Round-like behavior in multiple disks on a bus

Rakesh Barve[*]
Duke University
rbarve@cs.duke.edu

Phillip B. Gibbons
Bell Laboratories
gibbons@research.bell-labs.com

Bruce K. Hillyer
Bell Laboratories
bruce@research.bell-labs.com

Yossi Matias[*]
Tel-Aviv University
matias@math.tau.ac.il

Elizabeth Shriver[†]
Bell Laboratories
shriver@research.bell-labs.com

Jeffrey Scott Vitter[*]
Duke University
jsv@cs.duke.edu

Abstract

In modern I/O architectures, multiple disk drives are attached to each I/O bus. Under I/O-intensive workloads, the disk latency for a request can be overlapped with the disk latency and data transfers of requests to other disks, potentially resulting in an aggregate I/O throughput at nearly bus bandwidth. This paper reports on a performance impairment that results from a previously unknown form of convoy behavior in disk I/O, which we call *rounds.* In rounds, independent requests to distinct disks convoy, so that each disk services one request before any disk services its next request. We analyze log files to describe read performance of multiple Seagate Wren-7 disks that share a SCSI bus under a heavy workload, demonstrating the rounds behavior and quantifying its performance impact.

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 that perform I/O-intensive processing are designed to 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 and modeling (e.g., [RW94, Wil95, HP96, Shr97]) has focused on the disk drive, downplaying the importance of bus contention and other bus effects. Indeed, the bus effects play an insignificant role in I/O performance for UNIX workloads with small I/O request sizes. But many I/O-intensive applications benefit significantly from larger requests (32–128 KB). Among these are multimedia servers and certain database and scientific computing applications that use external memory and out-of-core algorithmic techniques to process massive data sets (e.g., [VS94, MNO$^+$96, CH97]). In such applications, parallel I/O performance is often limited by the bus.

Typically, when D disks, each of which can transfer data at rotational media bandwidth $\mathsf{bandwidth}_{rot}$, are attached to a bus, we have $\mathsf{bandwidth}_{bus} \leq D \cdot \mathsf{bandwidth}_{rot}$, where $\mathsf{bandwidth}_{bus}$ is the effective bus bandwidth (i.e., the maximum transfer rate achieved in practice between a particular type of host controller and disk controller). Hence the maximum data transfer rate one can hope to attain from such an I/O system is no more than $\mathsf{bandwidth}_{bus}$. When the application accesses disk blocks sequentially on each disk, a data transfer rate of close to $\mathsf{bandwidth}_{bus}$ can be attained quite easily in practice, owing to useful disk controller features such as readahead. What is not clear is to what extent a similar I/O throughput may be expected even when disk accesses are not sequential. Disk latency tends to idle the bus but, since there are multiple disks operating in parallel, these latencies can *potentially* be overlapped with data transfers, so that at least one disk is transmitting data on the bus at any time.

Our experiments indicate that the attained I/O throughput is far from the desired performance. We study an I/O-intensive workload in which the host generates a new random read request for a disk as soon as it receives the data previously requested from that disk. This workload has the potential to attain a steady state behavior in which all disk latencies are overlapped with data transfers by other disks. This workload also has the potential to exhibit considerable unfairness among the disks, such that disks with higher priority on the bus proceed through their sequence of requests at a faster rate while starving the lower priority disks.

Our experiments show that neither of these potential scenarios arises. Instead, we observe a round-like behavior of I/O service, in which in each round, one request from each disk drive gets serviced, and there is a bus idle period of length approximately equal to the time for a disk to position its head on a random access. We have fairness but at the cost of less than desired performance, thus motivating a need to address scheduling problems related to the sub-optimal performance of such I/O systems. To our knowledge, this particular form of round-like behavior has not been previously reported by others in the literature.

In this paper, we describe the measured read performance of multiple Wren-7 disks that share a SCSI bus under

[*]This work was done when R. Barve and J. Vitter were visiting Bell Labs, and when Y. Matias was with Bell Labs.

[†]Contact author. Room 2A-316, 600 Mountain Avenue, Murray Hill NJ 07974. Phone: (908) 582-3843. FAX: (908) 582-1239.

a heavy workload, demonstrating the rounds behavior and quantifying its performance impact. We first measure key I/O system parameters for a single disk. Then we analyze log files of timestamps collected during multiple-disk I/O experiments to deduce the sequence of I/O events. This analysis reveals the round-like behavior mentioned above. The experimental results show the pervasiveness of rounds in our hardware configuration across a variety of block sizes (16–128 KB) and numbers of disks (2–7 disks). We explain the observed behavior of the I/O system here, including the dependence of the I/O throughput on factors such as the transfer size and the number of disks sharing the bus. Elsewhere, (see [BSG+98, BSG+99]) we develop an accurate analytic model that predicts the performance of multiple disks sharing a SCSI bus (for varying numbers and types of disks on several hardware platforms), and we devise a "pipelining" technique that improves the random access performance.

Related work. Several publications have studied the performance of a parallel I/O subsystem, but none reflect the performance we observed for multiple disks on a bus. The analytic disk model of [Shr97] captures bus effects only in the single-disk case and does not directly model a disk's "fence" parameter (see Section 2). The Pantheon disk simulator [RW94, Wil95] incorporates bus contention and other bus effects, but no results have been published that explain our observations of bus idle periods and dynamic switching between bus-limited and disk head-limited I/O transfers. [HP96] presents a method to approximate 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 contention factor derived from a general queuing model. The Parallel Disk Model (PDM) [VS94] is an abstract model for the design and analysis of algorithms on a parallel disk system. [CH97] present an application-level study of the accuracy of the PDM. None of this previous literature describes the rounds phenomenon that we observe. Ignoring this phenomenon when modeling can result in throughput prediction errors that exceed 100% for the workloads that we consider. [WGPW95] proposes many experiments to measure the performance parameters of an individual disk drive. We use similar approaches to determine certain values used in our analysis of rounds.

Outline. In the next section, we discuss aspects of our hardware configuration and further details on the I/O workload whose performance we study. In Section 3, we discuss certain key performance measures needed for our analysis, and report on their values for our configuration. In Section 4, we describe the round-like I/O behavior occurring with our workload and the techniques we used to confirm the existence of rounds across a variety of block sizes, number of disks, and fence settings. Finally, in Section 5, we make concluding remarks.

2 Hardware configuration and workload

Our storage system consists of 7 Seagate (Imprimis) Wren-7 (94601-15) disks connected to a Sun Sparc 20 running Solaris 2.5. These disks use the SCSI-1 protocol, with a negotiated bus bandwidth of 5 MB/s.

Many kinds of SCSI disks implement a relatively obscure disk control parameter called the *fence*. (The fence is called the *buffer full ratio* on the SCSI-2 disconnect/reconnect 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 data can be transferred. 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. On the other hand, it can decrease overall I/O throughput, because once the disk's cached data has been sent to the host (at the full bus bandwidth), the remainder of the transfer occurs at the (slower) rate at which bits pass under the disk head (the "rotational media bandwidth").

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

We use a synthetic workload in which 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 cool down period during which requests continue to be issued. The purpose of the startup and cool down periods is to ensure that the I/O system is under full load during the measurements.

This workload captures the access patterns of some external-memory algorithms designed for the PDM (e.g., merge sort [BGV97]). In PDM algorithms, reads and writes are concurrent requests to a set of disks, issued in lock-step, one request per disk. This workload can also reflect the access patterns of video-on-demand servers that stripe data across multiple disks and serve many clients [MNO+96]: even though each video file is read sequentially, the large number of concurrent clients make the requests appear random at the disks.

3 Key performance measures

In our experiments, we record timestamps just before and after read calls and then use the logged timestamps to analyze the performance of the I/O system. The *read duration* of a read call is defined to be the time period between the timestamp taken immediately before the read call is made and the timestamp taken immediately after the read call returns. We assume that there is a negligible chance that a process gets context switched in the time interval between the return of a read call and its immediate invocation of the next read call; our experiments support this assumption. As mentioned earlier, when D disks with rotational media bandwidth $\mathsf{bandwidth}_{rot}$ are attached to a bus with bandwidth $\mathsf{bandwidth}_{bus}$ such that $\mathsf{bandwidth}_{bus} \leq D \cdot \mathsf{bandwidth}_{rot}$, the performance of the I/O system would be optimal (for our workload) during a given experiment if the data requested by the read calls were retrieved at a data rate equal to the value $\mathsf{bandwidth}_{bus}$.

We use the read duration measurements of a specially-designed experiment in order to estimate the sustained value of the quantity $\mathsf{bandwidth}_{bus}$, because the information supplied by disk vendors does not reflect system overhead. The experiment used to estimate $\mathsf{bandwidth}_{bus}$ uses a single disk on the SCSI bus and a single process requesting block transfers from that disk. During the experiment, we measure and log the read duration of each read call such that the block requested is already in the disk's cache before the read request reaches the disk drive. (To ensure that the desired block is in the disk cache, we read the block immediately preceding, which triggers the disk's readahead mechanism, and then we wait long enough for the readahead to occur before we issue the request.) Our estimate of the sustained transfer rate $\mathsf{bandwidth}_{bus}$ is the block size divided by the average read duration of these cache hits. Sufficiently large block sizes ensure that the accuracy of this estimate is not impaired by system overheads. In this paper, we use the term "bus bandwidth" to mean this estimate of $\mathsf{bandwidth}_{bus}$, and we assume that it is a constant over all experiments.

The quantity $\mathsf{bandwidth}_{bus}$ obtained by this single disk experiment is expected to be an upper bound on the actual retrieval rate in our parallel disk experiments. For a given parallel disk experiment, we use the read duration measurements to estimate the actual retrieval rate as follows. Suppose that the workload consists of D processes, each repeatedly requesting blocks of size B from a corresponding disk, as described in the previous section. Then our estimate of the data retrieval throughput during the experiment is simply DB times the reciprocal of the average read duration length, where the average is computed considering all read calls of all the reading processes.

There may arise circumstances in which the bus, although not idle, is transferring data not at bandwidth $\mathsf{bandwidth}_{bus}$, but at the significantly smaller bandwidth $\mathsf{bandwidth}_{rot}$. For example, when the fence is zero, the disk drive controller typically tries to seize the bus when the leading sector of the requested block has been read into the disk cache. If the requested block is large, most of the block gets transferred at $\mathsf{bandwidth}_{rot}$. On the other hand, if through a high fence setting, the block resides in the disk cache *before* the disk begins to transfer the data on the bus, then the block gets transferred at $\mathsf{bandwidth}_{bus}$.

When random disk blocks are read, data rates close to the bus bandwidth are attained only for sufficiently large requests and sufficiently many parallel disks. In many hardware configurations, random reads for blocks of size one or two sectors cannot be serviced at bandwidths close to $\mathsf{bandwidth}_{bus}$ because the SCSI command processing overhead at a disk controller already takes more time than the transfer of one or two sectors.

Measured attributes. Using a value for $\mathsf{bandwidth}_{bus}$, we compute the length of time required to transmit a disk block of a given size from the disk drive to the host at bus bandwidth. We also estimate the seek and rotational latency parameters (again using simple single disk experiments). All of the attributes measured with simple single disk experiments are assumed to be invariant over the block sizes requested and the number of disks, and are used as constant values for all of our parallel disk experiments.

The following are the measured values for these parameters on our system configuration.

1. The bus bandwidth for the Wren-7 disks in our system is approximately 3.3 MB/s. Thus, the bus transfer times for blocks of size $16, 32, 64, 96$ and 128 KB are approximately $4.7, 9.5, 18.9, 28.4$ and 37.8 milliseconds, respectively.

2. The seek latency for the Wren-7 ranges from 2.5 to 34 milliseconds, with the average being 15 milliseconds.

3. The average rotational latency is 8.3 milliseconds.

4 Rounds

In our experiments, we observe a periodic convoy behavior, which we call *rounds*, whenever the size of the requested blocks is at least 8 KB. In each round, exactly one requested block from each disk is sent to the host, despite the potential in the workload for disks to service requests at dramatically varying rates. The order of the disks in a round is determined by the positioning time needed to reach the disk's requested block.

The chronological order of important events in a round is shown in Figure 1. The time interval $[s_1, s_2]$ represents the *startup time* during which requests are sent by the host to the D disks. The time interval $[s_2, t_s]$ represents the time during which the bus is idle. At time t_s, some disk, say, i_1, starts transferring its requested block to the host. At time t_{i_1}, the transfer of disk i_1's block completes and some disk i_2 begins transferring its requested block to the host, and so on until the Dth disk, i_D, finishes transferring its requested block to the host at time t_{i_D}. The next round begins immediately after t_{i_D}. In this manner, in each round there is an interval $[s_1, t_s]$ of time in which no requested data blocks get transferred over the bus to the host, followed by the interval $[t_s, t_{i_D}]$ of time in which the D requested data blocks, one from each disk, are transferred to the host over the bus. Our observations indicate that the bus idle time in each round is approximately equal to the average value of the minimum positioning time for D disks that concurrently perform a random access.

The formation of rounds is counter-intuitive. Almost as soon as the read corresponding to disk i_1 of the round is completed, the process corresponding to that disk can generate the next request for that disk drive and pass it on to the host controller. Since the host controller has the highest priority on the bus, it should be able to send the request for disk i_1 before the data from i_3 is received. This would mean that disk drive i_1 would have all of time interval $[t_{i_3}, t_{i_D}]$ in which to position its head and then load the requested data into its cache. For sufficiently large blocks and sufficiently large D, this would mean that in the next round, the idle time $[s_1, t_s]$ would be eliminated or significantly reduced. In fact, one would expect that a steady state would arise in which the bus idle time would be significantly reduced throughout the experiment. Moreover, disks with higher

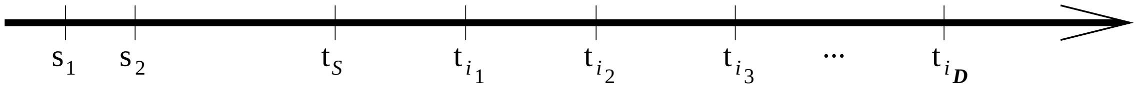


Figure 1: Time line of the important events of a round.

SCSI ids would have more I/Os serviced than disks with lower SCSI ids.

However, our analysis suggests that what is happening is as follows. Although a new read request for a disk is generated as soon as its previous request gets serviced, the host SCSI controller (which has the highest priority on the SCSI bus) sends the new request only after all the requested blocks from the ongoing round that are awaiting the bus complete their transfer back to the host. (In [BSG+98], this behavior is verified using a SCSI bus analyzer.)

When the blocks requested are small, say 512 bytes, we do not observe a round-like behavior. But we do observe fairness: the request service rate is approximately equal over all of the disks. Our explanation is that the bus is almost always idle because the block transfer time is so small, so whenever a read request is generated in any process, the request can be almost instantly sent to the target disk. As a result, we do not have round formation, and since we almost never have contention for the bus among the disks, the bus priorities determined by the SCSI ids have a negligible effect on the disks' service rates.

We now present our techniques for analyzing the measurements recorded during the experiments, and we explain how the analysis reveals the prevalence of rounds. We begin by describing the basic methodology used in the analysis of the timestamp logs.

4.1 Timestamp methodology

The two types of timestamp-recording events are the *start read* event (SR) and the *finish read* event (FR). The SR event is a timestamp recording immediately preceding a read call, and the FR event is a timestamp recording immediately following that read call. The only information we have from the timestamp of an SR event is that the process submitted a read call to the operating system at about that time (and no sooner). Similarly, the only information we have from the timestamp corresponding to an FR event is that the read call returned at about that time (and no later). Since we are dealing with a multi-process system and the read calls are blocking calls, one expects that an SR event also indicates that the read-invoking process is put to sleep at about that time. Likewise, an FR event is likely to indicate that the calling process is awakened due to the completion of its read call at about that time. The time interval from an SR event to the following FR event of the same process is defined to be the read duration of the read call. In all our logs, the time period from an FR event to the following SR event of the same process was of the order of a few microseconds, since the amount of computation carried out during this interval takes a negligible amount of time. Thus an FR event may be used to mark both the return of a read call in the process and the invocation of the next read call of that process. Hence we ignore SR events in the analysis that follows.

During a given experiment, we collect FR events from each process. Then we merge the per-process logs into a single sequence of FR events; each event labeled by the SCSI id of the disk being fed by that process. Since the timestamps originate from a single global clock, this results in a global ordering of all timestamp recording events.

Let *delta* denote a time interval between two consecutive FR events in the merged timestamp file. There is a 1-1 correspondence between deltas and read requests. The goal of our analysis is to associate an I/O-related event (a seek time, transfer time, etc.) with each delta and thereby reconstruct the sequence of events leading to the timestamp sequence. We use the length of a delta and the estimates of key I/O parameters (determined as discussed in Section 3) in order to map I/O events to deltas. We first obtain estimates for the time taken by important I/O events using vendor-specified disk parameters or separately conducted experiments. We can then map an I/O event to a delta when the estimate of the time required for that I/O event is close to the delta time interval. Though this technique is not guaranteed to provide unique reconstructions, in our experiments the quantities involved and the *clustering* of the deltas suggest a relatively-unambiguous reconstruction. We find that the lengths of the deltas in the merged timestamp file exhibit a distinct and consistently repeating pattern. This pattern, when viewed in light of the association between delta lengths and I/O events, reveals the round-like behavior.

We ran experiments on D disks for block sizes of 32, 64, 96 and 128 KB. Throughout these experiments, the delta-lengths clustered around two distinct and well-separated values, which we call the *large delta* and the *small delta*. In all these experiments, we observe a consistently-repeating pattern: one large delta followed by $D-1$ small deltas, and the set of D deltas are labeled with D distinct SCSI ids. We use the term *good round* for any such sequence of D deltas.

The following algorithm decides whether a delta is a large delta or a small delta, and decides whether the current window of D I/Os (the current round) is a good round. The algorithm uses a threshold value to distinguish large deltas from small deltas. We compute various statistics on the deltas, including the standard deviations of the large and small deltas. Variance in these quantities is consistent with the random workload and the imperfect measurement techniques. We find it difficult to determine the source of the variance in the deltas. But in most cases, we find the variance among the deltas to be sufficiently small that the round-like pattern can be considered the typical I/O service pattern.

Delta statistics algorithm:

1. Set *GoodRounds*, *LargeDeltaCounter* and *SmallDeltaCounter* to zero.
2. While there exists a delta in the merged timestamp file
   (a) Get the next delta δ from the file.
   (b) If (δ > Threshold), classify δ as a large delta. Otherwise, classify it as a small delta. Increment *LargeDeltaCounter* or *SmallDeltaCounter* accordingly.

(c) If (δ is a large delta *and* the most recent $D-1$ deltas prior to the current δ were classified as small deltas *and* the most recent previous large delta is the Dth most recent previous delta *and* each of the most recent $D-1$ small deltas correspond to $D-1$ distinct disks other than the disk corresponding to the current δ) then increment *GoodRounds*.

3. Compute the means, standard deviations, and extremal values for the large deltas and the small deltas. Set *OutOf* to $(SmallDeltaCounter+LargeDeltaCounter)/D$.

OutOf indicates the maximum number of possible rounds.

4.2 Random parallel I/O with maximum fence value

In this section, we report on our experiments with the fence set to its maximum value (255/256). Tables 1, 2, 3 and 4 present the delta statistics for block sizes of 32, 64, 96 and 128 KB respectively. In Figures 2(a) and 2(b) we plot, respectively, the percentage of I/Os that form rounds and the I/O bandwidth (in MB/s) obtained for these block sizes. All our experiments illustrate the pervasive presence of rounds but here we first focus on block sizes 32, 64 and 96 KB; the experiments involving 128 KB blocks are discussed at the end of this section since they raise some additional interesting questions.

For 32, 64 and 96 KB block sizes, we conjecture that the small deltas correspond to the event of transferring an entire block from the disk's cache to the host at sustained bus bandwidth rate. We also conjecture that the large deltas correspond to the minimum of D random disk latencies, plus the rotational transfer of one block from the disk surface to the disk's cache, plus the transfer of that block to the host over the bus. We see a pattern of deltas repeated throughout the merged timestamp files: the pattern consists of one large delta followed by a sequence of $D-1$ small deltas.

First we focus on the experiments with $D=4$ or $D=7$, since round formation is most prominent in these experiments. We observe that the mean small delta (Small_{ave}) values for 32, 64 and 96 KB blocks are all very close to the bus transfer times of the same-sized blocks given in Section 3, and that these deltas have small standard deviations (Small_{std}). The experiment used to obtain bus transfer times and the small delta experiments discussed here are very different. The former experiment measures read durations for a single disk when the requested block originates in the disk cache for that disk, whereas the latter experiments involve several processes making concurrent accesses to randomly addressed blocks on multiple disks. The proximity of the average small delta value to the bus transfer duration of a block of size B supports the conjecture that they both measure the same event: the transfer of a block of size B from the disk drive to the host at bus bandwidth.

Another crucial observation from the tables is that a very large fraction of the I/O activity occurs as good rounds as seen in Figure 2(a). (Recall that a good round is a sequence of D consecutive deltas consisting of a large delta followed by $D-1$ small deltas such that each delta corresponds to a unique disk drive.) For $D=4$, the number of read calls that belong to good rounds as a fraction of the total number of read calls in the experiment (GoodRounds/OutOf) is 81%, 99%, and 99%, respectively, for block sizes of 32, 64 and 96 KB. For $D=7$, the corresponding fractions were 93%, 99%, and 100%.

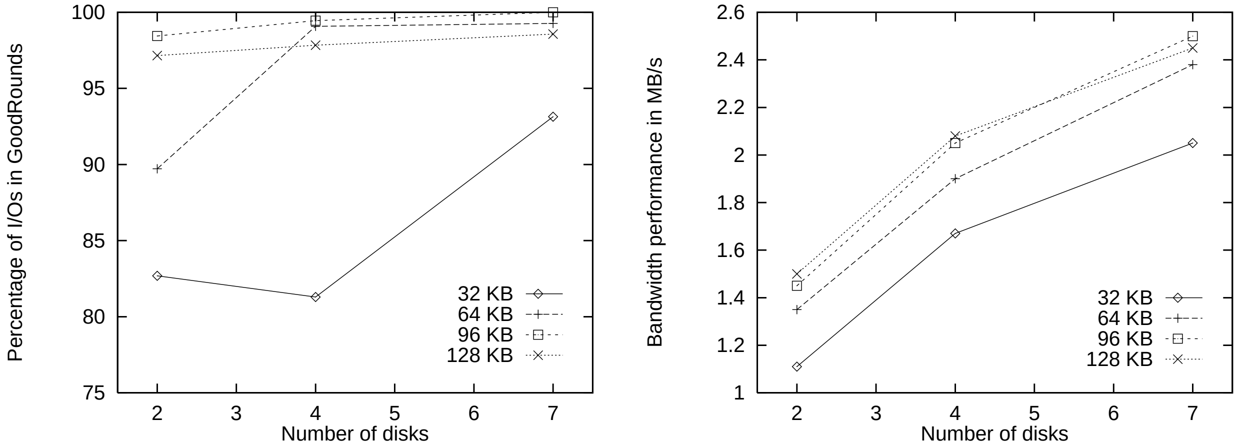
Putting together the observations made in the above two paragraphs, we infer that a large portion of all I/O takes place in rounds consisting of one transfer whose read duration is a large delta, followed by $D-1$ block transfers each occurring at bus bandwidth. One block transfer in each round originates from each disk. We conjecture that the large delta (Large_{ave}) corresponds to the time required for disk latency followed by time for a block transfer; thus making large deltas noticeably larger than small deltas.

With respect to the same experiments for 32, 64 and 96 KB blocks with $D=2$ (the first data row of the three tables), we find relatively higher standard deviations for both large and small deltas. We observe that the fraction of I/O in good rounds remains high: 83 through 98%. However, we find that the small delta values have relatively higher standard deviations compared to the $D=4$ and $D=7$ experiments. Moreover, the small delta values are not quite as close to the bus transfer times as they are for $D=4$ and $D=7$. We observe that the larger the block size, the closer the small delta value is to its bus transfer time. We conclude that round-like behavior is less pronounced for $D=2$ than for $D=4$ and $D=7$. We conjecture that the relatively smaller tendency for round-like behavior for $D=2$ is because the bus is less of a bottleneck compared to the $D=4$ or $D=7$ case; as a result, bus effects have a smaller role to play. But in experiments with $D=2$, we still see that round-like behavior increases with block size. In general, these experiments show that the tendency for round-like behavior increases with D and B.

The effect of a larger block size. Table 4 presents the delta statistics for a block size of 128 KB. The most dramatic difference between these statistics and those for smaller block sizes is that the mean of the small deltas is quite small, and far less than the 37.8 millisecond bus transfer time for blocks of this size. Moreover, in contrast to experiments with smaller block sizes, wherein each large delta seems to consist of the disk latency (for a round) followed by the time to transfer a single block, in the case of 128 KB blocks, the mean of a large delta is close to D times the (37.8 ms) bus transfer time for a block plus a random disk latency and some more. In spite of these differences in delta statistics, the fraction of I/O in good rounds is quite high (97% or more). These experimental results are consistent with the hypothesis of the presence of one disk latency per round, and each disk transferring one block per round back to the host.

Another important and counter-intuitive observation from the bandwidth numbers plotted in Figure 2(b) is that the I/O bandwidth with 128 KB blocks is smaller than that for 96 KB blocks when $D=7$. This surprising drop in bandwidth performance may be related to the difference in delta statistics for 128 KB blocks relative to those for the smaller block sizes, and may be indicative of some additional scheduling problems that need to be solved.

The mean small deltas are 2–5 milliseconds, and since the time taken by context switch events is roughly in the same range, we suspect that they correspond to context switch events. The large delta, we conjecture, corresponds to the transmission of D blocks at sustained bus bandwidth, plus a random disk latency, plus the time to rotationally transfer a single block from the disk surface to the disk's cache. There



(a) Percentage of I/Os forming rounds.

(b) I/O bandwidth in MB/s.

Figure 2: Percentage of good rounds and bandwidth plots. We connected the points, not for the purpose of interpolation, but to direct the eye to the order of the request size.

are two reasons for our conjecture. Firstly, the composition of each round for 128 KB blocks, not in terms of the deltas obtained from measurements, but in terms of *I/O events* such as bus idle time and block transmission events on the bus will likely be the same as the composition of each round for any of the smaller block sizes and the above conjecture is consistent with this claim. The second reason for our conjecture is that the sum of the time taken by each of the events mentioned above lies in the same ball park as the large delta.

Figure 3 serves to illustrate our conjecture that although the round composition with respect to deltas is different for 128 KB blocks relative to smaller blocks, the round composition in terms of I/O events remains the same. In the figure and in the text below, we denote by T_B the time to transfer a block size at sustained bus transfer rate. In both cases, a round consists of a large delta followed by $D-1$ smaller deltas; the difference lies in the events corresponding to large deltas and small deltas as indicated above.

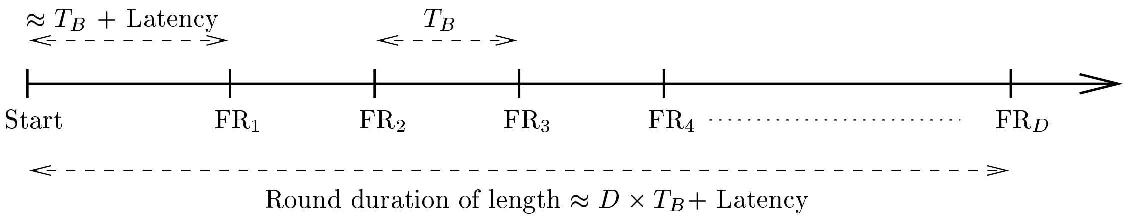
Since the difference in the nature of rounds may be related to the drop in bandwidth performance for 128 KB blocks relative to 96 KB blocks, it is worth considering the issues involved in some detail. The question that needs to be answered is: why is there an FR event (and the generation of a new I/O request) once every T_B time units (with the exception of the first FR event of a round, which requires some additional time) for block sizes smaller than 128 KB, whereas in the case of 128 KB blocks, all D FR events (and the generation of new requests) of a round occur in rapid succession after time $\approx D \times T_B$?

In case of blocks smaller than 128 KB, it appears that whenever a block transfer completes, the process switches in, receives data, instantaneously registers an FR event and then switches out since it makes a new read call; this would account for the pattern of FR events recorded for blocks smaller than 128 KB. According to our conjecture regarding rounds corresponding to the 128 KB blocks, block transfers corresponding to the D processes complete during a large delta, with no FR event occurring until all the block transfers complete. It appears that even when a 128 KB block transfer corresponding to a process completes, that process is not able to immediately register an FR event. We observe that all the processes awaken in close succession after the completion of the last of the D block transfers; this accounts for the pattern of FR events recorded for 128 KB blocks. This behavior is consistent with our observation in Section 4 that the host SCSI controller will not seize the bus to send a new request if some disk wishes to use the bus to transfer data to the host—it appears that the 128 KB transfer size is large enough that none of the user processes (making the read calls) gets a CPU slice until all D transfers complete.

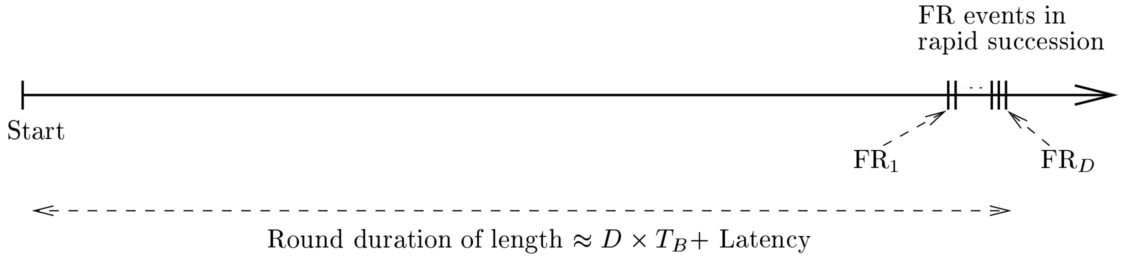
The small deltas in Table 4 exhibit large standard deviations. The underlying data show that the variance is caused by a few outliers—relatively large values for the small delta. This indicates that the above scenario sometimes does not occur, in which case the small delta includes portions of the bus transfer time.

4.3 Random parallel I/O with 0 fence value

In this section, we report on our experiments with the fence set to its minimum value. When the fence is zero, each disk drive contends for the bus as soon as the first sector of data has been transferred from the disk surface to the disk's cache. A drive that succeeds in grabbing the bus before its entire block has been loaded into its disk cache will transfer at the rotational media bandwidth $\mathsf{bandwidth}_{rot}$ the trailing portion of that block. Fortunately, during this time, the other drives can often complete the loading of their block into their disk cache, so that this extra overhead is not incurred for each block. In addition, the transfer of a single requested block may get interspersed with the transfer of other blocks, since the drive may disconnect from the bus mid-transfer, in response to a track or cylinder crossing. Disconnects for track or cylinder crossings rarely arise when the fence is set to its maximum value (as in the experiments of



(a) A typical round for block sizes smaller than 128 KB.



(b) A typical round for block size 128 KB.

Figure 3: Difference in the nature of rounds for (a) smaller block sizes and (b) 128 KB blocks.

the previous section), since the drive encounters such crossings prior to contending for the bus.

As a result, deltas encountered when the fence is 0 are not as orderly as compared to the case when the fence is 255/256. Nevertheless, our experiments with the fence set to 0 demonstrate the consistent round-like pattern of a large delta followed by a sequence of $D-1$ small deltas, one for each disk. Moreover, the mean small delta values are close to the bus transfer times, although not as close as in the experiments of the previous section. When we run the delta classification algorithm of the previous section, the standard deviations are high (relative to the 255/256 case), and we suspect that this is due to the complications mentioned above.

4.4 Sequential parallel I/O

We also perform experiments where the each processes requests a sequence of consecutive blocks that can be read sequentially from disk. We first present the results of the analysis for block sizes of 16, 32 and 64 KB, before considering the block size of 128 KB.

In the delta patterns for sequential accesses, we expect that the read-ahead mechanism of the Wren-7 disks will ensure that we do not observe any seek or rotational latency overheads (i.e., no large deltas). Our delta analysis for 16, 32 and 64 KB blocks is consistent with this. The deltas have small standard deviations. More important, for a given block size, the mean values of the deltas are extremely close to the mean values of the small deltas from the delta analysis of the random access patterns, thus further bolstering our conjecture that the deltas correspond to the transfer of entire blocks on the bus at sustained bus bandwidths. Table 5 presents the means and standard deviations of the delta analysis.

Behavior of the deltas for 128 KB blocks is different, although it can be explained consistently with our hypothesis. Below, we mention one experiment. Let us consider the sequential access experiment in which $D = 7$ and $B = 128$ KB. We observe that the delta values cluster around three distinct values: 120 milliseconds, 80 milliseconds, and 3 milliseconds. There is a consistent repeating pattern of these delta values. When $D = 7$, a round consists of one large delta value (120 milliseconds), two medium delta values (80 milliseconds), and four small delta values (3 milliseconds). We suspect that the reasons for the difference in behavior of the deltas for 128 KB blocks relative to blocks of other sizes are similar to the ones discussed in Section 4.2 of random requests for 128 KB blocks with the maximum fence value.

5 Concluding remarks

In this paper, we describe experiments that measure the performance of multiple SCSI disks that share a bus. We study intensive random-access workloads in which each disk is driven by a process that issues random read requests as quickly as possible. We do not observe the expected independence of I/Os, with faster service rates for the disks that have higher SCSI priority. Instead, the trace of I/O completion times reveals a convoy behavior that we term *rounds*: all disks receive a read request, then the SCSI bus is idle while waiting for the shortest access time for this set of requests, and then the SCSI bus remains busy while all the requests are satisfied, and only then does the SCSI host adapter send new requests to the disks. We observe this behavior in several different computer systems from different manufacturers. (By comparing in-kernel measurements with timings collected by a SCSI bus analyzer, we have been able to attribute this behavior to a policy that appears to be implemented in every SCSI host adapter that we have tested: although the host adapter has the highest SCSI priority, if any disk wishes to use the bus to send data to the host, the host will not seize the bus to send a new request to an idle disk.)

The measurements and observations reported in this paper are the foundation for additional work (reported in [BSG+99]) that (1) confirms the existence of the rounds phenomenon for Seagate Barracuda and Cheetah disks on PCs, Suns, and DEC Alphas, (2) develops a detailed analytic performance model for multiple disks sharing a SCSI bus, and (3) describes ways to reduce the bus idle time caused by the rounds phenomenon.

References

[BGV97] Rakesh D. Barve, Edward F. Grove, and Jeffrey S. Vitter. Simple randomized mergesort on parallel disks. *Parallel Computing*, 23(4):601–631, June 1997.

[BSG+98] Rakesh Barve, Elizabeth Shriver, Phillip B. Gibbons, Bruce K. Hillyer, Yossi Matias, and Jeffrey Scott Vitter. Modeling and optimizing I/O throughput of multiple disks on a bus. In *Joint International Conference on Measurement and Modeling of Computer Systems (Sigmetrics '98/Performance '98)*, pages 264–266, Madison, WI, June 1998. Extended abstract. Available at `http://www.bell-labs.com/~shriver/`.

[BSG+99] Rakesh Barve, Elizabeth Shriver, Phillip B. Gibbons, Bruce K. Hillyer, Yossi Matias, and Jeffrey Scott Vitter. Modeling and optimizing I/O throughput of multiple disks on a bus. In *Proceedings of the 1999 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, Atlanta, GA, May 1999. Available at `http://www.bell-labs.com/~shriver/`.

[CH97] Thomas H. Cormen and Melissa Hirschl. Early experiences in evaluating the parallel disk model with the ViC* implementation. *Parallel Computing*, 23(4):571–600, June 1997.

[HP96] John L. Hennessy and David A. Patterson. *Computer architecture: a quantitative approach*. Morgan Kaufmann Publishers, Incorporated, San Francisco, CA, 1996. 2nd edition.

[MNO+96] Cliff Martin, P.S. Narayan, Banu Ozden, Rajeev Rastogi, and Avi Silberschatz. The Fellini multimedia storage system. In Soon M. Chung, editor, *Multimedia Information Storage and Management*, chapter 5. Kluwer Academic Publishers, August 1996.

[RW94] Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, March 1994.

[Shr97] Elizabeth Shriver. *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/`.

[VS94] Jeffrey S. Vitter and Elizabeth A. M. Shriver. Algorithms for parallel memory I: two-level memories. *Algorithmica*, 12(2/3):110–47, August/September 1994.

[WGPW95] Bruce L. Worthington, Gregory R. Ganger, Yale N. Patt, and John Wilkes. On-line extraction of SCSI disk drive parameters. In *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 146–156, Ottawa, Canada, May 1995.

[Wil95] John Wilkes. The Pantheon storage-system simulator. Technical Report HPL–SSP–95–14, Storage Systems Program, Hewlett-Packard Laboratories, Palo Alto, CA, December 1995.

Table 1: Delta statistics for $B = 32$ KB.

D	Threshold	GoodRounds	OutOf	Large$_{ave}$	Large$_{std}$	Small$_{ave}$	Small$_{std}$	#Small> 20
2	25	1462	1768	40	9.22	14	4.67	320/1709
4	25	1082	1331	42	6.85	10	2.32	74/4080
7	25	869	933	42	4.37	9	1.47	0/5608

Table 2: Delta statistics for $B = 64$ KB, where $X = 35$ for $D = 2$ and $X = 25$ for $D = 4, 7$.

D	Threshold	GoodRounds	OutOf	Large$_{ave}$	Large$_{std}$	Small$_{ave}$	Small$_{std}$	#Small> X
2	45	969	1080	70	10.04	23	6.03	103/1131
4	45	754	761	70	5.40	19	2.33	5/2286
7	45	541	545	68	4.71	18	2.68	2/3270

Table 3: Delta statistics for $B = 96$ KB.

D	Threshold	GoodRounds	OutOf	Large$_{ave}$	Large$_{std}$	Small$_{ave}$	Small$_{std}$	#Small> 35
2	50	759	771	97	12.16	30	3.54	29/763
4	50	542	545	96	6.88	28	3.42	1/1635
7	50	380	380	94	5.24	27	3.15	0/2280

Table 4: Delta statistics for $B = 128$ KB.

D	Threshold	GoodRounds	OutOf	Large$_{ave}$	Large$_{std}$	Small$_{ave}$	Small$_{std}$	#Small> 20
2	75	582	599	160	14.40	5	11.98	33/600
4	75	407	416	228	13.93	2	3.96	5/1245
7	100	276	280	333	16.46	3	4.42	15/1681

Table 5: Mean and standard deviation of deltas for sequential accesses.

B	$D = 2$			$D = 4$			$D = 7$		
(KB)	# of I/Os	mean	std dev.	# of I/Os	mean	std dev.	# of I/Os	mean	std dev.
16	2506	5	1.26	2801	4	1.29	2941	4	0.57
32	1386	10	1.80	1609	8	1.21	1606	8	1.21
64	732	20	2.56	792	18	2.37	795	18	2.35