]). Both the top and bottom surfaces of the platters are coated with a very thin (~50nm) film of magnetic material, topped by about ~25nm of material for lubrication and protection.

The stack of platters rotates in lock step on a central *spindle*. Although 3600 RPM was a de facto standard for many years, spindle rotation speed has increased recently, and speeds of 4000, 5400, 6000 and even 7200RPM are now being used, with the median speed increasing at a compound rate of about 12% per year. A higher spin speed increases transfer rates and shortens rotation latencies (the time for data to rotate under the head), but also increases power consumption and requires better bearings for the spindle.

The spin speed is typically quoted as accurate to ±0.5 or ±1%; in practice, the disks vary their speed slowly around the nominal rate. Although this is perfectly reasonable for the disk's operation, it means that it is essentially impossible to model the disk's rotational position some 100–200 revolutions after the last known operation without some additional information. We return to this point later.

Each platter surface (possibly with the exception of the outermost ones) has an associated *disk head* that is responsible for recording (writing) and later sensing (reading) magnetic flux variations in the thin films on the surface. In active use, a disk head "flies" 3–6μinches (75–150nm) above the surface of the disk, kept there by a combination of external pressure towards the surface and the air between the head and the platter surface. Although decreasing the gap between the surface and the head (the "flying height") increases the storage density and improves the signal-to-noise ratio, it is bounded from below by a combination of platter flatness (large-scale) and surface roughness (small-scale), spindle vibration, temperature and altitude changes, rolling of the head during seeks, and manufacturing and assembly tolerances.

When the disk is powered down, the heads are moved away from the data portion of the disk to a reserved landing zone, where they eventually drop into contact with the platter when the latter stops spinning. This design—called a Winchester disk—allows the head-disk assembly to be hermetically sealed, significantly increasing reliability and resistance to environmental factors such as dust and humidity.

The disk has a single read-write *data channel* that can be switched between the heads.[2] It is responsible for encoding and decoding the data stream into or from a series of magnetic phase changes to be stored on the disk. The encoding schemes used are designed to mask significant amounts of errors: the signal to noise ratio from the head is typically about 25-to-1. The speed of the data channel electronics (especially the preamp close to the head), the encoding scheme used, and the signal-to-noise ratio obtainable from the head, provide a practical bound on the disk data rate. Current speeds are limited to around 100Mbits/s for the raw signal coming off the head, which corresponds roughly to a 100000 bpi linear recording density on a 3.5" platter rotating at 6000 RPM.

---

2. Multi-channel disks can support more than one read/write operation at a time, which means that higher data transfer rates can be achieved. However, these disks are relatively costly because of technical difficulties such as controlling the crosstalk between the concurrently-active channels, and keeping multiple heads aligned on their platters simultaneously, which is becoming harder to do as track densities increase.

2.2 The positioning components

Each data surface is set up to store data in a series of concentric circles, called *tracks*. A single stack of tracks at a common distance from the spindle is called a *cylinder*. A typical 3.5" disk today has around 2000 cylinders. As track densities increase, the notion of vertical alignment that is associated with cylinders becomes less and less relevant: the track alignment tolerances are simply too fine, and the tracks on each platter have to be considered essentially independent of the others.

To access the data stored in a track, the disk head has to be moved over it. This is done by attaching each head to a disk *arm*—a stainless-steel lever that is pivoted near one end on a rotation bearing. Notice that all the disk arms are attached to the same rotation pivot, so that moving one head causes the others to be moved as well. The rotation-pivot gives better immunity to linear shocks than the older scheme of mounting the head on a linear slider. One slight disadvantage is that the angle of the head to the tracks it is following changes between the inside and outside of the disk, which slightly reduces the available recording density at the edges of the platters.

It is the task of the positioning system to ensure that the appropriate head gets to the desired track as quickly as possible, and remains there, even in the face of external vibration and shocks. In addition, it has to cope with irregularities in the circularity of the tracks which are never quite concentric with the spindle as a result of small variations during mounting on the spindle, formatting, and differential thermal expansions.

2.2.1 Seeking

Moving the heads is referred to as *seeking*. The speed at which this happens is limited by the power available for the pivot motor (halving the seek time requires quadrupling the power), and by the arm stiffness: 30–40*g* is required to achieve good seek times, and too flexible an arm is subject to twisting while it is being accelerated, which can bring the head into contact with the platter surface. Smaller disks both decrease the distance the head has to move and reduce the size of the arm, which means that it becomes lighter and easier to stiffen against flexing—allowing higher accelerations and thus shorter seek times.

A seek is composed of the following components:

- *start-up*: the arm is accelerated, typically with constant torque (force) until it reaches half of the seek distance, or a fixed maximum velocity;
- *coast*: for a long seek, the arm will be allowed to coast across most of the distance at its maximum speed;
- *slowdown*: the arm is brought to rest, hopefully exactly on top of the desired track;
- *settle*: the disk controller uses servo positioning information to fine-tune the position of the head over the desired track on the selected surface.

Very short seeks (less than half a dozen cylinders) are dominated by the settle time (1–3ms)—in fact, they may not even perform a seek *per se*, but simply rely on the servo system that manages the settling process to handle it all. Short seeks (less than 200–400 cylinders) spend almost all of their time in the constant-acceleration phases, and so take time proportional to the square root of the seek distance, plus the settle time. Long seeks spend most of their time moving at a constant speed, and take time that is proportional to distance, plus a constant overhead. A full-stroke seek

from one edge of the disk to the other will take 18–25ms. As disks become smaller, and track densities increase, the proportion of the time spent settling increases.

"Average" seek times are commonly used as a figure of merit for disk drives, but they can be misleading. Such averages are calculated in a number of ways. One way is to sum the times to perform one seek of each size, and divide this sum by the number of different seek sizes. A variant is to scale this by the number of possible seeks of each size: thus, there are *(N–1)* different single-track seeks that can be done on a disk with *N* cylinders, but only 1 full-stroke seek. This weights the shorter seek more heavily in the average, which is a better approximation to measured seek distance profiles.

It is possible to show that if disk requests are completely independent of one another, then the average seek distance will be 1/3 of the full stroke. Some sources thus quote the 1/3-stroke seek time as the "average", even though independent seeks are rare in practice: shorter seeks are much more common [Patterson90, Ruemmler93]. Others just divide the full-stroke time by three, and quote that. What matters is the seek-time versus distance profile, rather than the marketing specifications: *caveat emptor*!

The information required to decide how much power to apply to the pivot motor for how long for a particular seek is encoded in tabular form in the disk controller. Rather than storing every possible value, a subset of the total is stored, and interpolation used for intermediate seek distances. The resulting fine-grain seek-time profile can look rather like a sawtooth (see [Patterson90, page 558] for an example).

It is occasionally necessary to recalibrate these tables, as a result of thermal expansion, arm pivot-bearing stickiness, and other factors. Such a recalibration can take 500–800ms. The recalibrations can be triggered by both temperature changes and timers, with the highest rates occurring just after the disk is powered up, reducing to once every 15–30 minutes in steady state conditions. These recalibrations are typically triggered by events independent of the request stream—but the recalibration itself may be delayed until the next I/O request arrives. Obviously, this can cause difficulties with real-time or guaranteed-bandwidth systems (such as multimedia file servers), so some disks are now appearing with modified controller firmware that either avoids these visible recalibrations completely, or allows their execution to be scheduled by the host.

2.2.2 The track-following servo

Fine-tuning the position of the head at the end of a seek, and keeping the head on the desired track, is the function of the track-following servo system. This uses positioning information recorded on the disk at manufacturing time to determine whether the disk head is correctly aligned. Servo information can either be embedded in the target surface or recorded on a separate dedicated surface. The former maximizes capacity (there doesn't need to be an entire surface dedicated to the servo data) so it is most frequently used in disks with small number of platters. As track densities increase, some form of embedded servo data becomes essential—perhaps combined with a dedicated servo surface. However, the embedded servo method alone is not so good at coping with shock and vibration, since feedback information is only available intermittently (e.g., between data sectors, although a few drives will interrupt their data sectors for a servo burst).

The servo system is also used to perform a head-switch: when the controller switches its data channel from one surface to the next in the same cylinder, the new head may need repositioning to accommodate small differences between the alignment of the tracks on the different surfaces. The time taken for such a switch (0.5–1ms) is typically one third to one half of the time taken to do a settle at the end of a seek. Similarly, a track-switch (sometimes called a cylinder switch) occurs when the arm has to be moved from the last track of a cylinder to the first track of the next one. This takes about the same time as an end-of-seek settle. Settling time is increasing as track densities increase, and the associated decoupling between the tracks on different platters means that head-switch times are getting closer to track-switch ones.

Many disks nowadays take an aggressive, optimistic attitude to head-settling before a read operation. This means that they will attempt a read as soon as the head is "near" the right track—after all, if the data is unreadable because the settle has not quite completed, nothing has been lost. (There is enough error correction and identification data in a mis-read sector to be sure that the data is not wrongly interpreted.) On the other hand, if the data is available, it might just save an entire revolution's delay. For obvious reasons this approach is not taken for a settle that immediately precedes a write. The difference in the settle times for reads and writes can be as much as 0.75ms.

On some recent drives designed for use in portable devices [HPKittyhawk92], the embedded servo information is augmented with an accelerometer, which can determine that a jolt has exceeded the track-following capabilities of the arm and servo system. If this happens, the controller immediately stops any active write to prevent the head from accidentally overwriting an adjacent track.

2.2.3 Data layout

A SCSI disk appears to its client computer as a linear vector of addressable *blocks*, each typically 256–1024 bytes in size. (Although sizes that are a power of two are frequently used, modern SCSI disks can often be re-formatted with other sized blocks. One use for this is to store metadata to be associated with each logical block [English92].) These blocks have to be mapped to physical *sectors* on the disk, which are the fixed-size data layout units on the medium itself. Separating the logical and physical views of the disk in this way means that the disk can hide bad sectors, and can perform some low-level performance optimizations, although it does complicate the task of higher-level software that is trying to second-guess the controller (e.g., the 4.2BSD fast file system).

Each disk sector has a header, followed by a short gap, followed by the user data. Both the header and the user data are protected by extensive error correction codes (ECCs, typically some form of Reed-Solomon code), which allow one or more bit errors to be hidden. Almost all disks map blocks one to one onto sectors.

Zoned bit recording. Track length at the outside of a disk is typically twice what it is on the inside edge of the usable portion of the platter surface. Maximizing storage capacity requires that the linear density remain near the maximum that the drive can support—i.e., the amount of data stored on each track should scale with its length. This is accomplished on many disks by a technique called Zoned Bit Recording (ZBR): adjacent disk cylinders are grouped into *zones*. Zones nearer the outer edge have more sectors per track than zones on the inside. Since keeping layout information about each zone costs some controller resources, the number of zones is typically limited to less than a dozen, although up to twice this many have been used in some disks. ZBR

results in track sizes of 25–60KB today—these numbers will probably double by the end of the decade. Since the data transfer rate is proportional to the rate at which the media passes under the head, the outer zones have higher data transfer rates: for example, on an HP C2240 3.5" disk, the burst transfer rate varies from 3.1MB/s at the inner zone to 5.3MB/s at the outermost one [HPC2240].

Track skewing. Better sequential access times across track and cylinder boundaries are obtained on SCSI devices by skewing logical sector zero on each track by just the amount of time required to cope with the most likely worst-case head- or track-switch times. This means that reading or writing data can proceed at nearly full media speed. Each zone has its own track and cylinder skew factors.

Sparing. The disk surfaces will invariably have some flawed sectors that cannot be used: it is prohibitively expensive to manufacture "perfect" surfaces. Such flaws are found by extensive testing in the manufacturing process, and a list built and recorded on the disk for the controller's use.

In order to not use the flawed sectors, references to them are remapped to other portions of the disk—a process known as *sparing*. Sparing can occur at the granularity of single sectors or whole tracks. The simplest technique is to remap a bad sector or track to an alternate location (e.g., to a spare sector at the end of the track, or to one of a group reserved in each cylinder). Alternatively, *slip sector sparing* can be used, in which the logical block that would map to the bad sector and the ones after it are "slipped" by one sector. Many combinations of techniques are possible: disk designers have to make a complex trade-off between performance, expected bad-sector rate, and space utilization. As one concrete data point, the HP C2240 disk uses both forms of track-level sparing: slip-track sparing at disk-format time, and single-track remapping for defects discovered during operation. Of its total of 2051 cylinders, 69 are reserved for spares, grouped into eight clumps (one for each zone). This prevents a slippage or remapping from propagating too far. An additional cylinder is reserved for logs and other maintenance information.

Error correction. During normal operation, latent surface defects, power fluctuations, vibration, shock, and a number of other causes may render previously-good sectors of the disk permanently or temporarily unusable. (For example, a temporary low-voltage on the power supply at write time may result in a sector that is hard to read later.) Such errors typically manifest themselves only on reads. If the controller detects a problem reading such a sector, it will retry the read a number of times in an attempt to recover the data. The precise number and techniques used for these retries vary, but 8 retries is not uncommon. The first attempt will simply retry the read. More aggressive attempts will skew the head slightly off track in order to approximate a possible similar skew at write time, or run the full error-correcting algorithm over the ECC code embedded in the recovered data. (Although the ECC code is capable of correcting several multi-bit errors, doing so can take several milliseconds. The code—and the hardware that supports it—are tuned to recover from single errors essentially instantaneously, possibly at the cost of longer recovery periods for more errors.)

In practice, almost all of these retries succeed, and although soft error rates of 1 in ~10^{13} occur, after the retries this drops to only 1 in ~10^{15} bits transferred. For those few failures that do eventually become hard (i.e., unrecoverable), the bad sector can be added to the disk's list of spared areas by a "reassign block" or "format disk" operation.

In many disks, the soft error rate increases markedly near the end of a disk's useful life, and this can be used to predict likely coming hard failures, and migrate data off the disk before it fails [Wasmuth86, Siewiorek92]. Such catastrophic failures are becoming increasingly rare: current state of the art disks specify failure *rates* as low as 300000 hours mean time between failures (MTBF)—note that this does not mean that the expected lifetime of any individual disk as expected to be 30+ years! However, it appears that disks from the same batch sometimes exhibit similar failure modes, so the usual failure-modelling assumption of independent failures should be treated with some caution [Jim Gray, personal communication].

2.3 The disk controller

The function of the disk controller is to mediate access to the mechanism, run the positioning servo, send and receive data from the client of the disk across the SCSI interface, and in many cases, to manage an embedded cache for recently read or written data.

Controllers are built around specially designed microprocessors, which often have digital signal processing capability and special interfaces that allow them to control hardware directly. The trend is towards ever more powerful controllers, in order to handle increasingly sophisticated interfaces (such as the command queueing of SCSI-2), and to reduce costs by replacing previously-dedicated electronic components with firmware.

Interpreting the SCSI requests and performing the appropriate computations takes time. The increase in speed of the controller microprocessor is just about keeping ahead of the additional functions it is being asked to perform, so controller overheads are slowly declining. Current values are typically in the range 0.3–0.6ms, reaching up to 1.5ms for older disks.

2.3.1 Bus interface

There are a multitude of different interfaces to disks, of which the most widespread are probably SCSI, IDE, IPI, the PC-AT interface, and HP-IB (IEEE-488). New standards such as FiberChannel and serial SCSI are being developed to alleviate the connectivity problems associated with SCSI's multi-wire cables. Each of these interfaces conforms to an electrical specification for the cabling, and, increasingly, ever more tightly specified standards for the logical interchanges that occur between disk and host computer.

In some interface standards, such as SCSI, the controller function is delegated to the disk drive. In others, such as IDE and IPI, the disk provides minimal functions, and additional intelligence is required in the host end of the interface. For example, multiple requests have to be sequenced in the host or the host interface unit with IPI, but this sequencing can be done in the disk drive with SCSI-2. We concentrate here on disks with SCSI bus interfaces, both because SCSI is currently popular, and because SCSI disks are good examples of the trend towards more delegation of responsibility to the disk drives themselves.

The most important aspects of any interface are its *topology* (bus, like SCSI; point-to-point, like HP-FL; or switched, like FiberChannel), its *transfer rate* and its *overheads*. SCSI is currently defined as a bus—although alternative versions are being discussed, as are encapsulations of the higher-levels of the SCSI protocol across other transmission media, such as FiberChannel and HIPPI. Most disks use the synchronous mode of SCSI bus operation, which can run at the full (electrical) rated bus speed. This was 5MB/s with early SCSI buses; differential drivers and the "fast SCSI" specification

increased this to 10MB/s a couple of years ago; disks are now appearing that are capable of 20MB/s ("fast, wide"); and the standard is defined up to 40MB/s.

The process of negotiating for the bus needs to be efficient, or the transfer times will be dominated by setup overheads. On SCSI, this overhead is quite small (approximately 50μs). Because it is a bus, more than one device can be attached to it. SCSI initially supported up to eight addresses, and this has recently been doubled with the use of wide SCSI. As the number of devices on the bus is increased, contention for the bus can occur, leading to delays. This matters more if the disks are doing large data transfers, or if they have higher controller overheads.

2.3.2 Bus and disk interaction

In older channel architectures, such as the IBM channel protocol and IPI, there was no buffering in the disk itself. As a result, if the disk was ready to transfer data to the host, but the host interface was not, then the disk had to wait an entire revolution time for the same data to come under the head again before it could retry the transfer. In SCSI, the disk is expected to have a speed-matching buffer to avoid this, masking the asynchrony between the bus and the mechanism. The precise form of this buffering, and the optimizations that are undertaken to make more effective use of the mechanism and the bus, vary widely.

Read requests require a seek and rotation to the start address and then a transfer of data into the disk's buffer memory. Most drives can get data off the media much slower than they can send it over the bus, so the drive partially fills its buffer before attempting to commence the bus data transfer. The amount of data read into the buffer before the transfer is initiated is called the *fence*; its size is a property of the disk controller, although it can be (partially) specified on modern SCSI disks by a control command. Write requests can overlap the data transfer to the disk's buffer with

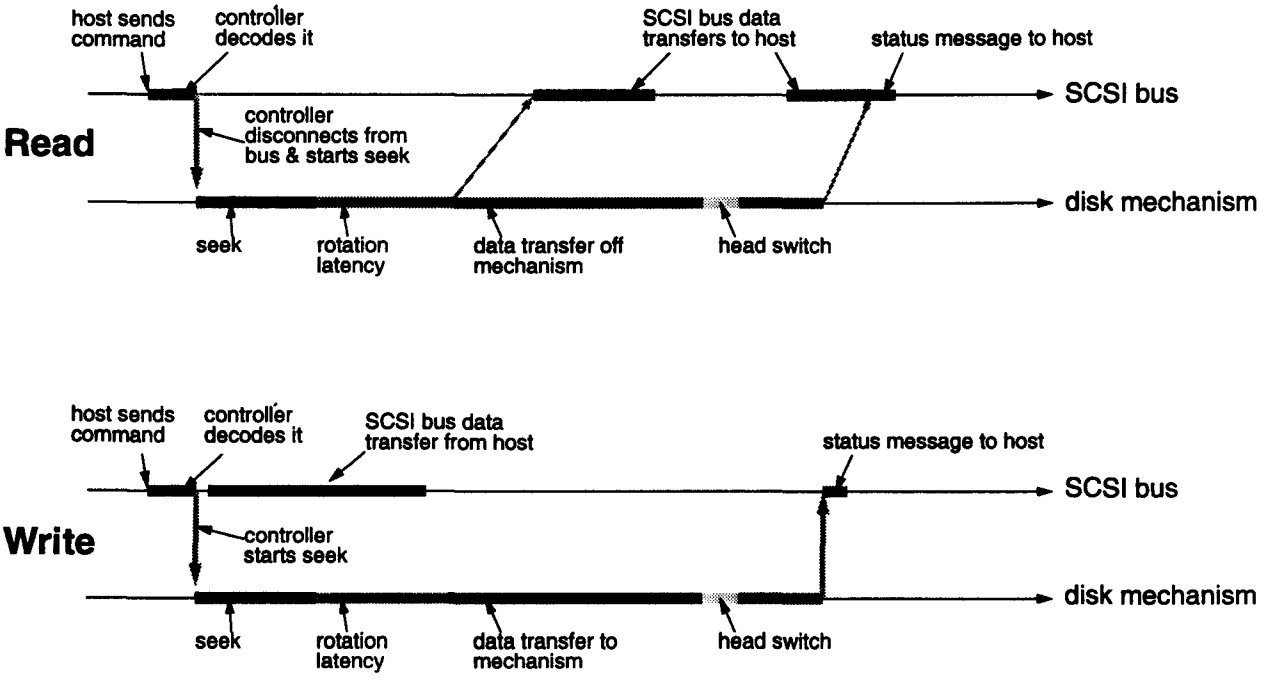


Figure 2: overlap of bus phases and mechanism activity. The low-level details of bus arbitration and selection have been elided.

the head repositioning, up to the limit permitted by the buffer's size. These interactions are illustrated in Figure 2.

To minimize bus contention, the SCSI bus protocol allows a device to disconnect from the bus when it is temporarily unable to perform a data transfer, and reconnect later. This allows a disk to release the bus between receiving a read command and the head being positioned over the data on disk, thereby enabling other disks to use the bus for transferring data they have already read.

2.3.3 Caching of requests

The functions of the speed-matching buffer in the disk can readily be extended to include some form of caching, for both reads and writes. Note that cache sizes in disks tend to be small (currently 64KB to 1MB, with most less 256KB or less) thanks to space limitations and the relatively high cost of the dual-ported static RAM needed to keep up with both the disk mechanism and the bus interface.

Readahead. A read that hits in the cache can be satisfied "immediately"—i.e., just the time it takes for the controller to detect the hit and send the data back across the bus. This is usually much less than a read request that has to go to disk, and so most modern SCSI disks provide some form of data caching. The most common form is *readahead*: caching data that the disk expects the host is just about to request.

As we will see, read caching turns out to be very important from the point of view of modelling a disk, but it is one of the least well-specified areas of disk behavior. For example, a read that partially hits in the cache may be partially serviced by the cache (only the non-cached portion read from disk), or simply bypass the cache altogether. Very large read requests may always bypass the cache. Once a block has been read from the cache, some controllers discard it; others keep it around in case a subsequent read is directed to the same block. The choices here are very much a function of the environment for which the disk is being tuned: what works well for an MS-DOS system (which does no host caching), is frequently a poor choice with a UNIX system (which does a great deal).

Early disks with caches did *on-arrival readahead* to minimize rotation latency for whole-track transfers that followed a seek: as soon as the head arrived at the relevant track, the disk started reading into its cache. At the end of one revolution, the full track's worth of data has been read, and this can then be sent off to the host without waiting for the data after the logical start point to be re-read. (This is sometimes—rather unfortunately—called a "zero-latency read", and is also why disk cache memory is often called a track buffer.) As tracks get longer while request sizes do not, on-arrival caching brings less benefit: for example, with 8KB accesses to 32 KB tracks, the maximum benefit is only 20% of a rotation time.

As a result, on-arrival caching has largely been supplanted by simple readahead: the disk continues to read where the last host request left off. This proves to be optimal for sequential reads, and allows them to proceed at the full disk bandwidth. (Without this, two back-to-back reads would be delayed by almost a full revolution, as the disk and host processing time for initiating the second read request would be larger than the inter-sector gap.) Even here, there is a policy choice: should the readahead be aggressive, and cross track and cylinder boundaries, or stop when the end of the track is reached? Aggressive readahead is optimal for sequential access,

but degrades random accesses, because head- and track-switches typically cannot be aborted once initiated, so can slow down an unrelated request that arrives while the switch is in progress.

A single readahead cache can provide effective support for only a single set of sequential read requests. If two or more sequential requests are interleaved, the result is no benefit at all. This can be remedied by segmenting the cache, so that several unrelated data items can be cached. For example, a 256KB cache might be splittable into eight separate 32KB cache segments by appropriate configuration commands to the disk controller.

Write caching. In most disks, the cache is volatile, losing its contents if power to the disk is lost. To prevent data loss, this kind of cache must be carefully managed if it is to provide write caching. One such technique is *immediate reporting*, which allows only selected writes to the disk to be reported complete as soon as they are written into the disk's cache. The selection is designed to allow file system code to recover from a power failure, so it usually applies only to data writes—and only sequential ones at that—but this allows it to optimize a particularly common case, which is large writes that the file system has split up into consecutive blocks. Individual writes (e.g., to file system metadata) can be flagged "must not be immediate-reported"; otherwise, a write is immediately reported if it is the first write since a read, or a sequential extension of the last write. Combining readahead and immediate reporting means that sequential data can be put into adjacent disk blocks (no interleave factor is necessary), resulting in full disk throughput for data reads and writes.

Although treating the disk's buffer cache as a write-though has been used in some IBM disk controllers [Menon88], this is of little benefit with SCSI disks, since it only benefits those operating systems that immediately reread data they have just written, and these are rapidly becoming uncommon. It is also economically impractical, since host RAM is typically much cheaper than disk buffer cache memory.

The problems of volatile write caches go away if the disk's cache memory can be made non-volatile. One technique is battery-backed RAM: a lithium cell can provide 10 year retention times. With this, the disk is free to accept as many write requests as will fit in its buffer, and acknowledge them all immediately. By delaying writing these requests to disk, it can reduce total mechanism traffic (by absorbing overwrites of the same block, which are surprisingly common), and optimize the write-back sequence (although preliminary studies show this to have less effect than was anticipated). These issues are discussed in more detail in [Ruemmler93].

As with read caching, there are many policies possible for handling write requests that hit in previously-buffered data. Without non-volatile memory, the safest solution is to delay such writes until the first copy has been written to disk. Again, very large writes usually bypass the cache altogether. Data in the write cache must also be scanned for read hits—in this case, the buffered copy has to be treated as primary, since the disk may not yet have been written to.

Command queueing. With SCSI, support for multiple outstanding requests at a time is provided through a mechanism called *command queueing*. This allows the host to give the disk controller several requests, and let the controller determine the best order in which to execute them—subject to additional constraints provided by the host, such as "do this one before any of the others you already have." Letting the disk do the sequencing potentially allows it to perform a better job by using its detailed knowledge of the disk's rotation position [Seltzer90b, Jacobson91]. The

downside is that the disk scheduling algorithm cannot as easily be tuned for different workloads, and it is executed on a slower processor than that available in most hosts.

2.4 Optical disks

Although magnetic disks is the focus of this paper, optical disks are becoming sufficiently common that we provide a short introduction to their characteristics here.

On purely-optical disks such as CD-ROM, there is usually only a single surface, and the data is written as a single long spiral rather than as independent tracks, as a result of their genesis in the audio and video industry. Track densities are generally much higher than with magnetic disks, and seek times are generally longer—partly because settling times are longer, partly because the head assemblies are heavier, and partly because these mechanisms are often adapted from those used in consumer electronics, where seek time is not so important. The last reason also sets the transfer rate for CD-ROM drives to 1.4Mbits/s. (Dual-standard CD-ROM readers are now available that run at twice this speed, but it is still very slow by comparison with a magnetic disk. .)

Magneto-optical (MO) disks combine multiple-overwrite capability (like magnetic disks) with the storage density of optical ones, by the use of material that has different optical properties when magnetized in different directions. Again, track densities are much higher than with magnetic (~20000tpi, or 800mm^{-1}), although linear densities are only half those of magnetic disks (30000bpi, or 1200mm^{-1}). First-generation 5.25" MO disks provided 325MB/surface, and read/write times of around 0.5MB/s. (Capacity has recently doubled: expect a similar storage density increase rate as for magnetic disks.) Writing on these disks is particularly expensive: the area to be written must first be erased, which takes an extra pass. Newer drives have increased read/write rates, and the separate write pass may be eliminated soon through the use of multiple lasers. The real advantage of these disks is their better storage density, coupled with the ease with which they can be removed from the disk mechanism, so their main use has been in environments were performance is a secondary consideration, such as tertiary storage systems. Advances in MO disk technology are roughly tracking those for magnetic disks, so it seems unlikely that they will oust magnetic disks from their preeminent position in regular applications.

3 Modelling disks

The remainder of the paper describes the way we model the behavior of the disks we have just described. Our intent is to describe our models in sufficient detail to allow others to reconstruct them, and to quantify the relative importance of the different components of our model, so that a conscious choice can be made as to how much—and which—detail is required in a model built using our framework.

In addition to poring over the manufacturers' data, we ran several sets of experiments to determine the performance of the disks we wished to model. Even as simple a problem as determining the seek profile for a disk is complicated by the need to second-guess the disk caching algorithm (e.g., the seek to be measured must be preceded by one more seek/read pair than there are independent readahead cache segments), and the effects of rotation latencies (for which a very large number of samples is required, using randomization of the start time for requests to prevent getting into lock-step with the disk mechanism). The descriptions offered above of how disk

controllers are likely to behave should provide the necessary data for others who wish to perform similar experiment on their own disks.

3.1 The simulator

We embodied the background information about disk drives presented above, together with a great deal of exploratory measurements of real disk drives, in a sequence of event-based simulation models. These were built in C++ using the AT&T tasking library [ATT89], although the basic ideas are readily applicable to other simulation environments. (We modified the tasking library to support time as a double rather than a long. This let us extend our simulation periods out to nearly arbitrary durations with very fine time granularity.)

The simulation framework is very simple: *tasks* represent independent units of activity that can *delay* to advance simulated time, or execute code. They can also wait for certain low-level events; on top of these primitives it is easy to construct a variety of synchronization primitives.

We model a disk as two tasks and some additional control structures (see Figure 3):

- One task models the *disk mechanism*, including the head and platter (rotation) positions. This task accepts requests of the form "read this much from here", "seek to there", and executes them to completion, one at a time. It also handles the data-layout mapping between logical blocks and physical sectors.

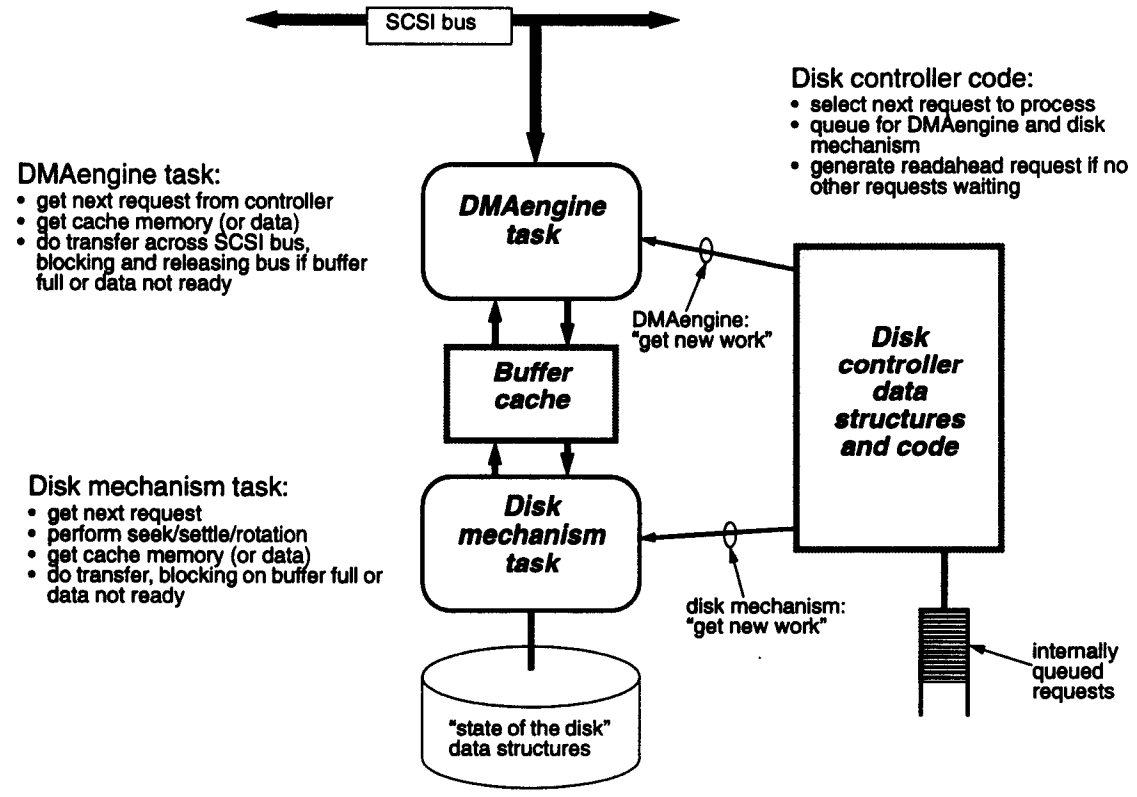


Figure 3: simulation model structure for a single disk.

- The second (*DMAengine*) task models the SCSI bus interface and its transfer (DMA) engine. This task accepts requests of the form "move this host request to (or from) the disk's memory from (or to) the host's memory". Again, it processes one request at a time.[3]
- A *buffer cache object*, shared by the DMAengine and the disk mechanism tasks, represents the disk buffer cache. This is used, in classic producer-consumer style, to manage the asynchronous interactions between the bus interface and the disk mechanism.

  Consider a read: if the disk mechanism gets ahead of the DMAengine, the cache fills up, and the mechanism task blocks, waiting for space to write into. (When it restarts, it does all the correct accounting for missed disk revolutions.) If the DMAengine gets ahead of the mechanism, it first releases its hold on the SCSI bus, and then blocks waiting for data to be available for it to transfer. Once data is available, it reclaims the bus and starts transferring. A parameter of the model determines the granularity with which this handshaking is done, in units of physical sectors. The smaller the value, the more accurate the results, but the longer the running time. Values in the range 0.5–4KB give reasonable results.
- A request scheduler determines the order in which to give work to the mechanism and DMA tasks. In a single-request disk, this is straightforward. In a disk that supports multiple outstanding requests, this scheduling is considerably more complicated—and outside the scope of this paper. If there are no outstanding requests, the scheduler generates single-sector readaheads until the readahead buffer is filled.

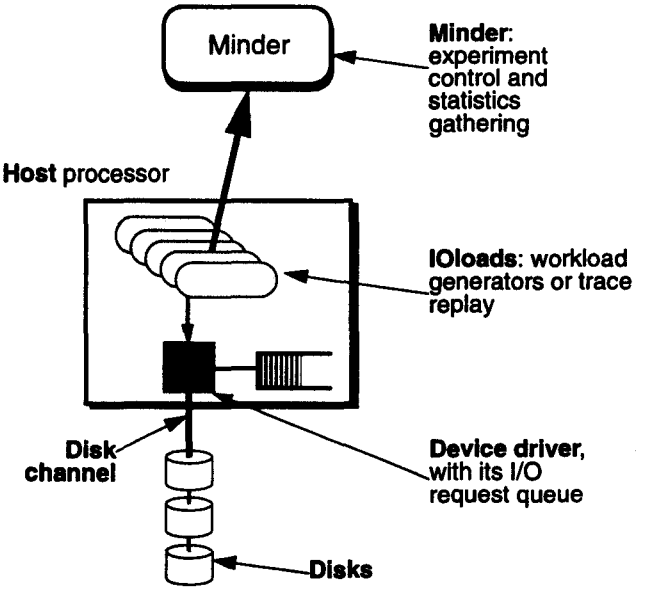


Figure 4: complete simulation-model structure.

The disk model fits into a larger system that has items for representing the SCSI bus itself (a mutex, so that only one access can be using the bus at a time), the host interface (another DMAengine), and synthetic and trace-driven workload generator tasks, as well as a range of statistics-gathering and reporting tools. Figure 4 puts all this together.

3. Actually, there is one of these tasks for each cache memory segment; they compete for access to a passive resource representing the DMA channel itself. Space prevents us from describing the multi-segment case in detail in this paper.

The disk-related portions of our simulation system consist of about 5800 lines of commented C++ code (1500 semicolons), divided roughly as follows:

function	*lines*
disk mechanism	600
DMAengine	400
disk controller	1800
request scheduling[a]	950
disk infrastructure	2050

[a] *six different scheduling policies.*

There are also around 7000 lines of other infrastructure (workload generators, synchronization primitives, statistics gathering, and the like). With minimal optimization, the simulator is able to process about 2000 I/Os per second on an HP9000 Series 800 model H50 system, which has a 96MHz (78 SPECint92) PA-RISC 7100 processor. We are working to improve the simulator's performance, although it is fast enough that we can run a simulation against the 1.2 million I/Os in the one-week trace from our file-server system in about 10 minutes.

3.2 Traces

For this study, we selected representative week-long traces from a longer trace series of HP-UX (UNIX) computer systems. The systems and the traces are described in much greater detail in [Ruemmler93].

For each request, the traces included data such as start and finish times to 1µs granularity, disk address and transfer length, and flags such as read/write, and whether the request was marked synchronous or not by the file system. The start time corresponds to the moment when the disk driver gives the request to the disk; the finish time when the "request completed" interrupt fires; we do not include time spent queued in the disk driver in the results we present here. We developed tools to compare the real trace with the output from our simulators fed with the same sequence of traced requests.

The disks we singled out for analysis in this paper are described in the table below. The I/O execution times from the traces are plotted as distribution curves in Figure 5.

	Formatted	*Track*	*Cyl-*		*Rotational*	*Average*	*Host interconnect*	
Disk type	*capacity*	*buffer*	*inders*	*Size*	*speed*	*8KB access*	*type*	*max speed*
HP C2200A	335 MB	none	1449	5.25"	4002 rpm	33.6 ms	HP-IB	1.2MB/s[a]
HP 97560	1.3 GB	128 KB	1935	5.25"	4002 rpm	22.8 ms	SCSI-2	10MB/s

[a] *See section 3.3.4.*

Modern disks with SCSI interfaces tend to have disk caches, and the presence of this cache complicates the analysis of the performance effects that are related to the underlying disk mechanism. To avoid this problem, we use a non-caching disk (the HP C2200A) for the first set of models that we present here. The HP 97560 disk, which has a buffer cache in the disk, is used later to exemplify the effects of adding caching, once the underlying disk model has been shown to be correct [HPdisks91a]. The HP C2200A disk has an HP-IB (IEEE-488) bus instead of a SCSI interface [HPdisks90]. From the point of view of the model, the only major difference is that the HP-IB bus

is slower than the disk mechanism, whereas the reverse is usually true of SCSI disks. This tends to emphasize the importance of bus-related effects, as we shall see.

3.3 Evaluation

This section presents a sequence of increasingly sophisticated disk models, and compares their performance with that of the real disk they were simulating.

3.3.1 Metrics

For comparison purposes, we need a metric to evaluate the models. A simple mean execution time for a request is of some value in calibrating a model to the real world, but provides little differentiation between models, precisely because the tuning is so easy. (Consider a simulation that simply modeled all requests as taking the mean completion time!) So, we plot the time distribution curves for the real and the model outputs (the cumulative fraction of requests that complete in less than a particular time), and use the mean distance between these two curves as our metric. We call this the *demerit figure* of the model, and present it in both absolute terms (as a difference in milliseconds), and relative ones (as a percentage of the mean I/O time). The real trace data has a demerit figure of zero—that is, it is an exact match to itself.

One might think that a better demerit figure would be to use the mean difference on a request by request basis. However, there is a problem with the rotation positional calculation that makes this approach undesirable. The 0.5–1% variance in the disk rotation speeds means that the simulator (which assumes a perfect disk, running constantly at exactly the nominal speed) can be up to a whole revolution out of sync with the real one. In practice, this means that a request that starts a burst of I/Os after a long delay (1 second or more) experiences a random rotation delay, but subsequent ones in the burst are properly modelled because the rotation position gets recalibrated at the end of this first request. This issue could be addressed in one of two ways:

1. Each simulator could be run multiple times, with each run using a different rotational offset for its disks at the start of the simulation, and the results averaged, or the best fit chosen. Besides being computationally intensive, this would only improve the match if each disk ran at exactly the nominal speed, but with a constant offset from the simulated one—but in practice, the disk rotation speed meanders gently around the nominal value.

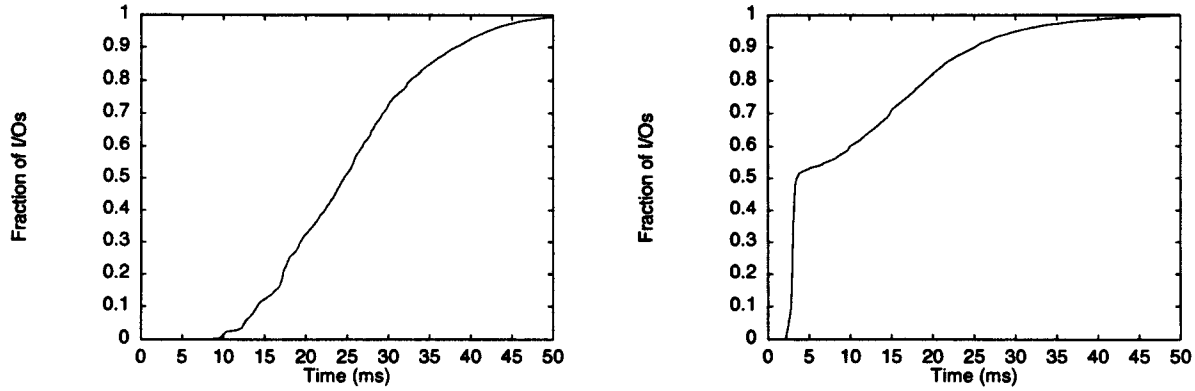


Figure 5: distributions of actual traced I/O times for the disks discussed in this paper. The left-hand graph is of the HP C2200A, the right hand one the HP 97560.

2. The simulated disks could be resynchronized to the real rotational position by running two simulations in parallel: one of the new model, one of the real system that was traced. By allocating the discrepancies between the real trace and its simulation to rotation-position mismatches, the actual rotation position could be recalibrated as often as was deemed necessary. This also has problems: it is technically hard to partially undo the effects of a disk I/O in the simulator to account for the rotational position; it is not obvious that the rotation position is always the primary cause of a mismatch; and it is of no value in cases where the cross-calibration cannot be done—e.g., when the original trace is replayed at a different rate than actually occurred, or if a synthetic workload is used.

Since we are more interested here in demonstrating that the models show a good match with the overall behavior of a particular kind of disk than we are with demonstrating that our model exactly captured the behavior of a particular disk on a particular day, the distribution graphs serve our purpose better than a request-by-request comparison. (An individual disk at a particular time can be influenced by environmental variations in temperature, humidity, power supply voltage, etcetera, that would be hard to model.)

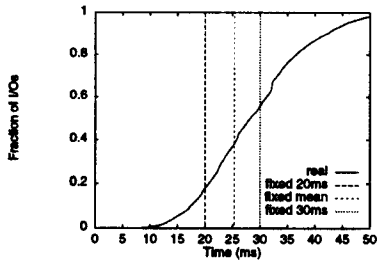
3.3.2 Trivial models

The simplest possible model is a constant, fixed time for each I/O. Figure 6 plots two "typical" values from the literature (20ms and 30ms), together with the actual mean I/O time for the week's traced data. (95% confidence intervals follow the mean values, preceded by a ± symbol.) This model is not good: even using the mean I/O time rather than a fixed estimate results in a demerit factor that is 28% of the average I/O time.

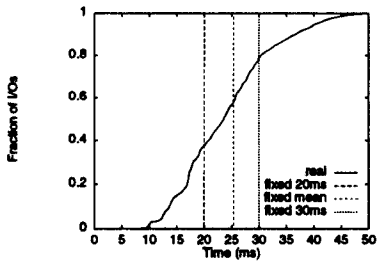
The next easy thing to model is the size of the I/O requests. We did this with a linear regression that matched the observed data to a constant overhead time plus a constant cost per byte transferred. For this disk, the best fit comes from the equation:

$$T = 20.8\times10^{-3} + 0.82\times10^{-6} \times l$$

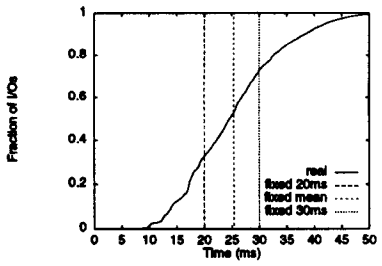
where l is the length of data transferred. The results from this model are shown in Figure 7: it is somewhat better than the fixed I/O size, with the demerit figure reduced to only 22% of the mean I/O time. However, it is possible to do *considerably* better.



Reads	mean	demerit
real disk	28.93±0.17	
fixed 20ms	20.00±0.00	10.17±0.97
fixed 30ms	30.00±0.00	7.58±0.66
mean	25.35±0.00	7.79±0.75

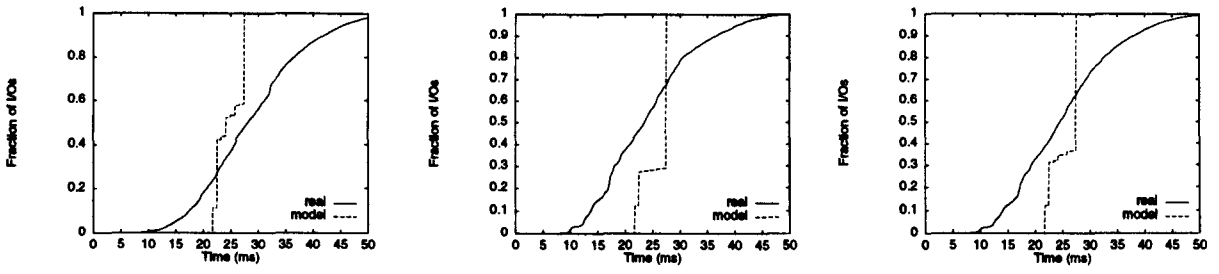


Writes	mean	demerit
real disk	24.12±0.10	
fixed 20ms	20.00±0.00	7.31±0.73
fixed 30ms	30.00±0.00	8.75±0.68
mean	25.35±0.00	7.01±0.61



Both	mean	demerit
real disk	25.36±0.09	
fixed 20ms	20.00±0.00	8.05±0.81
fixed 30ms	30.00±0.00	8.45±0.68
mean	25.35±0.00	7.21±0.65

Figure 6: I/O time distributions for the trivial models for the C2200A. The row labelled "mean" in the tables refers to the fixed value that is the mean I/O time for both reads and writes.



Reads	mean	demerit
real disk	28.93±0.17	
simulation	24.85±0.09	6.27±0.67

Writes	mean	demerit
real disk	24.12±0.10	
simulation	25.97±0.07	6.01±0.48

Both	mean	demerit
real disk	25.36±0.09	
simulation	25.70±0.06	5.69±0.53

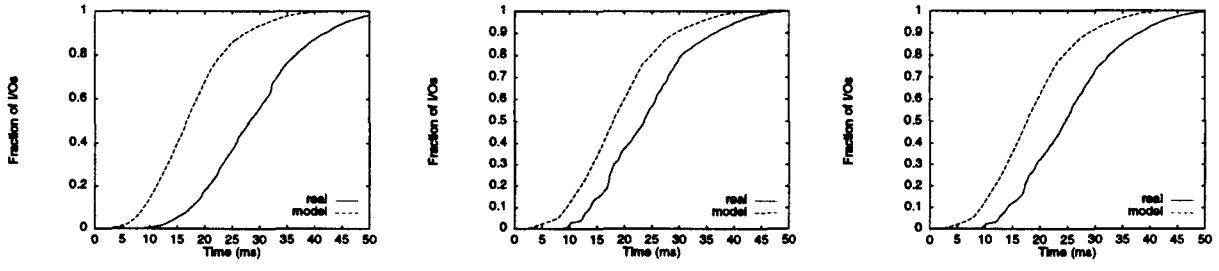
Figure 7: the linear-regression model for the C2200A.

3.3.3 Adding seek distance and random rotation delay

To improve on the simplistic models' lackluster performance, it is necessary to remember state information between requests. One of the simplest models that does this has the following combination of features:

- a linear seek time-versus-distance curve, derived from the full-stroke seek time published in the disk specification,
- no settle effects,
- a uniform random rotational latency over the interval [0, rotation-time),
- a fixed, constant transfer time, encompassing the media access time, the HP-IB bus transmission time, and the controller overhead.

Figure 8 shows how this fares when compared against the real traces for the C2200A disk. Using the published performance data for the disk drive, this first simulation model does worse than the linear regression, being consistently faster than the real thing, most noticeably on reads. The mean is off by 27%, and the demerit figure represents 27% of the mean I/O time. The linear regression has the advantage of being fitted to observed rather than published data (which is why its mean is a much better match), but there are some other problems that can also be rectified quite easily.



Reads	mean	demerit
real disk	28.93±0.17	
simulation	17.55±0.14	11.37±0.13

Writes	mean	demerit
real disk	24.12±0.10	
simulation	18.73±0.09	5.4±0.06

Both	mean	demerit
real disk	25.36±0.09	
simulation	18.43±0.08	6.92±0.09

Figure 8: a simple model with a linear seek-time and random rotation delays for the C2200A.

3.3.4 Improving the data-transfer model

There are three immediate problems with the simple C2200A model presented above:

- there is an asymmetry in transfer rates across the HP-IB bus: on real disks, data is read from the disk faster than it is sent to it. Figure 9 shows this by plotting transfer time as a function of request size for reads and writes separately: reads run at 1MB/s, writes at 1.2MB/s. (On this disk, the media transfer rate of 1.9MB/s is faster than the HP-IB bus, so bus speed dominates. This is not usually the case with SCSI disks.)
- The apparent fixed controller overhead is much greater than the published figure (1.5ms).

  For reads, this is caused by a "feature" of the disk controller firmware: although this is no longer present in more recent drives, it is instructive nonetheless. This particular disk behaves as if it *always* reads 8KB off the media, whatever the size of the original request. Thus if 1KB is requested, the drive will read 8KB off the media before starting the bus transfer of the requested 1KB. Subtracting the time to do this extra data transfer from the medium results in a more reasonable read overhead of 1.1ms.

  We have no explanation for the write overhead figure; fortunately, all we need to do is to model it accurately.
- Both transfer curves experience a jump at around the 28 KB transfer size which corresponds directly to where the transfer crosses a track boundary. The simple model does not take this into account.

We improved this simple model by: emulating the read-fence behavior, increasing the write overhead to 4.6ms, and overlapping the (now faster) data writes across the bus in parallel with the seek and rotation latency overheads. The result is shown in Figure 10: the demerit has decreased by a factor of two to 13% of a mean I/O time. This is better, but is still two to three times the size of many of the effects that I/O system designers may wish to investigate.

3.3.5 Modelling head-positioning effects

Until now we have been using a simple linear model of seek time as a function of distance. However, seek times are not linear with distance, as is shown in Figure 11—the mean difference between the linear seek model and the real one is 2.66ms, which is a a 9% error by itself. Given that some calculation has to be done to convert distance into time, we feel that it is not hard to do

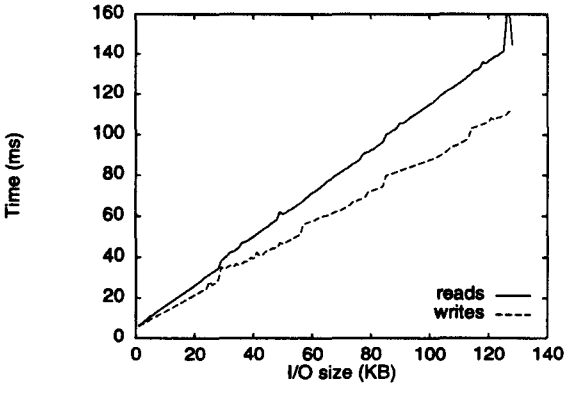
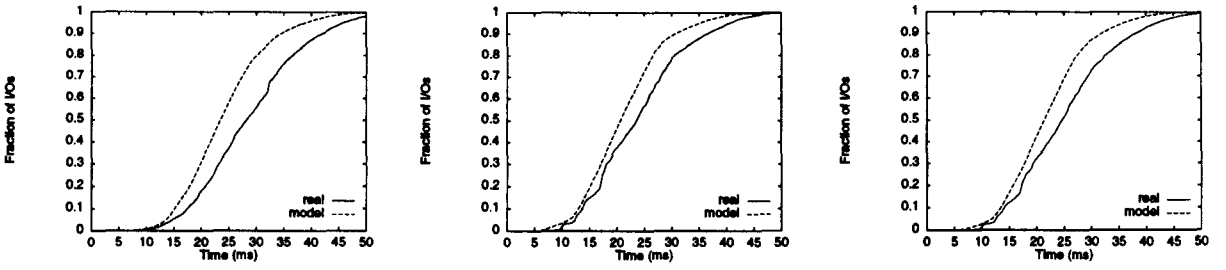


Figure 9: I/O times as a function of request sizes for the C2200A.



Reads	*mean*	*demerit*
real disk	28.93±0.17	
simulation	24.31±0.14	4.62±0.11

Writes	*mean*	*demerit*
real disk	24.12±0.10	
simulation	21.30±0.08	2.82±0.10

Both	*mean*	*demerit*
real disk	25.36±0.09	
simulation	22.08±0.08	3.28±0.11

Figure 10: improved transfer time model for the C2200A.

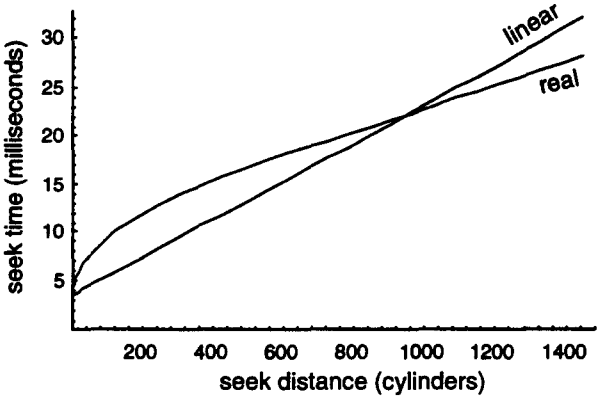
a decent one: all that is lacking is the data to drop in. The table in Figure 11 describes the model we used to approximate the measured seek time profile for this disk. Computing the better model is trivial: the difference is a 6-line calculation rather than a single-line one.

Since we were improving our positioning calculations, we also took the opportunity to model the costs of doing head- and track-switching. This was achieved by determining which track and cylinder the request started on, and where it ended, and then adding in a fixed cost of 2.5ms for each head- and track-switch needed to get from the start of the request to its end. The combined results are shown in Figure 12: the demerit figure has reduced by almost a factor of three to 5% of a mean I/O time.

3.3.6 Modelling rotation position

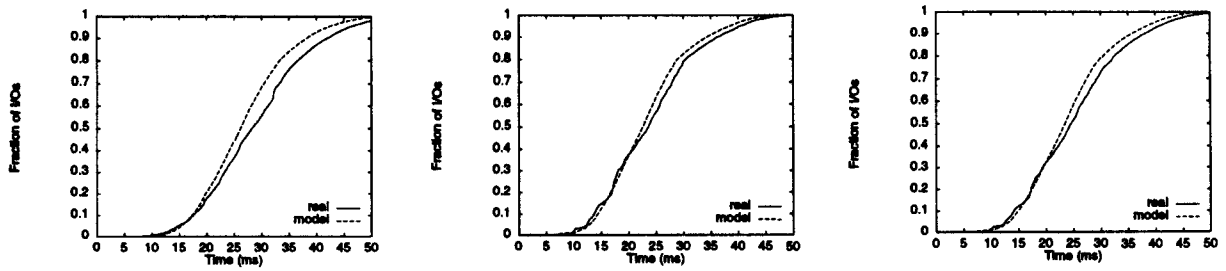
There are only two things left to model on the C2200A: detailed rotational latency and spare sector placement.

By keeping track of the rotational position of the disk the rotational latency can be explicitly calculated rather than just drawn from a uniform distribution. This is done by calculating how many revolutions the disk would have experienced since the start of the simulation, assuming it was spinning exactly at its nominal rated speed. The C2200A uses track and cylinder skewing,



seek distance	*seek time (ms)*
< 616 cylinders	$3.45 + 0.597\sqrt{d}$
≥ 616 cylinders	$10.8 + 0.012\ d$

Figure 11: the graph displays the measured seek time versus distance curve for the C2200A and a linear approximation to it. The table shows the formula we used to model the real curve.



Reads	mean	demerit
real disk	28.93±0.17	
simulation	26.94±0.15	2.08±0.08

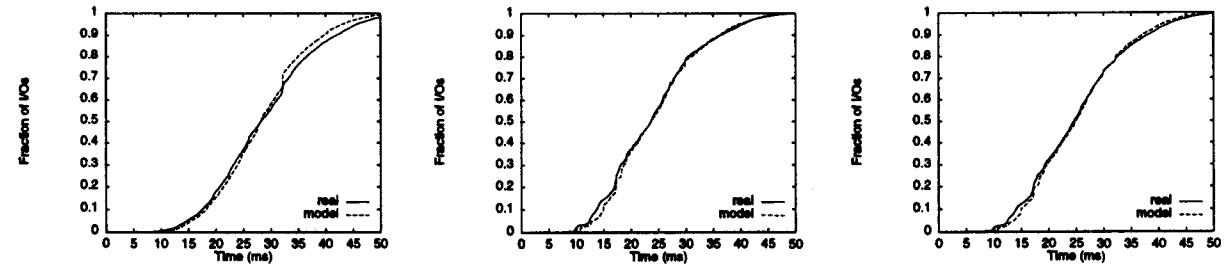
Writes	mean	demerit
real disk	24.12±0.10	
simulation	23.39±0.09	1.06±0.04

Both	mean	demerit
real disk	25.36±0.09	
simulation	24.31±0.08	1.30±0.06

Figure 12: adding a non-linear seek distance model and head-switch times for the C2200A.

and uses sector-based sparing with one spare sector per track, and this need to be taken into account in the mapping of logical blocks to physical sectors.[4]

Adding all these in results in the data shown in Figure 13. We consider this a good match between the real disk and the model: the model gets to within 2% of the measured I/O behavior. For example, the simulation distribution exhibits the same characteristic jump around 30ms for reads as the real trace—this was missing from all the previous models. A complete list of all the parameters used in this final model is shown in Figure 16.



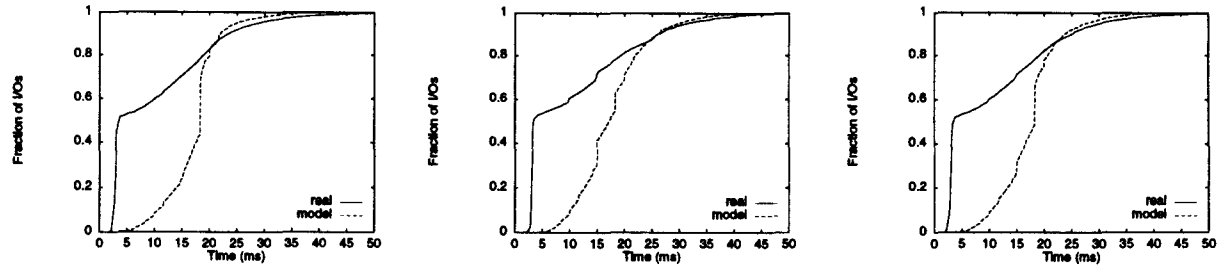
Reads	mean	demerit
real disk	28.93±0.17	
simulation	28.55±0.16	0.99±0.05

Writes	mean	demerit
real disk	24.12±0.10	
simulation	24.42±0.10	0.44±0.02

Both	mean	demerit
real disk	25.36±0.09	
simulation	25.49±0.09	0.52±0.03

Figure 13: final model for the C2200A: includes rotation position and detailed layout models.

4. It also has two reserved spare areas, but these are at the outside edges of the data area, so need not concern us for these calculations.



Reads	mean	demerit
real disk	10.33±0.04	
simulation	17.39±0.02	8.09±0.31

Writes	mean	demerit
real disk	10.66±0.05	
simulation	17.66±0.03	7.48±0.26

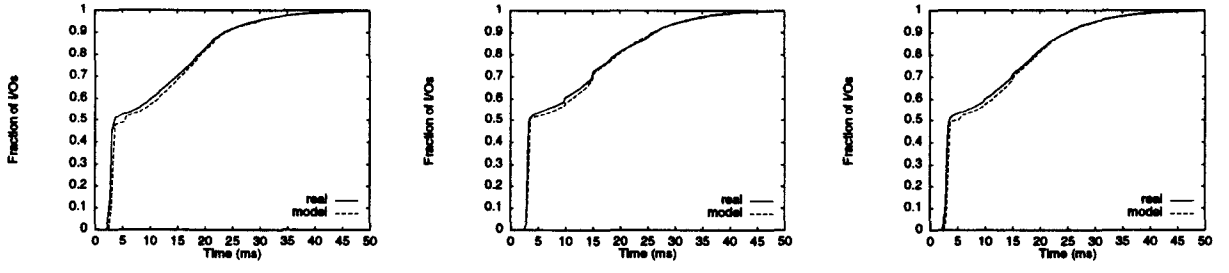
Both	mean	demerit
real disk	10.47±0.03	
simulation	17.51±0.02	7.77±0.29

Figure 14: initial model for the HP 97560.

3.3.7 Modelling data caching

We used the C2200A disk for the previous discussion because it has no on-disk buffer cache. When a disk has a cache, and uses it, the effects can be quite dramatic. Figure 14 shows the effect of using a model incorporating all the features described so far to simulate an HP 97560 SCSI disk that uses both readahead and immediate reporting write-behind.[5] The large disparity at small completion times is due to the caching: around 50% of the requests completed in 3ms or less. Clearly, caching needs to be modelled if the simulation is to get close to the real disk performance.

On this disk, the readahead model is to continue reading after the last read request until a new read request that is not in the buffer arrives, or a write request arrives, or the cache buffer fills up: it does not stop at track boundaries. The write-behind model is immediate-reporting, exactly as described in section 2.3.3. Adding these readahead and immediate reporting features to the disk model gives the results shown in Figure 15. We consider this a very good match: the relative demerit is now only 5% of the mean I/O time—and notice that this mean is only half that of the C2200A's, so the absolute value of the error is comparable.



Reads	mean	demerit
real disk	10.33±0.04	
simulation	10.91±0.06	0.60±0.03

Writes	mean	demerit
real disk	10.66±0.05	
simulation	10.98±0.05	0.43±0.03

Both	mean	demerit
real disk	10.47±0.03	
simulation	10.92±0.03	0.51±0.03

Figure 15: adding data caching to the HP 97560 model.

5. The HP 97560 moniker is the OEM part; variants of the same drive—sometimes with different controller firmware—are also available as the HP C247x and HP C246x series.

3.4 Summary

We selectively enabled a number of features of our disk model to arrive at one that performs extremely well when compared against a real disk. The following table summarizes the models and how well they did.

feature	*absolute demerit*	*relative demerit*	*disk type*
constant mean time	7.21ms	28%	HP C2200A
linear regression	5.69ms	22%	
basic model	6.92ms	27%	
add data transfer	3.28ms	13%	
add head positioning	1.30ms	5%	
add rotation position	0.52ms	2%	
no caching	7.77ms	74%	HP 97560
add caching	0.51ms	5%	

Clearly the full model is necessary if a very good match is required—and, given that it is not particularly onerous to implement, we encourage others to adopt it. Our full model includes the following details (numerical values for the parameters are provided in Figure 16):

- disk controller effects
  - controller overhead (we include a small amount of random noise in this value)
  - SCSI bus contention, and bus disconnects during mechanism delays
  - overlapped bus transfers and mechanism activity
- disk buffer cache, including:
  - readahead
  - write-behind
  - producer–consumer interlocks between the mechanism and bus transfers
- data layout model:
  - reserved sparing areas, including both sector- and track-based models
  - zones
  - track and cylinder skew
- head movement effects
  - non-linear seek time versus seek distance profiles derived from measurements on real disks
  - settle time, with different values for read and write
  - head-switch time
  - rotation latency

As with any model, there are things we have chosen to ignore. For example, we do not believe it is particularly fruitful to attempt to model soft-error retries (with a 1 in 10^{13} soft bit error rate, even a factor of 100 increase in access cost on a retry results in a mean increase in read time of less than one part in 10^5). A similar argument applies to actually simulating the effects of individual spared

sectors or tracks, since the actual rate at which they occur is small—certainly less than a couple of hundred sectors per disk, or 1 in 10^4 sectors.

4 Related work

Disk models have been in use ever since disks became available as storage devices. This survey concentrates on more recent examples to give an indication of the current state of practice, rather than the historical state of the art. In all cases, we have picked illustrative examples from the literature: not to single out any individuals, but because these happen to be examples of the simulation technique we are describing.

The non-linear, state-dependent behavior of disks means that they cannot be modeled analytically with any accuracy, so most work in this area uses simulation. Nonetheless, the simplest models just assume a fixed time for an I/O [Stonebraker89] or select times from a uniform distribution [Brumfield88]. The more elaborate models acknowledge that a disk I/O has separate seek, rotation, and transfer times, but most fail to model these components carefully. For example:

- Seek times modeled as linear functions of seek distance (e.g., [Oney75, Weikum90, Reddy89, Kim91]). This produces poor results at smaller seek distances, which are the most common ones.

Parameter		*HP C2200A*	*HP 97560*
sector size		256 bytes	512 bytes
cylinders		1449	1962
tracks per cylinder		8	19
data sectors per track		113	72
number of zones		1	1
track skew		34 sectors	8 sectors
cylinder skew		43 sectors	18 sectors
revolution speed		4002 RPM	4002 RPM
controller interface		HP-IB	SCSI-II
controller overhead	*reads*	1.1 ms	2.2 ms
	writes	5.1 ms	2.2 ms
seek time	*short (ms)*	$3.45 + 0.597\sqrt{D}$	$3.24 + 0.400\sqrt{D}$
	long (ms)	10.8 + 0.012D	8.00 + 0.008D
	boundary	D = 616	D = 383
track switch time		2.5 ms	1.6 ms
read fence size		8 KB	64 KB
sparing type		sector[a]	track[b]
disk buffer cache size		32 KB	128 KB

[a] The HP C2200A also does track sparing, but the spare regions are at the beginning and end of the data region so they have no effect on simulation performance. The HP C2200A has one spare sector at the end of each track (giving it 114 sectors per track).

[b] The HP 97560 does track sparing, and has dedicated sparing regions embedded in the data area. The table below shows where the three data regions are located physically on the HP 97560 disk, using the format "cylinder/track" to indicate boundaries in the physical sector space of the disk. This disk has 1962 physical cylinders, but only 1936 of these are used to store data: the rest are spares.

Region	*0*	*1*	*2*
Start	1/4	654/0	1308/0
End	646/3	1298/18	1952/18

Figure 16: final model parameters for the HP C2200A and the HP 97560.

- Uniform distribution for the rotational latency of an I/O [Oney75, Reddy89, Kim91, Weikum90, Chen91c, Akyurek92, Staelin90]. This is inappropriate when requests are not independent, as is frequently the case.
- Ignoring the disk-media transfer time because the rotation and seek time are believed to be the largest contributors [Kim91] or just using a fixed, constant transfer time per I/O (e.g., [Oney75, Staelin90, Weikum90, Akyurek92]).
- Even though many of the studies simulated disks that presumably shared an I/O bus, the channel contention on the bus was either not modelled, or not discussed (e.g., [Reddy89, Weikum90, Kim91]). This can hide a significant source of contention at higher system loads.

Work described in [Holland92], together with some of our own previous work [Thekkath92a, Ruemmler91, Ruemmler93], used more detailed models that avoided many of the limitations described above. These models simulated axial and rotational head positions, allowing the seek, rotation, and transfer times to be computed instead of drawn from a distribution. This paper is an outgrowth of the simulation work described in [Ruemmler93].

5 Conclusions

An accurate model of a disk drive is essential to obtaining good simulation results from I/O studies. Not modeling disk behavior can result in quantitative—and in extreme cases, qualitative—errors in an analysis. As yet, little has been published in this field, with the result that many I/O performance studies are using inadequate performance models.

We have demonstrated that disks are interestingly complex objects, whose behavioral quirks need to be understood carefully if they are to be accurately modeled for the purposes of making I/O system design choices. We have further demonstrated that such care is not too hard, nor too costly, and have provided data that enables a quantitative determination of benefit to be had from investing effort into a disk model.

By far the most important feature to model is the data caching characteristics of the disk (74% relative demerit if this is ignored). The next most important pair to get right are the data transfer model including overlaps between mechanism activity and the bus transfers (14%), and the seek time and head-switching costs (8%). Although the transfer model had a larger effect in our evaluation of the C2200A than the positioning model, the relative importance will probably be reversed for SCSI disks because there the bus is generally faster than the disk mechanism. Finally, modelling the rotational position and detailed data layout improved the model accuracy by a factor of two—and modelling rotational position accurately will become much more important as file system designers emphasize sequential transfers more.

We caution that even a good model needs careful calibration and tuning. For example, we did not present here the quantitative effects of modelling features such as bus contention and zone bit recording (our model handles these, but space prevented a detailed analysis). These features and others may become important in the face of a particular workload. For example, with many heavily-used disks attached to a single SCSI bus, bus contention may be a significant contributor to the mean request response time.

Finally, we have provided a detailed parameterization of two different disk drives, and described a simulator that uses these parameters to provide a very accurate disk simulation. Our future work includes using this model to explore a variety of different I/O designs and policy choices at host and disk levels. We hope to make the source code of our model available to interested researchers in the latter half of 1993, together with calibrated model parameters for a longer list of disk types than we had space to describe here.

Acknowledgments

Pei Cao contributed greatly to the simulator of which our disk model is a part; Marvin Keshner provided information on several of the underlying storage technology trends. Tim Sullivan provided helpful feedback on a draft of this paper. Needless to say, any errors here are entirely our own responsibility. This work was performed as part of the DataMesh research project at Hewlett-Packard Laboratories.

References

[Akyurek92] Sedat Akyurek and Kenneth Salem. *Adaptive block rearrangement.* Technical report CS–TR–2854. University of Maryland, February 1992.

[ATT89] *Unix System V AT&T C++ language system release 2.0 selected readings,* AT&T select code 307-144, 1989.

[Brumfield88] Jeffrey A. Brumfield, Janet Lynn Miller, and Hong-Tai Chou. Performance modelling of distributed object-oriented database systems. *Proceedings of International Symposium on Databases in Parallel and Distributed Systems* (Austin, Texas, 5–7 December 1988), pages 22–32. IEEE Computer Society Press, 1988.

[Chen91c] Shenze Chen and Don Towsley. *A queueing analysis of RAID architectures.* COINS Technical Report 91–71. Department Computer and Information Science, University Massachusetts, Amherst, September 1991.

[English92] Robert M. English and Alexander A. Stepanov. Loge: a self-organizing storage device. *USENIX Technical Conference* (San Francisco, Winter '92), pages 237–51. Usenix, 20–24 January 1992.

[Holland92] Mark Holland and Garth A. Gibson. Parity declustering for continuous operation in redundant disk arrays. *Proceedings of 5th International Conference on Architectural Support for Programming Languages and Operating Systems* (Boston, MA). Published as *Computer Architecture News* **20** (special issue):23–35, 12–15 October 1992.

[HPC2240] Hewlett-Packard Company, Boise, Idaho. *HP C2240 series 3.5-inch SCSI-2 disk drive: technical reference manual.* Part number 5960–8346, edition 2, April 1992.

[HPdisks90] Hewlett-Packard Company, Boise, Idaho. *HP series 6000 disk storage system's owners manual for models 335H, 670H, and 670XP.* Part number C2200-90901, February 1990.

[HPdisks91a] Hewlett-Packard Company, Boise, Idaho. *HP 97556, 97558, and 97560 5.25-inch SCSI disk drives: technical reference manual.* Part number 5960-0115, June 1991.

[HPKittyhawk92] Hewlett-Packard Company, Boise, Idaho. *HP Kittyhawk Personal Storage Module: product brief.* Part number 5091–4760E, 1992.

[Jacobson91] David M. Jacobson and John Wilkes. *Disk scheduling algorithms based on rotational position.* Technical report HPL–CSP–91–7. Hewlett-Packard Laboratories, Palo Alto, CA. February 1991.

[Kim91] Michelle Y. Kim and Asser N. Tantawi. Asynchronous disk interleaving: approximating access delays. *IEEE Transactions on Computers* **40**(7):801–10, July 1991.

[Menon88] Jai Menon and Mike Hartung. The IBM 3990 disk cache. *Proceedings of COMPCON'88* (San Francisco, CA), pages 146–51, June 1988.

[Muntz90] Richard R. Muntz and John C. S. Lui. Performance analysis of disk arrays under failure. *Proceedings of 16th International Conference on Very Large Data Bases* (Brisbane, Australia, 13–16 August 1990), pages 162–73.

[Oney75] W. Oney. Queueing analysis of the scan policy for moving-head disks. *Journal of the ACM* **22**(3):397–412, July 1975.

[Patterson90] David A. Patterson and John L. Hennessy. *Computer architecture: a quantitative approach*. Morgan Kaufmann Publishers, Incorporated, San Mateo, CA, 1990.

[Reddy89] A. L. Narasimha Reddy and Prithviraj Banerjee. Performance evaluation of multiple-disk I/O systems. *1989 International Conference on Parallel Processing* (University Park, Pennsylvania), pages 315–18. Pennsylvania State University (distributed by IEEE), August 1989.

[Ruemmler91] Chris Ruemmler and John Wilkes. *Disk shuffling*. Technical report HPL–91–156. Hewlett-Packard Laboratories, Palo Alto, CA. October 1991.

[Ruemmler93] Chris Ruemmler and John Wilkes. UNIX disk access patterns. *Proceedings of Winter 1993 USENIX* (San Diego, CA, 25–29 January 1993), pages 405–20.

[Seltzer90b] Margo Seltzer, Peter Chen, and John Ousterhout. Disk scheduling revisited. *Proceedings of Winter 1990 USENIX Conference* (Washington, DC, 22–26 January 1990), pages 313–23.

[Siewiorek92] Daniel P. Siewiorek and Robert S. Swarz. *Reliable computer systems: design and evaluation*. Second edition, Digital Press, 1992.

[Staelin90] Carl Staelin and Hector Garcia-Molina. *File system design using large memories*. Technical report CS–TR–246–90. Department of Computer Science, Princeton University, February 1990, revised June 1990.

[Stonebraker89] Michael Stonebraker. *Distributed RAID – a new multiple copy algorithm*. UCB/ERL report M89/56. Electronics Research Laboratory, University of California at Berkeley, May 1989.

[Thekkath92a] Chandramohan A. Thekkath, John Wilkes, and Edward D. Lazowska. *Techniques for file system simulation*. Published simultaneously as technical reports HPL–92–131 and 92–09–08. Hewlett-Packard Laboratories, Palo Alto, CA and Department of Computer Science and Engineering, University of Washington, Seattle, WA. October 1992.

[Wasmuth86] David B. Wasmuth and Bruce J. Richards. Predictive support: anticipating computer hardware failures. *Hewlett-Packard Journal* **37**(11):30–3, November 1986.

[Weikum90] Gerhard Weikum, Peter Zabback, and Peter Scheuermann. *Dynamic file allocation in disk arrays*. Technical report 147. Department of Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland, December 1990.