Modeling and optimizing I/O throughput of multiple disks on a bus

Rakesh Barve* Elizabeth Shriver† Phillip B. Gibbons†

Bruce K. Hillyer† Yossi Matias‡ Jeffrey Scott Vitter*

Abstract

In modern I/O architectures, multiple disk drives are attached to each I/O controller. A study of the performance of such architectures under I/O-intensive workloads has revealed a performance impairment that results from a previously unknown form of convoy behavior in disk I/O. In this paper, we describe measurements of the read performance of multiple disks that share a SCSI bus under a heavy workload, and develop and validate formulas that accurately characterize the observed performance (to within 12% on several platforms for I/O sizes in the range 16–128 KB). Two terms in the formula clearly characterize the lost performance seen in our experiments. We describe techniques to deal with the performance impairment, via user-level workarounds that achieve greater overlap of bus transfers with disk seeks, and that increase the percentage of transfers that occur at the full bus bandwidth rather than at the lower bandwidth of a disk head. Experiments show bandwidth improvements of 10–20% when using these user-level techniques, but only in the case of large I/Os.

1 Introduction

In the past decade, computer systems have enjoyed a hundred-fold increase in processor speed, while the speed of a disk drive has increased by less than a factor of 10. As a consequence of this disparity, computer systems designed for I/O-intensive processing use many disks in parallel, usually organized as a disk farm or a RAID array. The physical organization generally consists of one or more I/O buses (e.g., SCSI buses) with several disks on each bus.

Previous work related to disk I/O performance has focused on the disk drive, downplaying the importance of bus contention and other bus effects. Much of this work has focused on UNIX workloads with small I/O request sizes, and indeed, bus effects play an insignificant role in I/O performance for such workloads. But many I/O-intensive applications benefit significantly from larger request sizes (16–128 KB). Among these are multimedia servers and certain database and scientific computing applications that use external memory (out-of-core) algorithmic techniques to process massive data sets (e.g., [21, 14, 11]). In such applications, parallel I/O performance is often limited by the bus.

In a parallel I/O system with multiple disks on a bus, many factors influence the I/O throughput. The factors that are usually considered include the number of disks, the size of each I/O transfer, and the positioning time (seek and rotational latency). We also account for the difference between the transfer rate from the disk surface through the disk head to the disk's internal cache, and the burst rate from a disk's cache to the host over the SCSI bus. We model the effect of a disk drive's read lookahead mechanism, which sometimes enables the disk to prefetch data into its internal cache in anticipation of a future sequential read request.

We also consider the effect of a relatively obscure disk control parameter called the *fence*. (The fence is called the *buffer full ratio* on the SCSI-2 disconnect/reconnect control mode page.) When a SCSI disk is instructed to perform a read, and the disk recognizes that there will be a significant delay, such as for a seek, the disk releases control of the SCSI bus (it *disconnects*). When the disk is ready to transfer the data to the host, the disk contends for control of the SCSI bus (*reconnect*) so that the read can be completed.[1] The fence determines the time at which the disk will begin to contend for the SCSI bus. If the fence is set to the minimum value, the disk will contend after the first sector of data has been transferred from the disk surface to the disk's internal cache. In contrast, if the fence is set to the maximum value, the disk will wait until almost all of the requested data has accumulated in the disk cache before contending for the bus. The performance implication is as follows. A low fence setting tends to reduce the response time, because the disk attempts to send data to the host as soon as the first sector is available. But after the cached data has been sent to the host (at the bus bandwidth), the disk continues to hold the bus, and the remainder of the transfer occurs at the disk head bandwidth—the rate at which bits pass under the disk head. The disk head bandwidth is usually less than 25% of the bus bandwidth, and for some disks, far less. A high fence setting causes the disk to delay the start of data transfer to the host, but when the transfer does occur, it proceeds at the bus bandwidth, from semiconductor cache on the disk drive into the host controller and memory system. In systems with multiple disks on a bus, a high fence setting potentially increases overall throughput for I/O-intensive workloads.

Previous work (see Section 2) has presented detailed per-

*Duke University, visiting Bell Labs.

†Information Sciences Research Center at Bell Labs, contact shriver@bell-labs.com.

‡Tel-Aviv University. Work performed at Bell Labs.

[1] As discussed in Section 3 and Section 4.5, disks reaching track or cylinder boundaries on the disk surface may also disconnect, and later reconnect, during data transfer.

formance studies for single disk systems, and approximation techniques for multiple disk systems. For several important workloads, the previous disk models fail to give an accurate prediction of system performance. Accurate predictions are vital for applications such as video servers, which make admission control decisions based on predictions of system performance. Although external-memory algorithms are often designed simply to minimize the number of I/O operations, our model can support a more useful evaluation, by the performance metric of wall clock time.

To our knowledge, a quantitative study of I/O bandwidth involving bus effects and the fence parameter has not appeared in the literature. For a workload generated by one process per disk, each process generating a new read request as soon as its previous read completes, we determine the sustained I/O read bandwidth on the four systems listed in Table 1. Despite the heavy workload, we see some data transfers at the disk head bandwidth, rather than at the higher bus bandwidth, and we observe idle periods. For D disks, once every D requests we see an idle period approximately equal to the time for a seek and rotational latency (see Section 4.5), despite the potential provided by the workload for hiding such overheads.

A closer examination reveals that these performance impairments are due to a particular convoy behavior not reported in the literature: we observe that (for 88–99% of the requests) all disks receive a request, then all disks transmit data back to the host before any disk receives another request. We use the term *rounds* for this periodic convoy behavior. We observe rounds for workloads of random accesses as well as when the accesses on each disk have spatial locality. Rounds occurred regardless of the fence value. As a consequence of rounds, disks are unable to take advantage of a high fence value, so that the largest throughput is observed when the fence value is 0. We also observe that despite the potential unfairness of the SCSI protocol under heavy workloads (i.e., high priority disks potentially may starve low priority disks—see [10], for instance), rounds result in *equal* service to each disk, despite a workload amenable to exhibiting unfairness.

Using a SCSI bus analyzer and kernel instrumentation, we determined that rounds are caused by the host bus adapter, which implements a policy that services in-progress requests with higher priority than sending new requests to idle disks.[2] Thus although the host SCSI controller has a *higher* priority on the bus than any disk SCSI controller, the host's new requests effectively have a *lower* priority than any disk's requests to return data to the host, since the host is prevented from contending for the bus by the host bus adaptor until the disks' requests have been serviced.

We develop and validate formulas that characterize the observed performance to within 12% on the platforms of Table 1, for I/O sizes in the range 16–128 KB. Two terms in the formula clearly characterize the lost performance seen in our experiments. We describe user-level work-arounds that help deal with the performance impairment. Specifically, we employ a prefetching scheme, similar to that in [10], that causes the disks to prefetch data into their internal caches, even for workloads that have random requests. This achieves greater overlap of bus transfers with disk seeks, and increases the percentage of transfers that occur at the bus bandwidth rather than at the lower disk head bandwidth. Whereas the simulations in [10] employ the `SCSI PREFETCH` command, which is not widely supported (e.g., none of the disks in Table 1 support this command), we show how to obtain the effect of this command on any drive, and we describe the measured performance impact of doing so on real systems. Experiments show that bandwidth improvements of 10–20% can be obtained when using our technique to perform large reads from high-performance disk drives. (Our technique is not beneficial for small I/Os or when the SCSI bus is lightly loaded.)

One way to reduce seek times is to use tagged command queuing to send multiple requests to each disk, so that the disks can perform better disk arm scheduling. This approach has the significant limitation that it requires the reservation of buffer space in the host for all the outstanding requests. In addition, tagged commands complete in an arbitrary order, which may increase the application complexity. (In contrast, our pipelining techniques have only one outstanding request per disk.) Moreover, tagged queuing does not eliminate all seeks or increase the percentage of transfers that occur at the bus bandwidth.

Previous work is discussed in Section 2. Section 3 describes the parallel I/O systems that we use for measurements and validation. Section 4 presents and validates our model for the read service time. In Section 5, we propose a method to pipeline random requests to hide the disk latency, and we measure the performance of this technique. Section 6 presents our conclusions and discusses future work.

2 Related work

I/O subsystem modeling. A number of analytic models exist for the I/O subsystem. The analytic disk model of [19] captures bus effects only in the single-disk case and does not directly model the fence. The Pantheon disk simulator [18] incorporates bus contention and other bus effects, but no results have been published that describe the idle periods and head-limited bus transfers that we observe. [12] presents a method for approximating the throughput of multiple disks on a SCSI bus by summing the seek time, rotational latency, and transfer time, and derating the performance by a bus contention factor derived from a general queuing model. The Parallel Disk Model (PDM) [21] is an abstract model for the design and analysis of algorithms on a parallel disk system. [11] presents an application-level study of the accuracy of the PDM. None of this literature describes the rounds phenomenon that we observe. Ignoring rounds when modeling the I/O subsystem can result in throughput prediction errors greater than 100% for the workloads we consider.

[22] proposes many experiments to measure the performance parameters of an individual disk drive. We use similar techniques to determine parameter values for our model.

Readahead. In Section 5, we describe a technique to increase I/O throughput by causing disks to prefetch data from the disk surface to their internal caches (thus overlapping one disk's positioning time with other disks' bus transfer times). This technique differs from previous techniques that prefetch from disk to main memory during sequential access workloads (thus overlapping I/O with application think time), by using historical access patterns (see [17] for references) or by using application- or compiler-provided predictions of future accesses [17, 7, 13, 15]. [17]

[2] An engineer from Sun Microsystems informs us that their SCSI HBA driver processes its internal per-device command queues in round-robin order, not in order of priority implied by SCSI ID.

Table 1: System configurations.

hardware	operating system	I/O card	disks
Sun Ultra-1	Solaris 2.5.1	20 MB/s SCSI	4 Seagate Cheetahs
Sun Sparc-20	Solaris 2.5	20 MB/s SCSI	4 Seagate Cheetahs
DEC AlphaStation 600	Digital Unix 4.0	20 MB/s SCSI	4 Seagate Barracudas
Sun Sparc-20	Solaris 2.5	5 MB/s SCSI-1	7 Imprimis Wren-7s

mentions using application-provided knowledge to schedule future disk I/O to reduce access latency, but this idea is not developed further. [7] presents results on prefetching into main memory by scheduling multiple prefetch commands for two disks on a SCSI bus, but the technique only works when the bus is not the bottleneck. [1] suggests that prefetching from disk into main memory cache by a parallel file system can be beneficial, but we note that the benefit of this approach is less clear when the bus is the bottleneck.

[10] proposes and studies a scheduling algorithm for a video server that has multiple disks on each bus. A `SCSI PREFETCH` command is used to prefetch into each disk's internal cache, prior to a read operation that transfers the data on the bus. A token-based round-robin scheduling of these reads is proposed to overcome the problem of unfair arbitration in the SCSI protocol. [10] reports only simulation results, with no validation of whether or not the simulator adequately captures the disk and bus effects of real systems.

RAIDs. RAID arrays are an established way to increase I/O throughput [16, 9]. They have built-in redundancy, and support accesses to multiple disks by striping. In contrast, our work describes how to model and to improve application-level performance by coordinating concurrent accesses to multiple disks on a SCSI bus. Our technique may be applicable to the internal algorithms used in RAID controllers.

3 Hardware configurations and workload

We conducted measurements on four hardware configurations, as outlined in Table 1. The first consists of four Seagate Cheetah ST-34501W disks connected to a Sun Ultra-1 computer running Solaris 2.5.1. Although this Cheetah has an ultra-wide SCSI interface capable of a 40 MB/s maximum bus rate, the controller card in the computer is a fast-wide card that cannot exceed 20 MB/s. The second configuration is four Seagate Cheetah (ST-34501W) disks connected to a Sun Sparc-20 running Solaris 2.5, again with the 20 MB/s controller. The third configuration is four Seagate Barracuda (ST-32171W) disks connected to a DEC AlphaStation 600 5/266 running Digital UNIX 4.0 with the 20 MB/s controller. The fourth configuration uses seven Imprimis (Seagate) Wren-7 (94601-15) disks (5 MB/s SCSI-1) on a Sun Sparc 20 running Solaris 2.5. Table 2 presents additional parameters of the disks we used, obtained from product specifications [23, 2, 8] or, in the case of the average surface transfer rate, by measurement. In our search for the cause of rounds, we have also performed experiments on a Pentium-based PC with an Adaptec 2944WD controller and the NetBSD operating system with four Seagate Cheetah (ST-34501W) disks.

Each disk has a unique SCSI id which determines the priority of the disk when multiple devices are contending for the bus. The SCSI controller at the host has the highest priority, so it will win any contention in which it participates.

As mentioned earlier, the time at which the disk contends for reconnection to the bus depends on the *fence* parameter. If the fence value is 0, the disk contends for the bus as soon as one sector has been read into the cache. If the fence value is 255, the disk waits until 255/256 of the requested number of sectors are in the cache (or until the cache becomes full).

When any of the disks in our study reaches a track boundary during a data transfer from the disk surface to the host (through the cache), the disk disconnects from the bus. When the Wren disk reaches a cylinder boundary while performing readahead into the disk cache, readahead stops. In the more modern Barracuda and Cheetah disks, readahead can continue across a cylinder boundary, although these disks must detect a sequential access pattern before they will commence readahead.

We use three synthetic workloads: one consists of random, fixed-sized reads, another consists of fixed-sized reads randomly distributed on a subset of the cylinders of the disks, and the third consists of random reads where the size is uniformly distributed. In each workload, the requests are directed to a collection of independent disks that share a SCSI bus. The requests are generated by multiple processes of equal priority running concurrently on a uniprocessor, one process per disk. Each process executes a tight loop that generates a random block address on its disk, takes a timestamp, issues a seek and a read system call to the raw disk (bypassing the file system), and takes another timestamp when the read request completes. Each experiment consists of three phases: a startup period during which requests are issued but not timed, a measurement period during which the timings are accumulated in tables in main memory, and a cooldown period during which requests continue to be issued. The purpose of the startup and cooldown periods is to ensure that the I/O system is under full load during the measurements. We observed that the I/O systems provided fairness in all our experiments: each disk completed approximately the same number of I/Os.

These workloads capture the access patterns of some external-memory algorithms designed for the Parallel Disk Model, such as merge sort [5] and matrix multiplication [21]. In Parallel Disk Model algorithms, reads and writes are concurrent requests to a set of disks, issued in lock-step, one request per disk. Since more than D blocks are normally needed per compute step, the requests to the disks appear in "stages", where all of the requests in a stage have zero think-time. Our workloads also model applications that use balanced *collective I/O*, i.e., where all processes make a single joint I/O request rather than numerous independent requests. The workloads also can be used to model a video-on-demand server that stripes data across multiple disks and serves many clients [14]: even though each video file is read sequentially, the large number of concurrent clients make the requests appear random at the disks.

Table 2: Disk parameters.

parameter	Cheetah	Barracuda	Wren
maximum disk queue length	64	64	1
average seek latency (ms)	7.7	9.4	15
rotational speed (revolutions per minute)	10033	7200	3597
average surface transfer rate (MB/s)	13.2	6.25	2.1
sector size (bytes)	512	512	512
average sectors per track	170	163	70
number of data surfaces	8	5	15
disk buffer size (KB)	512	512	240

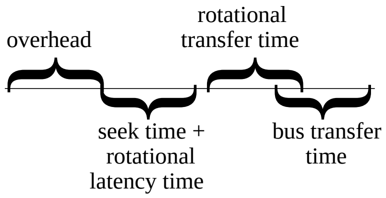

Figure 1: Time line of read duration (single disk).

4 Performance model

In this section, we first define the components of the read service time, and then describe the convoy behavior that we observed. Next, we present a performance model for one or more disks that share a SCSI bus. (The single disk model is merely a stepping stone for the development of the multiple disk model. A more accurate single disk model can be derived from [19].) For the workloads described in Section 3, our model predicts the average time to complete a read request, as a function of the number of disks sharing the bus, the request size in bytes, the setting of the fence parameter (0–255), and the performance parameters for the SCSI bus and the individual disks. Finally, we describe our experiments validating the model.

We will refer to the entire quantity of data in a request as a "data block". This is not to be confused with the file system notion of a "disk block" (a unit of space allocation).

4.1 Components of service time

Table 3 presents the significant components of the time to complete a read operation. The service time for a disk request is not simply the sum of these components. For instance, if the fence is 0, some of the rotational transfer time may be overlapped with the bus transfer time, as seen in Figure 1. Moreover, under different scenarios, different terms may dominate. If many disks share a bus, the overlapped I/O transfers may cause the bus busy time to dominate, leading to service times much larger than the bus transfer time. If the I/O requests are small, then the SCSI overhead, seek time, and rotational latency time may dominate, in which case the effective data rate on the bus cannot approach the bus bandwidth, even if many disks share the bus.

Since our experiments have at most one outstanding request per disk, both the host queue time and the device queue time are zero.

4.2 Rounds

In our experiments, we observed fairness in the servicing of sufficiently large I/O requests, despite the fact that the SCSI disks have different priorities when contending for the bus. Although each process attempts to progress through its requests as fast as possible, without coordinating with other processes, we typically observe a convoy behavior among the requests by *all* the processes. Namely, all disks receive a request, then all disks transmit data back to the host before any disk receives another request. We use the term *rounds* for this periodic convoy behavior.

We were surprised to see rounds. Since the host has the highest SCSI priority, one would expect that soon after a disk completes one request, the host would seize the bus to send another request to that disk, thereby keeping the bus and all the disks busy.

We observed rounds on a variety of hardware architectures (Sun, DEC, Intel-PC) and SCSI controllers, running several different versions of UNIX (Solaris, Ultrix, and NetBSD)[3]. By instrumenting the NetBSD kernel, we determined that the operating system does not queue requests before sending them to the host bus adapter. We then used an Ancot SCSI bus analyzer to determine whether the host bus adapter arbitrates for the bus when it has a disk request. We found that it does not: if any disk wants the bus, the host bus adapter will not arbitrate for the bus, even if it is waiting to send a request to an idle disk. The host bus adapter only arbitrates for the bus if no disk is arbitrating.

Since using the SCSI bus analyzer is time-intensive, we determine when rounds happen using application-level timestamps. To determine whether D disks are being served in rounds under some workload, we examine the ordered I/O completion timestamps from a 10 second run (after a startup interval) using a sliding window of size D. A violation of round ordering is said to occur on the jth timestamp in the window (where $0 \leq j \leq D-1$) if there is an $i < j$ such that the ith and jth I/O of the window both originate from the same disk: if the current sliding window contains a violation at the jth position, the window is advanced by j positions. Otherwise it is advanced by D positions. The fraction of I/O operations that do not violate round ordering is our measure of the extent of round formation for that experiment; we also validate that the number of requests serviced by each disk are within a few percent of each other. Measured this way, rounds occurred 88–99% of the time with the fence set to both 0 and 255, for uniform random workloads containing a mixture of 1, 2, 3, or 4 different request sizes and for workloads that have spatial locality. The workloads that we experimented with have request sizes of $B, \ldots, iB$, for i the number of request sizes in the workload and for $B = 8, 16, 32, 64$, or 128 KBs.

The Wren disks exhibit rounds for requests of size 8 KB

[3] Nils Nieuwejaar of Sun replicated our experiment on a 26-CPU Sun E6000 with Seagate Cheetah (ST-19171FC) Fibre Channel disks. Using one CPU to send requests to four disks on one Fibre Channel loop, we saw rounds for request sizes 16–200 KB.

Table 3: Significant components of the read service time.

component	description
host queue time	the time during which a request remains queued up in the device driver or the host controller, which depends on the particular strategy employed in those components.
SCSI overhead	the time for the SCSI protocol to send a request from the device driver to the disk, denoted as Overhead in the equations of Section 4.4 and 4.5. This includes HBA and disk controller overheads.
device queue time	the time that a request waits in the disk controller while the controller is serving some previous request.
seek time	the time required by the disk head to move to the track containing a requested data block address. It has a non-linear dependency on the number of tracks traversed.
rotational latency time	after a seek completes, the time during which the disk rotates to position the disk head at the start of the data block.
rotational transfer time	after the rotational latency completes, the time required for the head to transfer data from the disk surface to the disk buffer. This time is largely governed by the speed of rotation and the number of bytes per track. It is proportional to the number of bytes transferred, and includes any additional time required for track and cylinder switches when the data block extends across multiple tracks or cylinders.
bus busy time	the time period during which (some or all of) the data block resides in the disk's buffer, waiting for the SCSI bus to become available for a transfer to the host.
bus transfer time	the time required to transmit the data block over the SCSI bus, at the sustained bus bandwidth, from the disk to the host. It is proportional to the number of bytes transferred.

or larger (see [3]). The Barracudas and Cheetahs exhibit rounds with requests of size 16 KB or larger. If the request size is very small, we do not observe rounds. In this case, the bus is not a bottleneck, but the random seeks and rotational latencies cause reasonably fair workloads on the disks.

The current literature does not discuss rounds as we observed them. In fact, [20] states that even in cases when the load is symmetrically distributed and balanced, one process can monopolize the disks while all other processes starve.

4.3 Read duration

The experiments described in Section 3 collect timestamps immediately before a read call is made and immediately after the read call returns.[4] We use the term *read duration* for the time period between such a pair of timestamps.

The average read duration is closely related to the throughput at which data blocks are being retrieved by the parallel I/O system. For workloads that have zero think-time between requests for data blocks of size B bytes, retrieved in parallel from D disks, the average throughput in bytes per second is $DB/\mathsf{ReadDuration}$, where ReadDuration is the average read duration, computed over all the read calls of all the processes.

The bus bandwidth places an upper bound on performance. If the bus were continuously busy transferring data, the corresponding average read duration would equal the time required to transmit D data blocks. This scenario would correspond with a situation in which all the other components of the service time for a disk (e.g., seek time, SCSI overhead, bus busy time, rotational latency and transfer) are overlapped with the data transfers of the other disks that share the bus. Thus, the average read duration is an indicator of how effectively the disk latency is hidden by other concurrent transfers on the bus.

[4] The preemption of a process during the brief time interval between the return of a read call and the invocation of the next read call is sufficiently rare to be negligible. For example, in our Cheetah/Ultra-1 environment, the delay between completion and initiation exceeds 50 ns in only 47 out of 35000 I/Os.

4.4 Single disk model

In this section we present our model of read duration when only a single disk is active.[5]

Read duration for fence value 0. When the fence value is 0, the disk requests the bus as soon as the first sector is in the disk cache. After the first sector has been transferred to the host, the transfer of the remainder of the data occurs at the surface-to-cache rate ($\mathsf{bandwidth}_{rot}$), which is typically smaller than the cache-to-host rate ($\mathsf{bandwidth}_{bus}$). When using only a single disk, the average time to read a data block of size B ($B \gg 1$ sector) is well approximated by

$$\begin{aligned}\mathsf{ReadDuration} &= \mathsf{Overhead} + \mathsf{E[SeekTime]} \\ &+ \mathsf{E[RotationalLatency]} + \frac{B}{\mathsf{bandwidth}_{rot}} \\ &+ \mathsf{TrackCylinderSwitchTime}. \end{aligned} \tag{1}$$

This equation approximates the average read duration as the sum of the SCSI protocol overhead time, the expected seek time, the expected rotational latency, the time to read the data from the disk surface, and the track and cylinder switching time (defined below). The data is transferred over the bus at the rotational transfer rate; this is due to the disk cache being used only as a speed matching buffer.

When B is large, the requested data will extend over a number of tracks and possibly cylinders. Let TrackSwitchTime and CylinderSwitchTime represent the amount of time to perform one track switch and one cylinder switch, respectively. We can approximate the number of cylinder switches by $B/\mathsf{AverageCylinderSize}$, and the number of track switches (including those that also cross a cylinder boundary) by $B/\mathsf{AverageTrackSize}$. Let TrackCylinderSwitchTime be the sum of the track and cylinder switch times, given by

$$\mathsf{TrackSwitchTime}\left(\frac{B}{\mathsf{AverageTrackSize}} - \frac{B}{\mathsf{AverageCylinderSize}}\right) + \mathsf{CylinderSwitchTime}\,\frac{B}{\mathsf{AverageCylinderSize}}.$$

[5] We do not model *zero-latency reads*, a feature of obsolete disks in which the transfer of data from the disk surface to the disk cache begins as soon as the disk head settles—modern disks wait for the beginning of the data block before commencing data transfer.

Read duration for non-zero fence value. When the fence is set to a non-zero value, a fraction of the data is read into the disk drive's cache before the bus is requested. Data is transferred first from the disk surface into the disk cache (at $\mathsf{bandwidth}_{rot}$), and then over the bus at the cache-to-host rate ($\mathsf{bandwidth}_{bus}$). When the data is going over the bus to the host, either the rest of the surface-to-cache transferring will be hidden by the cache-to-host transfer, or the cache-to-host transfer will be visible.

Let $B_c = B \cdot (\mathsf{FenceValue}/256)$ represent the number of bytes in the disk cache before the bus is requested. When using only a single disk, the average time to read a data block of size B is

$$\begin{aligned}\mathsf{ReadDuration} &= \mathsf{Overhead} + \mathsf{E}[\mathsf{SeekTime}] \\ &+ \mathsf{E}[\mathsf{RotationalLatency}] + \frac{B_c}{\mathsf{bandwidth}_{rot}} \\ &+ \mathsf{TrackCylinderSwitchTime} \\ &+ \max\left(\frac{B}{\mathsf{bandwidth}_{bus}}, \frac{B - B_c}{\mathsf{bandwidth}_{rot}}\right). \end{aligned} \tag{2}$$

Modification for multiple request sizes. For multiple request sizes, the models presented in equations (1) and (2) are extended by taking the weighted average of the read durations for each request size.

Modification for spatial locality. If the workload requests are not distributed across the entire disk, but instead are confined to a contiguous subset of the disk cylinders, the expected seek time used in equations (1) and (2) is calculated over that number of cylinders. Also, if the location of the cylinders is known, the average bytes per track (which is used in computing $\mathsf{bandwidth}_{rot}$) can be computed using the number of bytes in those cylinders.[6]

4.5 Multiple disk model

As explained in Section 4.2, we observe the formation of rounds of I/O transactions. In each round, one request is served from each disk. When the fence is 0, a disk is ready to transfer data to the host after it has positioned its head to the data and read the first sector into its disk cache. This time is dominated by the positioning time, which greatly exceeds the rotational transfer time for one sector. Transmission of data to the host begins when any one of the disks is ready, so on a bus with D disks, the idle time on the bus at the beginning of a round is well approximated by the expected minimum positioning time, denoted $\mathsf{MPT}(D)$.

Parallel read duration for fence value 0. The general scenario in a round is as follows. One request is sent to each of D disks. Usually the requested data blocks are not in the disk caches, so the drives disconnect from the SCSI bus. After the smallest of the D positioning times, that disk reads the first sector into its cache, and reconnects to the host. It transmits the first sector at $\mathsf{bandwidth}_{bus}$, and then continues transmitting at $\mathsf{bandwidth}_{rot}$. After sending some data to the host, the disk disconnects, either because it has transferred the entire data block, or because the remaining portion of the data block lies on the next track or cylinder. By the time this disconnection occurs, it is likely (this is quantified below) that other drives have read enough data into their disk caches that the remaining portion of the D data blocks can be sent to the host at $\mathsf{bandwidth}_{bus}$. There may be several disconnects during this transmission, as various drives reach track or cylinder boundaries, but as soon as one drive disconnects, another reconnects to continue sending data to the host. We denote the average size of the leading portion of the first data block (i.e., the amount transferred prior to the first disconnection) by B_t.

We make two simplifications. Although the first disk sends one sector at $\mathsf{bandwidth}_{bus}$ before sending more at $\mathsf{bandwidth}_{rot}$, we say that the entire leading portion from the first disk is sent at $\mathsf{bandwidth}_{rot}$. Second, the overhead of the disconnection and reconnection is sufficiently small that it is absorbed into the overhead term. Thus the average read duration is given by

$$\begin{aligned}\mathsf{ReadDuration} &= \mathsf{Overhead} + \mathsf{MPT}(D) \\ &+ \frac{B_t}{\mathsf{bandwidth}_{rot}} + \frac{DB - B_t}{\mathsf{bandwidth}_{bus}}. \end{aligned} \tag{3}$$

When the request size B is small, it is usual for the entire data block to reside on a single track, whereas for large request sizes the expected size of the leading portion is one half the track size. Thus $B_t = \min(B, \mathsf{AverageTrackSize}/2)$.

Note that equation (3) does not contain terms to account for the track and cylinder crossings as equations (1) and (2) do. These crossings do not add to the read duration because the bus remains busy: one disk disconnects and another disk immediately seizes the bus to send its data to the host.

Parallel read duration for non-zero fence value. In this case, the bus is idle during the shortest positioning time, then the bus continues to remain idle while that disk reads $B_c = B \cdot (\mathsf{FenceValue}/256)$ bytes of the B bytes into its cache. Next the bus transmits those bytes to the host, followed by the rest of the data block and the data blocks from the other $D - 1$ disks. Thus the average read duration in this case is given by

$$\begin{aligned}\mathsf{ReadDuration} &= \mathsf{Overhead} + \mathsf{MPT}(D) \\ &+ \frac{B_c}{\mathsf{bandwidth}_{rot}} + \frac{DB}{\mathsf{bandwidth}_{bus}}. \end{aligned} \tag{4}$$

In (4), we have abstracted away the terms that describe the detailed behavior of the first disk. If present, these terms would make (4) look more like equation (3). Our experiments show that this equation is accurate to within 9.1% for fence values greater than or equal to 50.

Impact of rounds on fence effects and throughput. Note that a higher fence value would increase overall throughput if the time to read the B_c bytes into the cache at each disk were fully overlapped with bus bandwidth transfers by other disks. Since the workload attempts to keep all disks busy, one might expect that the fully overlapped scenario would occur. However, in each round the first read (as well as the corresponding positioning time) is not overlapped, so that smaller fence values result in higher throughput, even with the aggressive workload.

Expected minimum positioning time. Consider a system consisting of D disks where each disk receives a random request at approximately the same time. Let ST be the random variable denoting the seek time of one disk and let

[6] In modern disks, the length of the track increases as the radius of the cylinder increases, and hence outer cylinders can hold more sectors. Consequently, the set of cylinders is partitioned into *zones*. Within each zone, all tracks have the same number of sectors. Thus, within a zone, $\mathsf{bandwidth}_{rot}$ is constant.

MST_D be the random variable denoting the minimum seek time for a D-disk system. The expected minimum positioning time can be approximated as the sum of the expected minimum seek time and the mean rotational latency:

$$\mathsf{MPT}(D) = \mathsf{E}[\mathsf{MST}_D] + \mathsf{E}[\mathsf{RotationalLatency}]. \tag{5}$$

The rest of this section determines the value of $\mathsf{E}[\mathsf{MST}_D]$. Since the D disks are independent and have identical seek curves[7], we have

$$\mathsf{Pr}[\mathsf{MST}_D \geq z] = (\mathsf{Pr}[\mathsf{ST} \geq z])^D. \tag{6}$$

Let $\mathsf{cylinder}[t]$ represent the number of cylinders that the disk head can move past in t time; given by

$$\mathsf{cylinder}[t] = \begin{cases} \left(\frac{t-a}{b}\right)^2 & a < t < \mathsf{SeekTime}[e] \\ \left(\frac{t-c}{d}\right) & \mathsf{SeekTime}[e] \leq t \end{cases}$$

where the seek curve of the disk is defined as

$$\mathsf{SeekTime}[\mathsf{dis}] = \begin{cases} 0 & \mathsf{dis} = 0 \\ a + b\sqrt{\mathsf{dis}} & 0 < \mathsf{dis} \leq e \\ c + d\,\mathsf{dis} & \mathsf{dis} > e \end{cases} \tag{7}$$

where a, b, c, d, and e are device-specific parameters and dis is the number of cylinders to be traveled. Let SD be the random variable for the seek distance (measured in cylinders); by definition, $\mathsf{Pr}[\mathsf{ST} \geq t] = \mathsf{Pr}[\mathsf{SD} \geq \mathsf{cylinder}[t]]$. Let MaxCyl be the maximum number of cylinders. By equation (4.5) in [19], we have

$$\mathsf{Pr}[\mathsf{SD} \geq x] = \left(1 - \frac{x}{\mathsf{MaxCyl}}\right)^2. \tag{8}$$

Using (6) and (8), we have

$$\mathsf{Pr}[\mathsf{MST}_D \geq z] = \left(1 - \frac{\mathsf{cylinder}[z]}{\mathsf{MaxCyl}}\right)^{2D}. \tag{9}$$

Using the definition of expectation for a finite continuous real random variable and (9), we have that $\mathsf{E}[\mathsf{MST}_D]$ is

$$\int_0^\infty \mathsf{Pr}[\mathsf{MST}_D \geq z]\,dz = \int_0^\infty \left(1 - \frac{\mathsf{cylinder}[z]}{\mathsf{MaxCyl}}\right)^{2D} dz. \tag{10}$$

Assuming the three-part seek curve as presented in equation (7), (10) can be simplified to

$$a + b\sqrt{\mathsf{MaxCyl}} \cdot \sum_{i=0}^{2D} \binom{2D}{i} (-1)^i \frac{\sqrt{e/\mathsf{MaxCyl}}^{\,2i+1}}{2i+1} + \frac{d\mathsf{MaxCyl}}{2D+1}\left(1 - \frac{e}{\mathsf{MaxCyl}}\right)^{2D+1}. \tag{11}$$

[6] presents an equation for $\mathsf{E}[\mathsf{MST}_D]$ for a linear seek curve. We found (11) to be more accurate than the equation in [6] ($\mathsf{SeekTime}[\mathsf{MaxCyl}/(2D+1)]$) for larger values of D.

Modification for multiple request sizes. For multiple request sizes, the models presented in equations (3) and (4) are extended by taking B and B_t as weighted averages over the request sizes.

[7] The effect on the average read duration of seek curve anomalies (e.g., those that arise from remapping bad sectors) is nil.

Modification for spatial locality. If the workload requests are not distributed across the entire disk, but instead are confined to a contiguous subset of each disk's cylinders, the expected seek time is calculated over that number of cylinders, by setting MaxCyl accordingly.

4.6 Validation

To validate the four performance equations of the previous section, we ran experiments using the workload and hardware configurations described in Section 3 and measured the average read durations. The experiments contained startup and cooldown phases to ensure that the bus experienced a typical load during the measurement period. The measurement period was 10 seconds, which is sufficiently long to give reproducible measurements. We used three different workloads with the request sizes varying from 16 KB to 128 KB, which is the range of request sizes for which the model is accurate. We believe that the cause for model inaccuracy for large requests is the fact that the operating system splits a large application request into multiple physical I/Os[8]. The three different workloads are: fixed-sized read requests where the destination is uniformly distributed across the disk, fixed-sized read requests where the destination is uniformly distributed across a subset of cylinders of the disk, and (only on the Sun Ultra / Seagate Cheetah testbed) multiple-sized read requests drawn from a uniform distribution. We varied the values of D from 1 to the maximum for that hardware configuration (4 or 7 disks on a SCSI bus). Using the Sun Ultra with Cheetah disks, we also studied the effect of setting the fence value to 0, 50, 120, 200, and 255. We compared the measured average read durations with the values predicted by equations (1)–(4) to calculate the relative model error. In summary, the results are as follows. For request sizes 16 KB to 128 KB, the *maximum* relative error on the Wren disk for the single disk model is 6.4% and for the multiple disk model is 9.1%. The maximum errors for the Barracuda are larger: 8.7% for the single disk model and 10.8% for the multiple disk model. The maximum errors for the Cheetah are similar to the Barracuda: 8.8% for the single disk model and 9.1% for the multiple disk model. A representative sampling of our detailed results is in [4].

We also validated a model inspired by the multiple disk model in Section 4.5 on CineBlitz, a media-on-demand server based on Fellini [14]. We had to improve the approximation of the expected minimum positioning time in equation (5) to include the expected minimum rotational latency as a function of the number of disks. The hardware configuration that we used was an SGI Challenge running Irix 6.5 with two Seagate Barracuda (ST-15150WD) disks. CineBlitz serves a number of clients during a *period*; we consider a heavy load to be one where a large number of clients are being serviced. Our maximum errors were 4% for a heavy load of 50–320 clients with files striped on the two disks. Even though each video file is read sequentially, the requests appear random at the disk due to the large number of files.

Our model is accurate for large I/Os (i.e., 16 KB or larger). The model counts the positioning time of only one disk in the read duration, under the assumption that the others will be overlapped with the positioning time and bus transfer time of the first disk. This assumption is not true

[8] We observed this using the SCSI bus analyzer.

for transfers of small data blocks on fast disks—the transfer of the first small data block sometimes completes before any other disks are ready to use the bus, increasing the amount of bus idle time. If a more accurate model is needed, it is possible to improve the accuracy by hybrid analytical/simulation modeling techniques [19].

Equations (1)–(4) require a number of disk-specific parameters such as $\mathsf{Overhead}$, $\mathsf{bandwidth}_{rot}$ and $\mathsf{bandwidth}_{bus}$. We measured some of these values experimentally, and used the device specifications for others (see [4]).

5 Pipelining to reduce disk latency

The model equations (3) and (4) suggest two ways to decrease the read duration. We can decrease the minimum positioning time, and we can convert those transfers that occur at $\mathsf{bandwidth}_{rot}$ to the faster $\mathsf{bandwidth}_{bus}$.

Assuming that during iteration $j-1$ we know the data blocks that will be requested during iteration j, we propose using a simple pipelining technique to overlap the positioning time for iteration j with the transfer time of the previous iteration. Furthermore, this pipelining technique stages data in the disk caches, so that the first data block transmitted during iteration j is sent from cache at $\mathsf{bandwidth}_{bus}$, rather than from the disk surface at $\mathsf{bandwidth}_{rot}$. Let $b_{i,j}$ denote the data block to be retrieved from disk i in round j. Then the pipelining technique schedules the SCSI bus as follows.

for $0 \le i \le D-1$
  Request LoadIntoDiskBuffer($b_{i,0}$) on disk i.
for $0 \le j \le$ NumRequests
  for $0 \le i \le D-1$
    Read($b_{i,j}$) from disk i. *//already in disk i's buffer*
    Request LoadIntoDiskBuffer($b_{i,j+1}$) on disk i.

LoadIntoDiskBuffer(b) causes the disk to prefetch data block b into its cache so that a subsequent Read(b) to that disk will not incur disk head positioning time or a head-limited transfer rate. The prefetch occurs while the bus is busy transmitting data blocks from other disks and from the previous round. Thus, the random access latency is overlapped with bus transfers, and the bus transfers occur at the higher cache data rate, rather than the slower disk-head rate. The result is fair parallel I/O in rounds, with a high aggregate bandwidth for random I/O.

This simple pipelining technique contrasts with the more involved SCSI bus scheduling algorithm proposed in [10], which considers deadlines for requests, sends out multiple prefetch requests per disk, and performs the reads according to a token-based scheme with two tokens, in order to optimize simulated performance for a video server workload.

The `SCSI PREFETCH` command would implement LoadIntoDiskBuffer, but it is an optional command, not supported by the Wren, Barracuda, or Cheetah disks. We implement LoadIntoDiskBuffer(b) by an `aioread()` (a non-blocking read), of the disk sector just before b, which triggers the disk readahead mechanism to load b into the disk cache. For each data block, the `aioread()` implementation incurs the overhead of sending an extra SCSI request to the disk and receiving the unwanted sector that triggers the disk readahead. The `SCSI PREFETCH` implementation would only have the overhead of sending one extra SCSI request for each data block.

Note that, unlike the scenario of the previous sections, when a Read request is about to be issued, there are no outstanding Read requests. If the `aioread()` is used to implement the LoadIntoDiskBuffer command, there may be a few outstanding responses to the 1-sector `aioread()`'s that need the bus. Since these requests occupy the bus for only a couple of milliseconds, each Read or LoadIntoDiskBuffer request is able to grab the bus with negligible delay, unlike the scenario of the previous sections.

We now present the results of experiments on several hardware configurations to see whether the performance gain from pipelining outweighs the additional overhead of the `aioread` implementation of LoadIntoDiskBuffer.

Table 4 evaluates the effectiveness of the pipelining technique with 2, 4, and 7 Wren disks on a Sparc-20, using transfers ranging from 8 KB to 128 KB. The measurements are averaged over 1000 I/Os. The table compares the aggregate transfer rate in MB/s achieved by the "naive" approach (one process per disk performing random I/Os) with the pipeline technique. The column labeled "%" contains the relative improvement (in percent) of the pipeline technique. With small data block sizes, the overhead outweighs the improvement. With 7 disks, the disk positioning time is small compared to the read duration, leaving little room for improvement. With 2 or 4 disks and moderate or large data block sizes, the overlaps gained by the pipeline technique more than compensate for the increased overhead. For example, with 4 disks and 64 KB data blocks, the bandwidth improves from 2.42 MB/s to 2.72 MB/s.

Table 5 presents the results of the experiments (averaged over 300 I/Os) on the DEC Alpha with Seagate Barracuda disks. For large transfers and many disks sharing the bus, the gain achieved by pipelining is greater than the overhead of the `aioreads`.

When we run the experiments using Cheetah disks on a Sun Sparc-20, we do not see improvements with the pipeline method. We attribute this to the host overhead for the `aioread()` command, which we measure to be 2.1–2.3 ms (approximately equivalent to the amount of time needed to read 16–32 KB from disk cache to host). As shown in Table 6, the faster Sun Ultra-1 reduces the overhead sufficiently to make the pipeline technique viable for large data block sizes. If the disks supported `SCSI PREFETCH` with a command overhead equal to that of a seek, we would expect to see improvements of 10–30% for pipelining over the naive method for workloads with request sizes of 8 KB and larger. We observe that faster CPUs would decrease the overhead of pipelining, making the technique beneficial for larger ranges of workloads.

We also compare our results to what might happen if rounds did not happen. We compute the ideal throughput, assuming that rounds do not happen; Figures 2 and 3 show this for the Cheetah / Ultra-1 experiments, compared against the throughput attained by the naive and pipeline methods. Figures 2 and 3 show that the pipeline approach improves over the naive method, but there is still room for improvement before the ideal throughput is reached.

Our results for various block sizes are consistent with the findings in [10] that the use of `SCSI PREFETCH` is not recommended for block sizes of less than 64 KB.

Consider the concurrent I/O requests generated by I/O-intensive algorithms using asynchronous I/O calls or multiple processes or threads. These I/O requests all funnel through the disk device driver code. Depending on the I/O sizes and the disk configuration, it may be possible to improve the performance of such algorithms by interposing an I/O scheduling thread that causes pipelining by issuing

Table 4: MB/s for naive and pipelined I/O, fence 0, Wren/Sun Sparc-20.

Data block size (KB)	D=2			D=4			D=7		
	Naive	Pipeline	%	Naive	Pipeline	%	Naive	Pipeline	%
8	0.52	0.48	-8	0.97	0.85	-13	1.43	1.22	-15
16	0.89	0.83	-7	1.53	1.49	-3	1.89	1.93	2
32	1.37	1.40	2	1.97	2.19	11	2.35	2.45	4
64	1.88	2.05	9	2.42	2.72	12	2.80	2.98	6
96	2.15	2.42	13	2.65	2.89	9	3.00	3.09	3
128	2.26	2.53	12	2.68	2.98	11	2.93	2.90	-1

Table 5: MB/s for naive and pipelined I/O, fence 0, Barracuda/DEC Alpha.

Data block size (KB)	D=2			D=3			D=4		
	Naive	Pipeline	%	Naive	Pipeline	%	Naive	Pipeline	%
8	1.05	0.93	-11	1.58	1.36	-13	2.10	1.78	-15
16	1.96	1.75	-11	2.95	2.62	-11	3.90	3.30	-15
32	3.49	3.15	-10	5.18	4.66	-10	6.78	6.06	-11
64	5.51	5.19	-6	7.97	7.43	-7	9.76	9.92	2
96	6.87	6.48	-7	9.58	9.69	1	11.25	12.51	11
128	7.87	7.60	-3	10.35	11.19	8	12.13	14.40	19

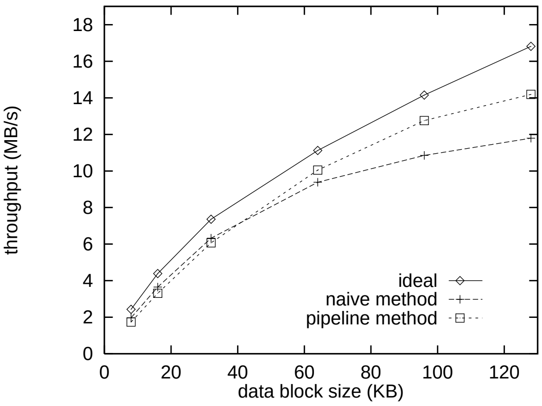

Figure 2: The throughput of naive and pipeline methods, compared to the ideal throughput, with $D = 3$.

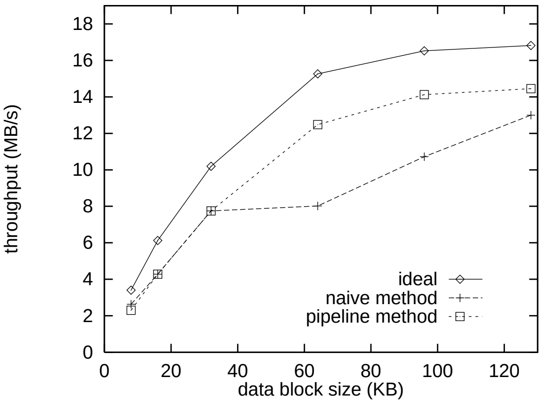

Figure 3: The throughput of naive and pipeline methods, compared to the ideal throughput, with $D = 4$.

the appropriate LoadIntoDiskBuffer operations. Similarly, it may be possible to improve the performance of a RAID disk array by an internal implementation of pipelining.

Pipelining is complementary to double-buffering I/O requests from the host. Double-buffering overlaps I/O operations with host computation, but does not decrease the time required for each I/O.

6 Conclusions

By measuring the performance of I/O-intensive workloads on several UNIX systems from various manufacturers and operating system vendors, we observed a convoy behavior that we call "rounds" in the completion of I/O requests directed to multiple disks that share a SCSI bus. We have developed an accurate model of the I/O bandwidth achieved by multiple disks that share a SCSI bus for a variety of random-access workloads. This model quantifies the performance impact of rounds, and explicitly displays two opportunities for performance improvement. Although the current literature does not discuss rounds as we observed them, our studies indicate that they are essential to understanding the aggregate and per-disk I/O bandwidth of multiple disks on a bus under I/O-intensive workloads.

The performance advantage of scheduling within the request queue for each disk is well known. We have shown scenarios for which coordinating the accesses *across* a collection of disks that share a SCSI bus can also improve performance. The scheduling is performed by an application-level pipelining technique, which can increase the aggregate disk bandwidth on the shared SCSI bus by increasing the overlap between disk seeks and data transfers, and by increasing the fraction of transfers that occur at the disk cache transfer rate rather than the slower disk-head rate. Moreover, the pipelining technique enables the disks to be self-governing, in that we do not need to predict the positioning time that will be incurred by each I/O request. Experimental results show the performance regions where pipelining improves overall performance 10–20% despite the associated overhead.

In each of our experiments there is at most one request in the I/O path for each disk. As future work, we would like to extend our pipelining technique to incorporate the advantages of disk controller command queuing to schedule multiple requests per disk. Preliminary experiments suggest that

Table 6: MB/s for naive and pipelined I/O, fence 0, Cheetah/Sun Ultra-1.

Data block size (KB)	D=2			D=3			D=4		
	Naive	Pipeline	%	Naive	Pipeline	%	Naive	Pipeline	%
8	1.32	1.22	-8	1.97	1.74	-12	2.63	2.30	-13
16	2.48	2.34	-6	3.65	3.31	-9	4.83	4.28	-11
32	4.43	4.27	-4	6.32	6.07	-4	8.02	7.75	-3
64	7.08	7.09	0	9.38	10.05	7	10.72	12.48	16
96	8.76	9.48	8	10.85	12.76	18	12.09	14.12	17
128	9.86	11.01	12	11.79	14.19	20	13.00	14.45	11

pipelining may be beneficial for applications such as video-on-demand servers, which can generate batches of requests for each disk. Other future work would model the performance of small request sizes ($\leq$ 8 KB) and write requests. Writes are a challenge to model because of the complexity of write-to-disk policies. Our model could also be extended to approximate wall clock time in a system-dependent fashion, furthering the results of [11].

Acknowledgments. Many thanks to Jeff Chase, Andrew Gallatin, Cliff Martin, and Nils Nieuwejaar. Cliff Martin helped set up our machines at Bell Labs and Andrew Gallatin helped set up our machines at Duke University. Jeff Chase and Andrew Gallatin participated in many helpful discussions. Nils Nieuwejaar ran the experiments on the Sun E6000. Philip Bohannon, Cliff Martin, and Mary McShea provided the necessary details of CineBlitz and useful discussions.

References

[1] Arunachalam, M., Choudhary, A., and Rullman, B. A prefetching prototype for the parallel file system on the Paragon. In *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (May 1995), ACM Press, New York, NY, pp. 321–323. Extended Abstract.

[2] *Specifications for ST-32171W.* Found at `http://www.seagate.com/cgi-bin/view.cgi?/scsi/st32171w.txt`.

[3] Barve, R., Gibbons, P. B., Hillyer, B. K., Matias, Y., Shriver, E., and Vitter, J. S. Round-like behavior in multiple disks on a bus. In *Proceedings of the Sixth Annual Workshop on I/O in Parallel and Distributed Systems (IOPADS'99)* (Atlanta, GA, May 1999). Available at `http://www.bell-labs.com/~shriver/`.

[4] Barve, R., and Shriver, E. Validation of a multiple disk model. Tech. rep., Bell Labs, 1999.

[5] Barve, R. D., Grove, E. F., and Vitter, J. S. Simple randomized mergesort on parallel disks. *Parallel Computing 23*, 4 (June 1997), 601–631.

[6] Bitton, D., and Gray, J. Disk shadowing. In *Proceedings of the 14th International Conference on Very Large Data Bases (VLDB)* (Los Angeles, CA, Aug. 1988), F. Bancilhon and D. J. DeWitt, Eds., pp. 331–338.

[7] Cao, P., Felten, E. W., Karlin, A. R., and Li, K. Implementation and performance of integrated application-controlled caching, prefetching and disk scheduling. *ACM Transaction of Computer Systems (TOCS) 14*, 4 (Nov. 1996), 311–343.

[8] *Cheetah 4LP family: ST34501 N/W/WC/WD/DC: Product manual, volume 1*, revision a ed.

[9] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., and Patterson, D. A. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys 26*, 2 (June 1994), 145–185.

[10] Chen, S., and Towsley, D. A performance evaluation of RAID architectures. *IEEE Transactions on Computers 45*, 10 (Oct. 1996), 1116–1130.

[11] Cormen, T. H., and Hirschl, M. Early experiences in evaluating the parallel disk model with the ViC* implementation. *Parallel Computing 23*, 4 (June 1997), 571–600.

[12] Hennessy, J. L., and Patterson, D. A. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Incorporated, San Francisco, CA, 1996. 2nd edition.

[13] Kimbrel, T., Tomkins, A., Patterson, R. H., Bershad, B., Cao, P., Felten, E., Gibson, G., Karlin, A. R., and Li, K. A trace-driven comparison of algorithms for parallel prefetching and caching. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation* (October 1996), USENIX Association, pp. 19–34.

[14] Martin, C., Narayan, P., Ozden, B., Rastogi, R., and Silberschatz, A. The Fellini multimedia storage system. In *Multimedia Information Storage and Management*, S. M. Chung, Ed. Kluwer Academic Publishers, Aug. 1996, ch. 5.

[15] Mowry, T. C., Demke, A. K., and Krieger, O. Automatic compiler-inserted I/O prefetching for out-of-core applications. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation* (October 1996), USENIX Association, pp. 3–17.

[16] Patterson, D., Gibson, G., and Katz, R. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Chicago, IL, June 1988), ACM Press, pp. 109–116.

[17] Patterson, R. H., Gibson, G. A., Ginting, E., Stodolsky, D., and Zelenka, J. Informed prefetching and caching. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (Copper Mountain, CO, Dec. 1995), ACM Press, pp. 79–95.

[18] Ruemmler, C., and Wilkes, J. An introduction to disk drive modeling. *IEEE Computer 27*, 3 (Mar. 1994), 17–28.

[19] Shriver, E. *Performance modeling for realistic storage devices.* PhD thesis, New York University, Department of Computer Science, May 1997. Available at `http://www.bell-labs.com/~shriver/`.

[20] Sinclair, J. B., Tang, J., and Varman, P. J. *Placement-related problems in shared disk I/O*, vol. 362 of *The Kluwer International Series in Engineering and Computer Science.* Kluwer Academic Publishers, 1996, ch. 12, pp. 271–289.

[21] Vitter, J. S., and Shriver, E. A. M. Algorithms for parallel memory I: two-level memories. *Algorithmica 12*, 2/3 (August/September 1994), 110–47.

[22] Worthington, B. L., Ganger, G. R., Patt, Y. N., and Wilkes, J. On-line extraction of SCSI disk drive parameters. In *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Ottawa, Canada, May 1995), ACM Press, New York, NY, pp. 146–156.

[23] *Product specification for Wren VII SCSI disk drive Model 94601*, revision a ed.