Diskbench: User-level Disk Feature Extraction Tool

Zoran Dimitrijević[1,2], Raju Rangaswami[1], David Watson[2], and Anurag Acharya[2]

Univ. of California, Santa Barbara[1] and Google[2]
{zoran,raju}@cs.ucsb.edu, {davidw,acha}@google.com

Despite the increases in disk capacity and decreases in mechanical delays in recent years, the performance gap between magnetic disks and CPU continues to increase. To improve disk performance, operating systems and file systems must have detailed low-level information (e.g., zoning, bad-sector positions, and cache size) and high-level information (e.g. expected read and write performance for different access pattern) about the disks that they use. In this paper, we present Diskbench, our tool for extracting such information. Diskbench uses both interrogative and empirical methods for extracting various disk features. We present our extraction methods and results for several testbeds. From our empirical study, we conclude that intelligent data placement and access methods can be devised to improve disk performance, by exploiting low-level disk knowledge. Diskbench has benefitted our video storage research in the implementation of *Semi-preemptible IO* and guaranteed real-time scheduling.

Key Words: Disk features, disk modeling, video servers, disk QoS, disk I/O, file placement.

Thanks: This research was supported by SONY/UC DiMI and the NSF CISE Infrastructure grant EIA-0080134. Scsibech was written in February 2000. Extensions and additional features were added in 2001 for Xtream [1] and in 2002 for Semi-preemptible IO [2], [3].

I. Introduction

The performance gap between hard drives and CPU-memory subsystems is steadily increasing. In order to bridge this gap, the operating system must use intelligent disk management strategies [4], [5], [6], [7], [8]. Most strategies assume that detailed disk parameters, such as zoning, bad-sector locations, and disk latency, can be obtained from the disk manufacturers. However, the information they provide can be imprecise and static. For instance, disk vendors usually give out only the maximum, minimum, and average data transfer rates and seek time. In addition, some dynamic information such as the locations of bad sectors cannot be known prior to actual use. As a consequence, the effectiveness of most traditional disk management strategies can be compromised.

For optimal disk performance, it is necessary to tune disk accesses to the requirements of the application by extracting the necessary disk features. For example, a multimedia streaming server must predict the hard disk performance to maintain real-time streaming requirements without under-utilizing disks. However, disk abstractions (e.g., SCSI and IDE interface) hide low-level device characteristics from the operating system and virtualize the access to the device in the form of logical blocks. Such device abstractions make the task of tuning disk operation to match application requirements (and thus improving IO efficiency) difficult. In this chapter we present Diskbench, a tool for the extraction of disk features. Diskbench consists of two applications, Scsibench and Idextract. Scsibench [9] runs in user space on Linux systems and accesses SCSI disks through the SCSI generic interface provided by Linux. It uses interrogative and empirical methods for feature extraction similarly to the previous work in disk profiling [10], [11], [12], [13]. Idextract uses Linux raw disk access and empirical methods to extract features from any disk-like device (the approach used by Patterson et al. [12]). Scsibench is open source and available for download [14].

Using Diskbench, we can obtain many low-level disk features including 1) rotational time, 2) seek curve, 3) track and cylinder skew times, 4) caching and prefetching techniques, and 5) logical-to-physical block mappings. Diskbench also exctract several high-level disk features useful for real-time disk schedulers. In this paper we present two important high-level features: optimal chunk size and admission control curves. In addition, we show that the access time (including seek time and rotational delay) between two disk accesses can be predicted with high accuracy. Scsibench also supports the execution of disk traces by providing an intuitive interface for executing primitive SCSI disk commands (e.g., read, write, seek, enable/disable cache, etc.) with an accurate timing mechanism. Using knowledge about disk features and the trace support, system or application programmers can obtain the precise distribution of times spent by the disk performing various operations. Bottlenecks can thus be identified, and disk scheduling can be adjusted accordingly to utilize the disk more efficiently. Two such systems which currently benefit from using Diskbench are XTREAM and Semi-preemptible IO.

The XTREAM multimedia system [1] provides real-time video streaming capability to multiple clients simultaneously. For video playback, the disk management must guarantee that all IOs meet their real-time constraints. If the system does not have information about low-level disk features, it must assume the worst-case IO time or use statistical methods. These pessimistic and statistical estimates of disk drive performance lead to sub-optimal performance of the entire system. In contrast, XTREAM uses Diskbench to obtain the required disk features for making accurate admission control decisions.

Semi-preemptible IO [3], [2] is an abstraction for disk IO, which provides preemptible disk access with little loss in disk throughput. Preemptible IO relies on an accurate disk-access prediction. The implementation of Semi-preemptible IO was made feasible due to Scsibench, which extracts essential disk information for accurate disk-performance modeling.

II. Related Work

Several previous studies have focused on the problem of disk feature extraction. Seminal work of Worthington, Ganger, et al. [13], [10] extract disk features in order to model disk drives accurately. Both studies rely on interrogative SCSI commands to extract the LBA-to-PBA mapping from the disk. Patterson et al. [12] present several methods for empirical feature extraction, including an approximate mapping extraction method. In [11], the authors propose methods for obtaining detailed temporal characteristics of disk drives, including methods for predicting rotational delay. Diskbench was first implemented in early 2000, based on the ideas from Worthington et al. [10]. The main difference at the time was our empirical extraction of disk mappings, which showed possibility of accurate mapping using approaches similarly to Patterson et al. [12]. Our main contribution (apart from providing Scsibench to community as open source) is investigation of several high-level disk features and using those features to provide semi-preemptible IO and implement real-time disk admission-control algorithms.

Scheduling algorithms in [4], [5], [6], [7] assume the ability to predict the rotational delay between successive requests to the disk. In [15], [8] authors implemented their scheduler outside of disk firmware relying on their disk profiler [13]. This study is similar to work by Schindler et al. [13] and includes both interrogative and empirical methods for the accurate extraction of disk mapping information. The empirical methods can be used for disks that do not support interrogative commands in order to extract disk mapping information. Using empirical extraction, we can obtain accurate disk mapping information, including precise positions of track and cylinder boundaries. However, empirical methods are slower than interrogative ones. To predict rotational delay between disk IOs, Diskbench uses an approach that is similar to the approach presented in [11]. Methods in [11] are continuously keeping track of the disk head position. We concentrate on predicting rotational distance between two LBAs and do not require the head position knowledge, as explained in Section IV-B.2 and Section V-H. Such prediction capability can be used for scheduling requests in real systems, where requests are known to arrive in a bursty fashion [16]. Diskbench can extract all the disk features that are required in order to predict disk access times with high accuracy (we have used the prediction in [3], [2]).

Scsibench is open source and available for download [14]. At this time, we are not aware of any other disk feature extraction tool which is currently available for download as open source, and which runs on a widely available operating system without any kernel modifications.

III. Disk Architecture

Before we get into the details of disk features that are of interest when designing high performance systems, we provide a brief overview of the disk architecture. The main components of a typical disk drive are:

- One or more *disk platters* rotating in lockstep fashion on a shared *spindle*,
- A set of *read/write heads* residing on a shared arm moved by an *actuator*,
- *Disk logic*, including the disk controller, and
- *Cache/buffer memory* with embedded replacement and scheduling algorithms.

The data on the disk drive is logically organized into disk *blocks* (the minimum unit of disk access). Typically, a block corresponds to one disk *sector*. The set of sectors that are on the same magnetic surface and at the same distance from the central spindle form a *track*. The set of tracks at the same distance from the spindle form a *cylinder*. Meta-data such as error detection and correction data are stored in between regular sectors. Sectors can be used to store the data for a logical block, to reserve space for future bad sector re-mappings (spare sectors), or to store disk meta-data. They can also be marked as "bad" if they are located on the damaged magnetic surface.

The *storage density* (amount of data that can be stored per square inch) is constant for the magnetic surfaces (media) used in disks today. Since the outer tracks are longer, they can store more data than the inner ones. Hence, modern disks do not have a constant number of sectors per track. Disks divide cylinders into multiple *disk zones*, each zone having a constant number of sectors per track (and hence having its own performance characteristics).

The *rotational speed* of the disk is constant (with small random variations). Since the track size varies from zone to zone, each disk zone has a different *raw bandwidth* (data transfer rate from the disk magnetic media to the internal disk logic). The outer zones have a significantly larger raw disk bandwidth than the inner ones.

When the disk head switches from one track to the next, some time is spent in positioning the disk head to the center of the next track. If the two adjacent tracks are on the same cylinder, this time is referred to as the *track switch time*. If the tracks are on different cylinders, then it is referred to as *cylinder switch time*. In order to optimize the disk for sequential access, disk sectors are organized so that the starting sectors on two adjacent tracks are skewed. This skew compensates for the track or cylinder switch time. It is referred to as *track skew* and *cylinder skew* for track and cylinder switches respectively.

The *seek time* is the time that the disk arm needs in order to move from its current position to the destination cylinder. In the first stage of the seek operation, the arm accelerates at a constant rate. This is followed by a period of constant maximum velocity. In the next stage, the arm slows down with constant deceleration. The final stage of the seek is the settle time, which is needed to position the disk head exactly at the center of the destination track. Since the disk seek mainly depends on the characteristics of the disk arm and its actuator, the seek time curve does not depend on the starting and destination cylinders. It depends only on the seek distance (in cylinders).

The disk magnetic surfaces contain defects because the process of making perfect surfaces would be too expensive. Hence, disk low-level format marks bad sectors and skips them during logical block numbering. Additionally, some disk sectors are reserved as spare, to enable the disk to re-map bad sectors that occur during its lifetime. The algorithm for spare sector allocation differs from disk to disk. In order to accurately model the disk for intelligent data placement, scheduling, or even simple seek curve extraction, a system needs detailed mapping be-

tween the physical sectors and the logical blocks. In addition to mapping, a system must be able to query the disk about re-mapped blocks. Re-mapping occurs when a disk detects a new bad sector.

The *disk cache* is divided into a number of *cache segments*. Each cache segment can either be allocated to a single sequential access stream or can be further split into blocks for independent allocation. In this paper, the cache parameters of interest are the segment size, the number of cache segments, the segment replacement policy, prefetching algorithms, and the write buffer organization. Prefetching is used to improve the performance of sequential reads. The write buffer is used to delay the actual writing of data to the disk media and enable the disk to re-schedule writes to optimize its throughput.

IV. Disk Profiling

In this section, we present methods for extracting certain disk features. We use a combination of interrogative and empirical methods. Interrogative methods use the inquiry SCSI command [17], [18] to get required information from the disk firmware. Empirical methods measure completion times for various disk access patterns, and extract disk features based on timing measurements.

A. Low-level Disk Features

We now present the methods Diskbench uses to extract low-level disk features. In some of our extraction methods we assume the ability to force access to the disk media for read or write requests (hence, avoiding the disk caching and buffering). Most modern disks allow turning off the write buffer. In the case of SCSI disks, this can be done by turning off the disk buffers, or by setting the "force media access bit" in a SCSI command [18].

1) OS Delay Variations: In order to estimate variations in operating system delay for IO requests, we use the following method. First, we turn off all disk caching and disk buffering. Then, we read the same block on disk in successive disk rotations, as in the empirical method for extracting T_{rot}. We measure completion times for each read request. For current disks, variations in the rotational period T_{rot} are negligible. Because of this, variations in $T_i - T_{i-1}$ from Equation 2 give us the distribution of $\Delta T_{OS\_delay_{i,i-1}} = T_{OS\_delay_i} - T_{OS\_delay_{i-1}}$. Thus, by measuring variations in $T_{i+1} - T_i$ from Equation 2, we can estimate variations in the operating system delay.

2) Rotational Time: Since modern disks have a constant rotational speed, if the interrogative SCSI command for obtaining rotational period (T_{rot}) is supported by the disk, it will return the correct value. In the absence of the interrogative command, we can also use the following empirical method described in Worthington et al. [10] to obtain T_{rot}: First, we ensure that read (or write) commands access the disk media. Next, we perform n successive disk accesses to the same block, and measure the access completion times. The absolute completion time for each disk access is

$$T_i = T_{end\_reading} + T_{transfer} + T_{OS\_delay_i}. \quad (1)$$

$T_{end\_reading}$ is the absolute time immediately after the disk reads the block from the disk media. $T_{transfer}$ is the transfer time needed to transfer data over the IO bus. $T_{OS\_delay_i}$ is the time between the moment when the OS receives data over the IO bus, and the moment when the data is transfered to the user level Diskbench process. Since the disk need to wait for one full disk rotation for each successive disk block access, we can write the following equations:

$$T_{i+1} - T_i = T_{rot} + (T_{OS\_delay_{i+1}} - T_{OS\_delay_i}); \quad (2)$$

$$T_{n+1} - T_1 = n \times T_{rot} + (T_{OS\_delay_{n+1}} - T_{OS\_delay_1}). \quad (3)$$

The rotational period for current disks is much greater than OS delays and other IO overheads (not including the seek and rotational times). Thus, we can measure the rotational period as

$$T_{rot\_measured} = T_{rot} + \frac{\Delta T_{OS\_delay_{n+1,1}}}{n} = \frac{T_{n+1} - T_1}{n}. \quad (4)$$

For large n, the error term ($\frac{\Delta T_{OS\_delay_{n+1,1}}}{n}$) is negligible.

3) Mapping from Logical to Physical Block Address: Most current SCSI disks implement SCSI commands for address translation (Send/Receive Diagnostic Command [18]) which can be used to extract disk mapping. However, in the case of older SCSI disks, or for disks where address translation commands are not supported (e.g. ATA disks), empirical methods are necessary.

Interrogative Mapping: For interrogative mapping, we use an algorithm based on the approach described in Worthington et al. [10]. Using the interrogative method, a single address translation typically requires less than one millisecond. But, since the number of logical blocks is large, it is inefficient to map each logical block. Fortunately, modern disks are optimized for sequential access of logical blocks. Additionally, most disks use the skipping method to skip bad sectors (instead of re-mapping them) during the low-level format. Due to this, logical blocks on a track are generally placed sequentially. Thus, we can extract highly accurate mapping information by translating just one address per track, except when we detect anomalies (tracks with bad blocks).

Since the number of re-mapped blocks is small compared to total number of blocks, we propose using two data structures to store mapping information obtained using the interrogative method. In the first data structure, we store the mapping in the form the disk had immediately after the low-level format. Since there are no re-mapped blocks, we simply store the starting logical block number and the track size (in blocks) for each track. The second data structure is used to store information about the re-mapped blocks. This way, we only need to update the second structure periodically, using the inquiry SCSI command.

Figure 1 presents a simplified algorithm for the interrogative extraction used in Diskbench. Using the SCSI command for physical-to-logical address translation, we extract the LBA for the first sector (sector zero) of each track. If sector zero is marked as bad, we continue performing address translation for subsequent sectors until we obtain a valid logical block number. When we come across a cylinder in which the number of sectors that lack a valid LBA (bad or spare blocks) is above a

Procedure: *Interrogative Mapping*

- **Variables:**
  1) cyl_num : Total number of cylinders
  2) $track\_per\_cyl$: Number of tracks per cylinder
  3) i : Cylinder number
  4) j : Track number
  5) $cyl\_info[i]$: data structure for cylinder i info
  6) $cyl\_info[i].logstart[j]$: Starting LBA for track j on cylinder i
  7) $cyl\_info[i].logsize[j]$: Track j's size in blocks
- **Execution:**
  1) for i = 0 to cyl_num do
  2) for j = 0 to $track\_per\_cyl$ do
  3) for k = 0 to $K_{threshold}$ do
  4) $cyl\_info[i].logstart[j]$ = phys_to_log(i,j,k)
  5) if valid($cyl\_info[i].logstart[j]$) then break
  6) if not_valid($cyl\_info[i].logstart[j]$) then
  7) mark track bad.
  8) sort by $cyl\_info[i].logstart[j]$
  9) calculate $cyl\_info[i].logsize[j]$

Fig. 1. Interrogative method for disk mapping.

fixed threshold ($K_{threshold}$, we mark that cylinder as logically bad. We do not use these cylinders in our seek curve extraction method. Tracks which have a substantial number of blocks without a valid LBA are usually the ones containing mostly spare sectors.

Empirical Mapping: Empirical methods for the extraction of mapping information are needed for disks that do not support address translation. Our empirical method is essentially similar to approach presented in Patterson et al. [12]. Our improvements include using the first derivative of access times for mapping and heuristics to prune the mapping errors near the track boundaries. The empirical extraction method used in Diskbench follows. In the first step, we measure the time delay in reading a pair of blocks from the disk. We repeat this measurement for a number of block pairs, always keeping the position fixed for the first block in the pair. In successive experiments, we linearly increase the position of the second block in a pair. Using this method, our tool extracts accurate positions of track and cylinder boundaries. We now emphasize this method in detail. Time $T(i)$ defined in Equation 5 is the completion time measured at the moment when the user process receives data for logical block address i. (The variables on the right side of Equation 5 are defined in Section IV-A.2.)

$$T(i) = T_{end\_reading} + T_{transfer} + T_{OS\_delay}. \quad (5)$$

The access time ($T_a(x,0) = T_{end\_reading}(x) - T_{end\_reading}(0)$) is the time needed to access block x after accessing block 0. It includes both seek time and rotational delay, but does not include transfer time and OS delay. Equation 7 presents the first derivative of $\Delta T(x,0)$ defined in Equation 6.

$$\Delta T(x,0) = T(x) - T(0);$$

$$\Delta T(x-1,0) = T_a(x-1,0) + (T_{OS\_delay_{x-1}} - T_{OS\_delay_0});$$

$$\Delta T(x,0) = T_a(x,0) + (T_{OS\_delay_x} - T_{OS\_delay_{0'}}). \quad (6)$$

$$\Delta = \Delta T(x,0) - \Delta T(x-1,0);$$

$$\Delta = T_a(x,0) - T_a(x-1,0) + (\Delta T_{OS\_delay_{x,0'}} - \Delta T_{OS\_delay_{x-1,0}}). \quad (7)$$

When the OS delay variations are small, and both blocks $x-1$ and x are on the same track, Δ is small, and proportional to $T_{rot}/track_{size}$. When the access can be performed without any additional rotational delay after seek, or with delay of a full T_{rot}, Δ is proportional to $-T_{rot} + T_{rot}/track_{size}$. The sign of Δ depends on T_{OS_delay}. If x and $x-1$ are on different tracks, or cylinders, Δ is proportional to skew time T_{skew}, where T_{skew} is the track or cylinder skew time. Since the cylinder skew is usually larger than the track skew, we can use the positions of the skew times in Δ in order to find out accurate track and cylinder boundaries. We define normalized first derivative in Equation 8. This way we eliminate the $-T_{rot}$ factor from Equation 7, which helps us to automatically extract accurate disk mapping.

$$norm(\Delta) = (\Delta T(x,0') - \Delta T(x-1,0)) \bmod T_{rot}. \quad (8)$$

The $norm(\Delta)$ is useful for automatic extraction only if the OS delay is small compared to the skew times. Since the OS delay is a random variable, we perform several measurements whenever $|\Delta|$ is greater than a specific threshold ($0.02 \times T_{rot}$) and stop the measurement when the difference between two consecutive Δ's is less than a specific threshold (5%).

4) Seek Curves: Seek time is the time that the disk head requires to move from the current to the destination cylinder. We implement two methods for seek curve extraction. The first method uses the SCSI seek command to move (seek) to a destination cylinder. The second one measures the minimum time delay between reading a single block on the source cylinder and reading a single block on the destination cylinder to obtain the seek time. In order to find the minimum time, we can measure the time between reading a fixed block on the source cylinder, and reading all blocks on one track in the destination cylinder. The seek time is the minimum of the measured times. Since LBAs increase linearly on a track, we also implement an efficient binary-search method in order to find the minimum access time or seek time. $T_{seek}(x,y)$ returns seek time (in μs) between logical blocks x and y. This function is symmetrical, i.e., $T_{seek}(x,y) = T_{seek}(y,x)$. This seek time also includes the disk head settling time.

5) Disk Buffer/Cache Parameters: Most disk drives are equipped with a read cache. Read caches improve disk performance by allowing for data prefetching. We now present methods to extract the cache segment size, the number of segments, and the segment replacement policy.

Read Cache Segment Size: The method for extracting the cache segment size consists of three steps. First we read a few sequential disk blocks from a specific disk location. Next we wait for a long enough period of time (several disk rotations) to allow the disk to fill up the cache segment with prefetched data. Finally we read consecutive disk blocks occurring immediately after the first read and measure the completion time. If the block is in the cache, the completion time includes only a

block transfer time from the cache to the OS through the IO bus and a random T_{OS_delay}. If the block is not in the cache, the completion time also includes seek time and rotational delay, as well as data transfer time. The seek time and rotational delay are the dominant factors in the IO completion time if present. We can thus detect the size of a cache segment by detecting the moment when the completion time includes mechanical seek and rotational delays.

Number of Cache Segments: In order to extract the number of disk cache segments, we need to be able to clear the cache. We clear the cache by performing a large number of random sequential reads for different logical blocks, which effectively clear the cache by polluting it. In this extraction method we linearly increase the number of sequential streams accessing the disk. In each iteration we do the following: For each stream we read a few blocks (from locations which are not used in cache pollution). Then we wait a sufficient amount of time so that the disk can fill the cache segment with prefetched data. After this step, we assume that the disk cache has allocated one cache segment for each stream.

We perform read requests for all streams and measure completion times. If the completion times are smaller than a specific threshold, we assume that all blocks were in the cache, and that the number of cache segments is greater than or equal to the number of streams in this iteration. When we detect that one read request requires an amount of time that exceeds the threshold, we repeat the entire experiment to confirm that the excessive time is not caused by a large random OS delay. On confirmation, we deduce that the number of cache segments is equal to the number of streams in the previous iteration. By changing the access pattern in the previous experiment, and noting which streams are not serviced from the cache, we can deduce the policy used for the cache segment replacement.

Write Buffer Parameters: Most disk drives are equipped with a write buffer to improve the disk write performance. If the disk has sufficient space in the write buffer, then the write command will be completed as soon as data is transferred to the disk's write buffer. The disk writes this data to the disk surface in an optimal manner at some later time.

Figure 2 presents an empirical method for extracting the write buffer size. Between iterations, we allow the disk to purge the contents of the write buffer to the disk media. Before issuing the write request, we also seek to a cylinder far away from the write request's destination. When the write request size is smaller than the write buffer, we expect that the write completion times will increase linearly, proportional to the throughput of the IO bus. When the write request size is greater than the write buffer, the completion time will incur seek and rotational delays. We can detect this using simple heuristics.

B. High-level Disk Features

Most real-time schedulers rely on simple disks models to reduce problem complexity. In this section, we present several high-level disk features that are used for data-placement algorithms, rotationally-aware schedulers [4], [5], [6], preemptible schedulers [2], and admission control methods [1].

Procedure: *Write Buffer Size*

- **Variables:**
  1) max_size : Maximal estimated write buffer size
  2) $start$: Starting LBA for write request
  3) far : LBA for a block with large $T_{seek}(start, far)$
  4) i : Write request size iteration
  5) T_1, T_2, T_{prev}: Time registers
- **Execution:**
  1) T_{prev} = 0
  2) for i = 1 to max_size do
  3) disk_seek(far)
  4) wait($20 \times T_{rot}$)
  5) T_1 = get_time()
  6) disk_write($start$, i)
  7) T_2 = get_time()
  8) if $T_2 - T_1 - T_{prev} > \frac{T_{rot}}{10}$ then
  9) return $i - 1$
  10) $T_{prev} = T_2 - T_1$

Fig. 2. Empirical method for extracting write buffer size.

1) Disk Zones: Using the extracted disk mapping, Diskbench implements methods for the extraction of zoning information, including precise zone boundaries, the track and cylinder skew factors for each zone, the track size in logical blocks, and the sequential throughput of each zone. The algorithm used to extract zoning information scans the cylinders from the logical beginning to the logical end based on the disk mapping table. Due to the presence of bad and spare sectors, some tracks in a zone may have a smaller number of blocks than the others. Since we store only the track size (in logical blocks) for each track, we may detect a new zone incorrectly. In order to minimize the number of false positives, we use the following heuristics. First, we ignore cylinders with a large number of spare sectors. Second, during the cylinder scan, we detect a new zone only if the maximum track size in the current cylinder differs from the track size of the current zone by more than two blocks. Third, we detect a new zone only when the size of the new zone (in cylinders) is above a specific threshold.

2) Rotational Delay: In order to optimize disk scheduling, the OS may use both seek and rotational delay characteristics of a disk [5], [7], [6], [19]. We can predict rotational distance between two LBAs using the following:

- mapping information extracted in Section IV-A.3, and
- skew factors for the beginning of each track, relative to a chosen rotational reference point.

We choose the disk block with LBA zero as the reference point. Let c_i be the track's cylinder number, t_j the track's position in a cylinder, and $track_{size}(c_i, t_j)$ the track's size in logical blocks. Let $LBA_{start}(c_i, t_j)$ be the track's starting logical block number, and T(LBA) the time after access to a specific LBA is completed. The skew factor of a track is defined as

$$s_{c_i,t_j} = [T(LBA_{start}(c_i, t_j)) - T(LBA_0)] \bmod T_{rot}. \quad (9)$$

If the number of spare and bad sectors is small, we can accurately predict the rotational distance between two LBAs (x and

y) using the following equations:

$$X = T_{rot} \times \frac{[x - LBA_{start}(c_x, t_x)] \bmod track_{size}(x)}{track_{size}(x)} + s_{c_x,t_x}; \quad (10)$$

$$Y = T_{rot} \times \frac{[y - LBA_{start}(c_y, t_y)] \bmod track_{size}(y)}{track_{size}(y)} + s_{c_y,t_y}; \quad (11)$$

$$T_{rot\_del}(y, x) = (Y - X) \bmod T_{rot}. \quad (12)$$

Using the seek time $T_{seek}(y, x)$ defined in Section IV-A.4 and the rotational delay prediction from Equation 12, we can predict the access time to a disk block y after access to a block x, $T_a(y, x)$ as

$$T_a(y, x) = T_{rot\_del}(y, x) + T_{rot} \times \left\lceil \frac{T_{seek}(y, x) - T_{rot\_del}(y, x)}{T_{rot}} \right\rceil. \quad (13)$$

3) Sequential Throughput and Chunking: The maximum IO size in current schedulers in commodity operating systems is bounded to reasonable small values (approximately between 128 and 256 kB). Since large files are usually placed sequentially, the sequential access is divided into "chunks" [20], [2]. In this section, we present a method to extract optimal chunk size for the sequential disk access. Figure 3 illustrates the effect of the chunk size on the disk throughput using a mock disk. The optimal chunk size lies between a and b. For chunk sizes smaller than a, due to the overhead associated with issuing a disk command, the IO bus is a bottleneck. Point b in Figure 3 denotes the point beyond which the performance of the cache may be sub-optimal. Points a and b in Figure 3 can both be extracted using Diskbench.

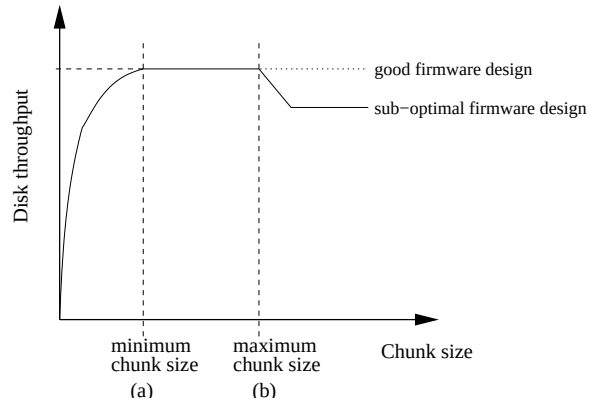


Fig. 3. Effect of chunk size on disk throughput.

4) Admission Control Curves: Equation 14 offers a simple model for disk utilization (U) which depends on the number of IO requests in one cycle (N). The transfer time ($T_{transfer}$) is the total time that the disk spends in data transfer from disk media in a time cycle. The access time (T_{access}) is the average access penalty for each IO request, which includes both the disk seek time and rotational delay.

$$U = \frac{T_{transfer}}{N \times T_{access} + T_{transfer}} \quad (14)$$

Since the disk utilization U depends only on the number of requests and the total amount of data transfered in a time cycle, it can be expressed as a function of just one parameter: the average IO request size (S_{avg}).

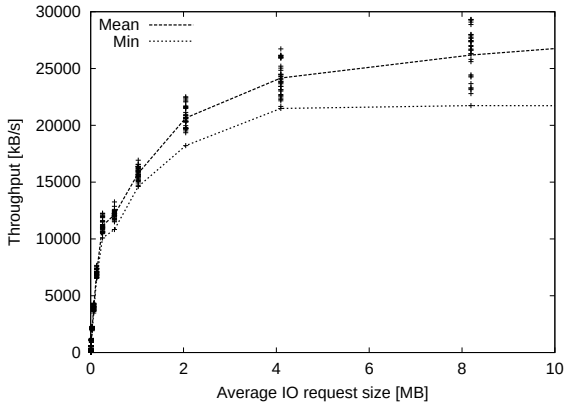


Fig. 4. Disk throughput vs. average IO size.

We use our disk profiler tool to measure the disk-throughput utilization. The profiler performs sequential reads of the same size from random positions on the disk. Figure **??** depicts the achieved disk throughput depending on the average IO request size.

V. Experimental Evaluation

Before we present our experimental results, we first provide an overview of Diskbench. Diskbench consists of two separate tools: *Scsibench* and *Idextract*. *Scsibench* runs as a user-level process on Linux systems. It uses our custom user-level SCSI library to access the disk over Linux SCSI generic interface [21], [18]. Using the command line interface, a user can specify features to extract, or traces to execute. *Idextract* runs as a user-level process using Linux raw disk support for accessing any disk. All extraction methods in Idextract rely only on read and write disk commands.

A. Methodology

We now present experimental results for each disk feature described in Section IV on three testbeds. The first testbed is a dual Intel Pentium II 800MHz machine with 1GB of main memory and a 9GB Seagate ST39102LW 10000 RPM SCSI disk (12 disk heads). The second testbed is an Intel Pentium III 800MHz machine with 128MB of main memory and an 18GB Seagate ST318437LW 7200 RPM SCSI disk (2 disk heads). The third testbed is Intel Pentium 4 1500MHz with 512MB of main memory and an 40GB WD400BB-75AUA1 7200 RPM IDE disk.

The first configuration is a typical SMP server system with a fast SCSI disk and a large number of tracks per cylinder. The second configuration is a typical workstation, with a large but slower hard disk (lesser rotation speed). The third configuration is slightly newer PC workstation with IDE disk. We present results for the IDE disk only for methods which differ from SCSI disk methods.

	Disk	T_{rot}(in μs)	RPM	Interrog.
1.	ST39102LW	5972.56	10045.94	10045
2.	ST318437LW	8305.83	7223.84	7200

TABLE I
ROTATIONAL TIME FOR TWO TESTBED DISKS.

B. Rotational Time

Using Equation 4 to calculate the time required for a single rotation of the disk, we obtained rotational times for the two testbed configurations. These are presented in Table I.

C. Variations in OS Delay

Based on Equation 2, we note that the variation in operating system delay for disk accesses is proportional to the variation in completion times for the same request. Using the trace execution described in Figure 5, we measured variations in request completion times (and thus, the distribution of operating system delay variations) for the two testbed configurations. These are presented in Figures 6 and 7.

```
B 0      ; Turn off all disk buffering
R 0 1    ; Read one sector starting from LBN 0
T 2      ; Store current time to register T_2

         ; repeat the following:
R 0 1    ; Read one sector starting from LBN 0
T 3      ; Store current time to register T_3
- 0 3 2  ; Print T_3 - T_2
- 2 3 0  ; T_2 = T_3 - 0
...
```

Fig. 5. Sample trace file to find T_{OS_delay} variations.

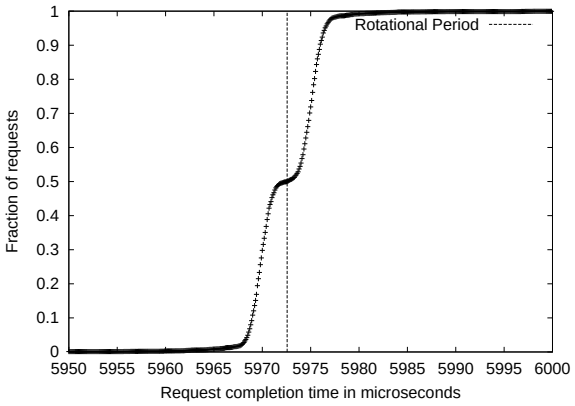


Fig. 6. Distribution of request completion times for Seagate ST39102LW.

Figure 6 shows the results for our first testbed configuration. We can see that the variations in OS delay are of the order of $10\mu s$. Figure 7 shows results for our second testbed configuration. Here the OS delay variations are of the order of $40\mu s$, with greater variations in OS delay as compared to the first testbed.

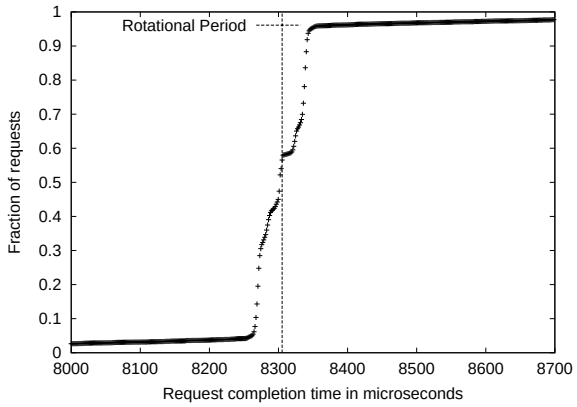


Fig. 7. Distribution of request completion times for Seagate ST318437.

D. Mapping from Logical to Physical Block Address

1) Interrogative Mapping: Sample results from the mapping extraction for our testbed 2 configuration are presented in Figure 8. In each line we print the mapping information for one cylinder. After C we print the cylinder number, the starting LBA, and the cylinder size (in logical blocks). Then we print information for individual tracks after T, namely the track number, its starting LBA, and size (in logical blocks). We print this information for all tracks.

Diskbench stores the starting LBA and the size of each disk track. If the number of bad sectors in a track is greater than the number of spare sectors allocated per each track, then the track size is smaller than the size of a regular track in that zone (cylinders $718 - 721$ in Figure 8).

```
 1) C 0 0 1500 T 0 0 750 1 750 750
 2) C 1 1500 1500 T 0 2250 750 1 1500 750
 3) C 2 3000 1500 T 0 3000 750 1 3750 750
 4) C 3 4500 1500 T 0 5250 750 1 4500 750
 5) C 4 6000 1500 T 0 6000 750 1 6750 750
 6) C 5 7500 1500 T 0 8250 750 1 7500 750
 7) C 6 9000 1500 T 0 9000 750 1 9750 750
 8) ...
 9) C 718 1077000 1499 T 0 1077000 750 1 1077750 749
10) C 719 1078499 1499 T 0 1079248 749 1 1078499 750
11) C 720 1079998 1499 T 0 1079998 750 1 1080748 749
12) C 721 1081497 1499 T 0 1082246 749 1 1081497 750
13) ...
```

Fig. 8. Sample LBA-to-PBA mapping for Seagate ST318437LW.

2) Empirical Mapping: We present results for the empirical extraction of mapping information for testbed 1 in Figures 9-12. This disk has 12 tracks per cylinder. In Figure 10 we present the access time $\Delta T(x, 0)$ (defined in Equation 6) between disk blocks 0 and x. The rotational period of the disk (T_{rot}) is approximately $6ms$. We detail our results in Figure 9, which is an enlargement of small section of Figure 10.

We can see that for small values of x, the access time $\Delta T(x, 0)$ is larger than T_{rot}. When T_{OS_delay} (defined in Section IV-A.2) is larger than the rotational distance between blocks 0 and x, the second read request (to the logical block

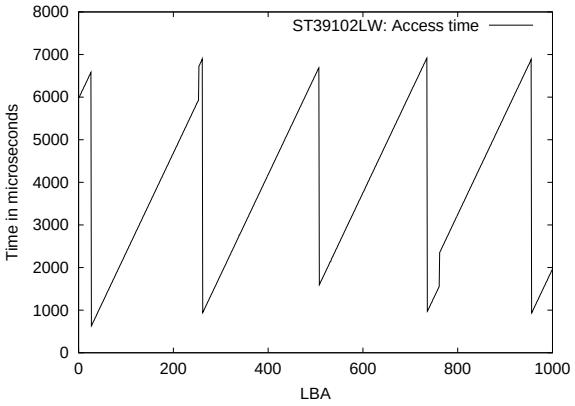


Fig. 9. A sample of $\Delta T(x,0)$ needed to read logical block x (on the X-axis) after reading LBA_0 for the ST39102LW.

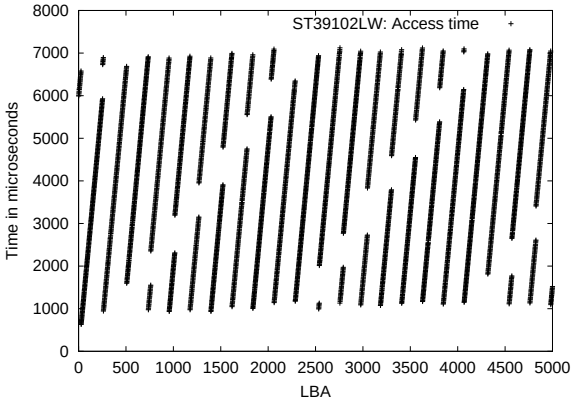


Fig. 10. The time $\Delta T(x,0)$ needed to read logical block x (on the X-axis) after reading LBA_0 for the ST39102LW.

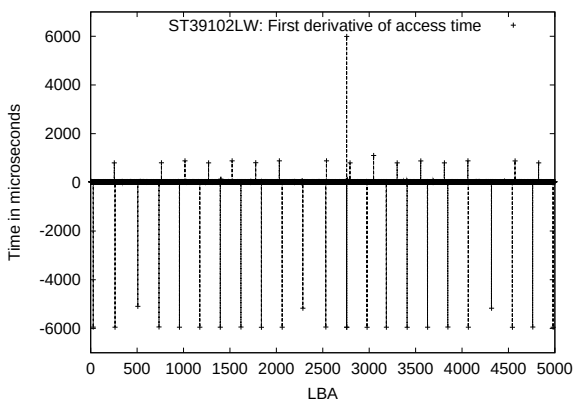


Fig. 11. First derivative (Δ) of access time($\Delta T(x,0)$) for ST39102LW.

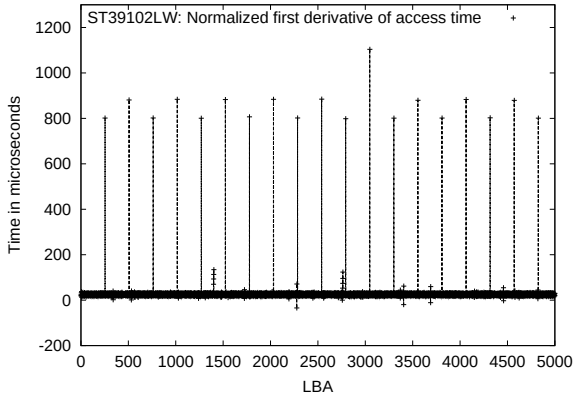


Fig. 12. Normalized first derivative ($norm(\Delta)$) for ST39102LW.

x) has to be serviced during the next disk rotation. When $\Delta T(x,0)$ is greater than T_{OS_delay}, an additional disk rotation is not needed. $\Delta T(x,0)$ increases linearly for all blocks on the same track. When blocks $x-1$ and x are located on different tracks, $\Delta T(x,0)$ increases by the track (or cylinder) skew time (after which $\Delta T(x,0)$ continues to increase linearly). In our example this happens at logical block number 254.

When the access time to the block $x-1$ requires a rotational delay of nearly T_{rot}, and the access to x does not require any rotational delay after seek, $\Delta T(x,0)$ decreases by T_{rot}. This happens at block number 262 for the first time. At the next track boundary (508), a skew time increase and a T_{rot} decrease overlap. Figure 10 shows the $\Delta T(x,0)$ curve for the distances up to 5000 logical blocks.

Figure 11 shows the first derivative of the $\Delta T(x,0)$ curve (Δ) defined in Equation 7. We can see that, since T_{OS_delay} is a random variable, $\Delta T(x,0)$ can also incur a sudden increase of T_{rot} in successive measurements. In this experiment, it occurs at $x = 2758$. This happens when the difference between T_{OS_delay} in successive measurements is substantial, so that $\Delta T(x,0)$ incurs one disk rotation more than $\Delta T(x-1,0)$. A positive T_{rot} increase in $\Delta T(x,0)$ is always followed by a negative T_{rot} decrease in the next few accesses.

In order to perform the empirical mapping automatically, we use several heuristics to find the normalized value for Δ ($norm(\Delta)$) defined in Equation 8. Figure 12 presents $norm(\Delta)$ for our testbed 1, where we capture only the track and cylinder skew times. The positions of the track and cylinder skew times on the x-axis are exact positions of the track and cylinder boundaries (occurring every 254 blocks in Figure 12). The track and cylinder skew times for this disk are approximately $880\mu s$ and $1100\mu s$ respectively. The skew times to switch from an odd to an even track, and from an even to an odd track, are also slightly different ($880\mu s$ and $800\mu s$ respectively).

Figure 13 presents the empirical mapping results for testbed 2. The disk used in this configuration has two tracks per cylinder. Results from Section V-C show that variations in operation system delay, and hence the noise in the measured $norm(\Delta)$, is much higher than for the first testbed configuration. However, since T_{OS_delay} is a random variable, we can repeat the experiment to limit the noise level and extract $norm(\Delta)$ accurately.

Figure 14 presents the empirical mapping results for testbed 3. We can see that Idextract results are similar to Scsibench ones. For this particular disk we are not able to find out the number of tracks per cylinder since the track skew is nearly identical to the cylinder skew time.

E. Seek Curves

In Figure 15 we present the seek curve for our first testbed (ST39102LW). We can see that the difference between the ro-

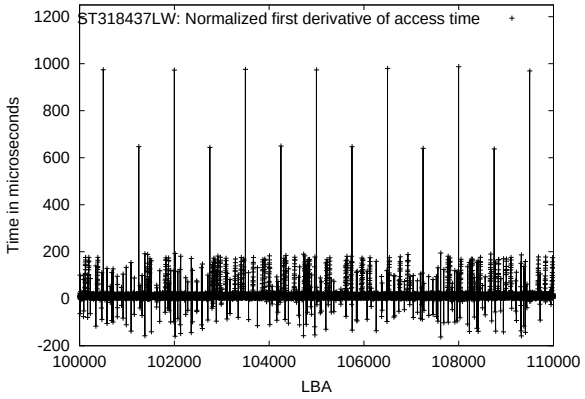


Fig. 13. Normalized first derivative $norm(\Delta)$ for ST318437LW.

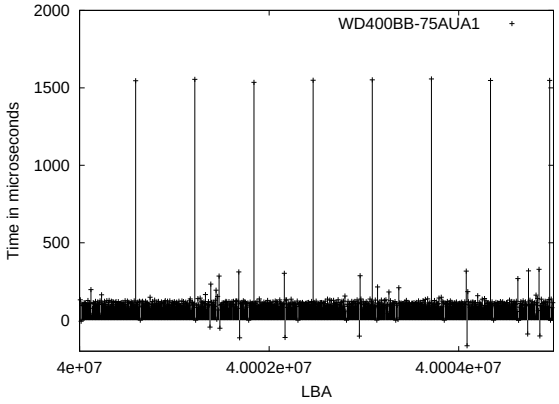


Fig. 14. Normalized first derivative $norm(\Delta)$ for IDE WD400BB-75AUA1.

tational period and the maximum seek time is less than a factor of two. Since the variations in the seek curve are negligible, we can also deduce that the seek time depends mainly on the seek distance in cylinders, and not on starting or destination cylinder positions. Figure 16 presents the seek curve for the second testbed.

F. Read Cache

Figures 17 and 18 present results for the extraction of the cache segment size, using the method explained in Section IV-A.5. Both disks stop prefetching when they fill up the first cache

Zone	*Cylinders*	t_{size}	$\mathcal{TR}$	$\mathcal{TR}_{max}$	$\gamma(1)$	H
1	0-847	254	18.85	21.77	1108	884
2	848-1644	245	18.02	21.01	1108	885
3	1645- 2393	238	17.51	20.40	1098	876
4	2394- 3097	227	16.70	19.45	1114	891
5	3098- 3758	217	15.99	18.60	1115	890
6	3759- 4380	209	15.43	17.91	1105	885
7	4381- 4965	201	14.84	17.23	1100	876
8	4966- 5515	189	13.98	16.20	1123	901
9	5516- 6031	181	13.39	15.52	1125	903
10	6032- 6517	174	12.89	14.92	1107	885
11	6518- 6961	167	12.38	14.31	1118	899

TABLE II
DISK ZONE FEATURES FOR ST39102LW.

Zone	*Cylinders*	t_{size}	$\mathcal{TR}$	$\mathcal{TR}_{max}$	$\gamma(1)$	H
1	0- 4553	750	31.07	46.24	977	652
2	4554- 6582	687	28.94	42.35	985	654
3	6583- 8247	678	28.48	41.80	987	648
4	8248-11554	666	27.92	41.06	1134	651
5	11555-14597	625	27.13	38.53	981	646
6	14598-17370	600	26.00	36.62	983	652
7	17371-19908	583	25.44	35.94	987	657
8	19909-22226	550	24.49	33.91	982	648
9	22227-26338	500	22.74	30.82	982	650
10	26339-28170	458	21.04	28.23	1159	654
11	28171-29850	437	20.24	26.94	990	663

TABLE III
DISK ZONE FEATURES FOR ST318437LW.

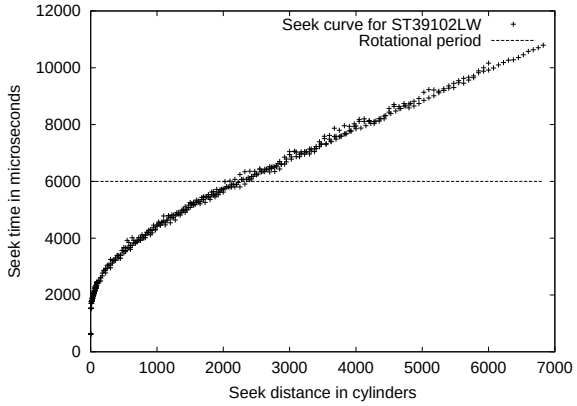


Fig. 15. Complete seek curve for ST39102LW.

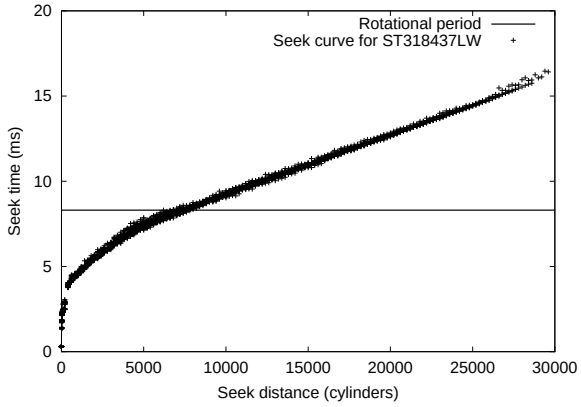


Fig. 16. Complete seek curve for ST318437LW.

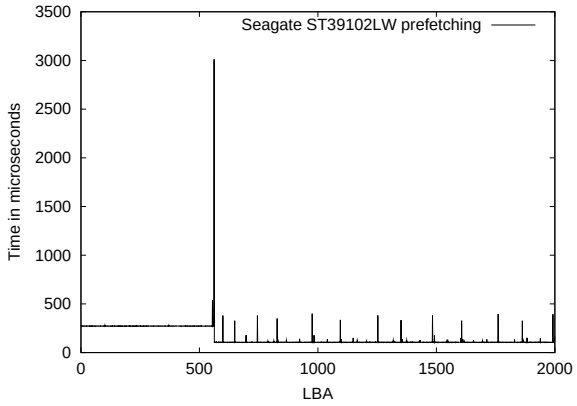


Fig. 17. Read request completion times for ST39102LW.

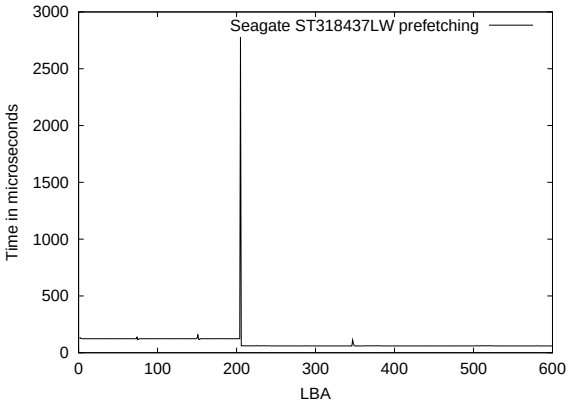


Fig. 18. Read request completion times for ST318437LW.

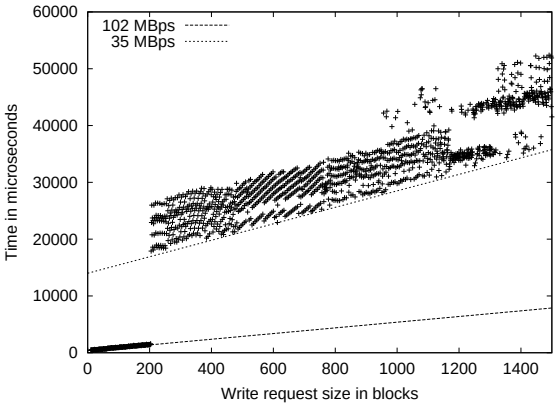


Fig. 20. Write request completion times for ST318437LW

segment. The extracted sizes for testbeds 1 and 2 are 561 and 204 blocks respectively. We also note that both disks continue prefetching into the next available cache segment, when they detect a long sequential access.

Using the extracted cache segment size, we can find out the number of cache segments in the disk read cache, as explained in Section IV-A.5. Using this method, we detected three cache segments for testbed 1, and 16 segments for testbed 2.

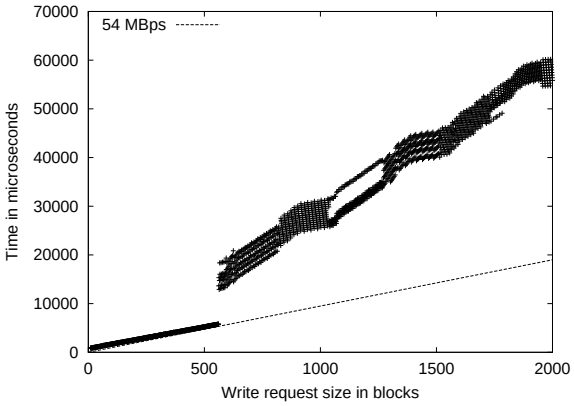


Fig. 19. Write request completion times for ST39102LW.

1) Disk Zones: Tables II and III present the zoning information extracted for the Seagate ST39102LW and Seagate ST318437LW disks respectively. t_{size} denotes the track size in logical blocks. $\mathcal{TR}$ is the transfer rate measured for long sequential reads that span multiple cylinders. $\mathcal{TR}_{max}$ is the calculated theoretical maximum transfer rates for read requests which incur no seek, rotation or switching overheads. H and $\gamma(1)$ are the track and cylinder switch times in microseconds.

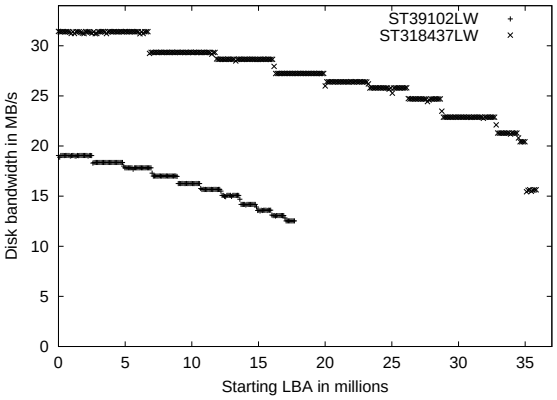


Fig. 21. Disk bandwidth depending on data location for two SCSI disks.

G. Write Buffer

Figures 19 and 20 present the results for write buffer size extraction using the method presented in Section IV-A.5.

The write buffer size for testbed 1 and testbed 2 were measured to be 561 and 204 blocks respectively. Comparing these to the cache segment sizes extraction presented in Section V-F, we can see that both disks use exactly one cache segment as a write buffer.

Using these measurements we can also measure the write throughput, both to the disk write buffer (slope of the curve for write request sizes that fit into write buffer), and to the disk platter (slope for request sizes greater than the write buffer size). In the future we plan to extract the number of cache segments that can be used as write buffers. We believe that we can use a method similar to the method we used for detecting the number of read cache segments presented in Section IV-A.5.

Figure 21 depicts the disk bandwidth for large sequential accesses depending on the starting LBA for testbed 1 and 2. For modern disks, the difference between the maximum and minimum sequential disk bandwidth is usually a factor of two. Figure 22 presents the disk zone bandwidths for testbed 3.

H. Rotational Delay Prediction

From Equation 12, in order to predict the rotational delay accurately, we need to extract the skew factor (defined in Equation 9) for each track. Sample results for the extracted skew factors (in μs) are presented in Table IV, wherein we also present the LBAs for blocks residing on the same disk radius (LBA_{rot0}).

In Figure 23, we plot the skew times against track numbers. We notice a distinct trend in skew times, a property which enables us to compress this information effectively and to reduce its space requirement. In Figure 23, we also notice a slight deviation from the normal trend for tracks 12, 24 and 36. This is

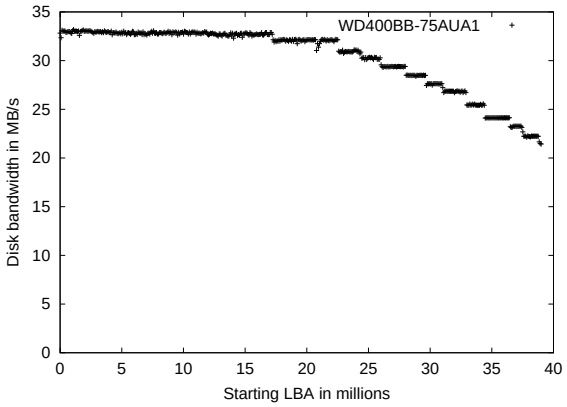


Fig. 22. Disk bandwidth depending on data location for an IDE disk.

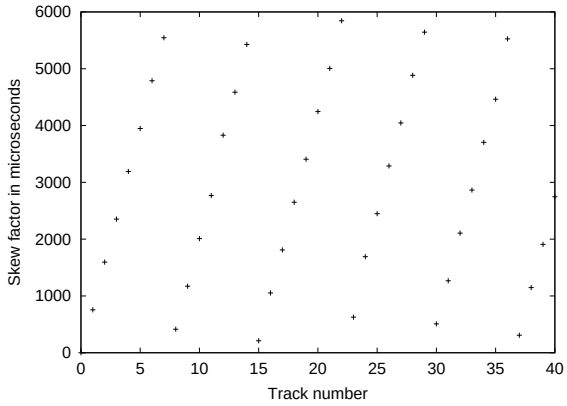


Fig. 23. Skew factor s from Table IV for ST39102LW.

Cyl	$Track$	$Skew(\mu s)$	LBA_{rot0}
0	0	0	0
0	1	755.15	475
0	2	1595.02	694
0	3	2352.35	915
0	4	3191.09	1134
0	5	3949.03	1356
0	6	4788.61	1574
0	7	5546.36	1796
0	8	414.31	2268
0	9	1170.86	2490
0	10	2009.76	2708
0	11	2767.37	2930
1	0	3829.42	3139
...	...	...	...

TABLE IV

ROTATIONAL DELAY MODELING FOR ST39102LW (THE TRACK SIZE FOR THE FIRST ZONE IS 254). DISK BLOCKS WITH LBA_{rot0} ARE ON THE SAME DISK RADIUS.

due to cylinder skew, which occurs when the next track falls on an adjacent cylinder instead of the same cylinder. The experimental disk had exactly 12 surfaces. Hence, we expect a trend deviation on tracks that are multiples of 12 to account for an increased switching overhead.

Based on Equation 12 and the compressed information about skew times above, we were able to predict the rotational delay between two disk accesses. In Figure 24, we present the error distribution of rotational delay predictions for a large number of random request-pairs. We note that for the SMP-like testbed (testbed 1), which has a very predictable distribution of OS delay variations (Figure 6), our prediction is accurate within 25 μs for 99% of the requests. Even for the workstation-like testbed (testbed 2), which has less predictable OS delay variations (Figure 7), our prediction is accurate within 80 μs for 99% of the requests. These errors are negligible compared to variations in seek time, which are of the order of a millisecond. We thus conclude that with detailed disk parameters, systems can implement very accurate mechanisms for predicting rotational delays. We used seek time and rotational delay predictions from Diskbench to predict disk access times in the implementation of [3].

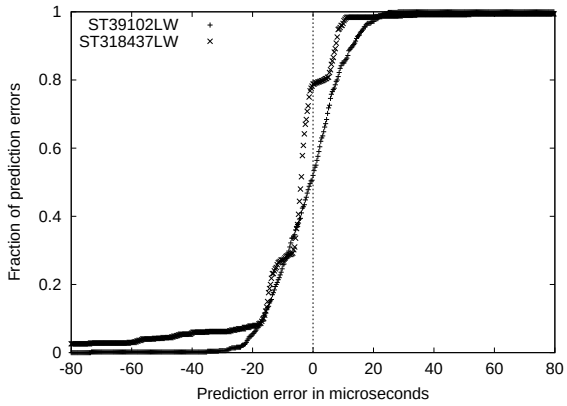


Fig. 24. Rotational delay prediction accuracy for ST39102LW and ST318437LW.

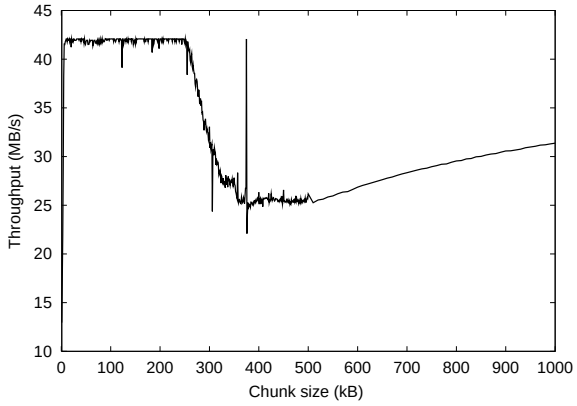
I. Sequential Throughput and Chunking

As regards chunking, the disk profiler provides the optimal range for the chunk size. Figure 25 depicts the effect of chunk size on the read throughput performance for one SCSI and one IDE disk drive. Figure 26 shows the same for the write case. Clearly, the optimal range for the chunk size (between the points a and b illustrated previously in Figure **??**) can be automatically extracted from these figures.

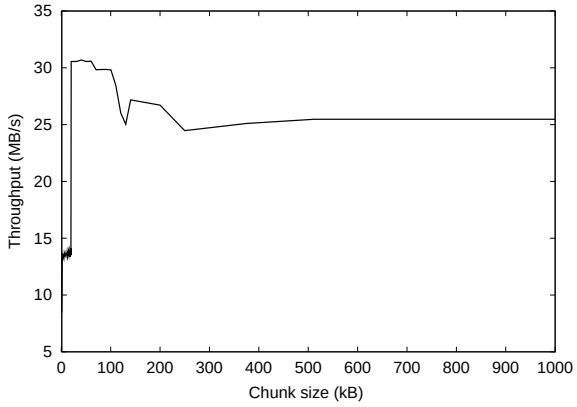
VI. SUMMARY

We have presented Diskbench, a user-level tool for disk feature extraction. Diskbench uses both interrogative and empirical methods to extract disk features. The empirical methods extract accurate low-level disk features like track and cylinder boundaries, track and cylinder skew times, the number of tracks per cylinder, the track sizes (in logical blocks), and the read and write buffer parameters. Diskbench also extracts high-level disk features necessary for advanced scheduling methods like our Semi-preemptible IO [2] or rotationally-aware schedulers [4], [5], [6], [7], [15], [8].

We believe that this work can be used by system and application programmers to improve and guarantee real-time disk performance. Using knowledge about disk features provided by Diskbench, system or application programmers can fine-tune disk accesses to match application requirement and can predict the disk performance, which is necessary for real-time disk scheduling.

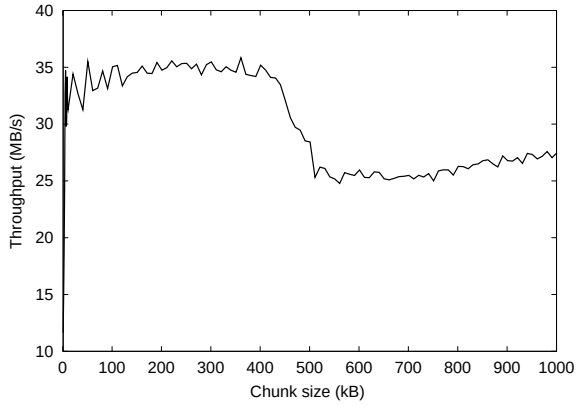


(a) SCSI ST318437LW

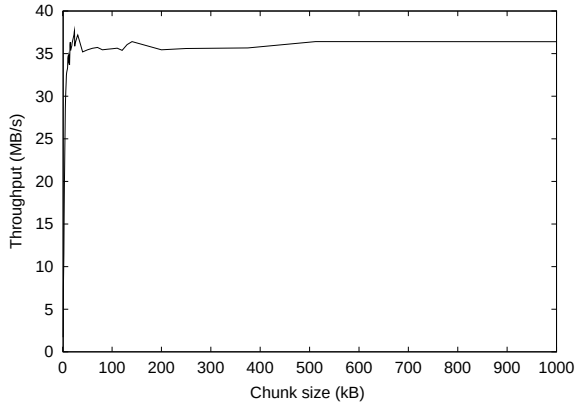


(b) IDE WD400BB

Fig. 25. Sequential read throughput vs. chunk size.



(a) SCSI ST318437LW



(b) IDE WD400BB

Fig. 26. Sequential write throughput vs. chunk size.

REFERENCES

[1] Zoran Dimitrijevic, Raju Rangaswami, and Edward Chang, "The XTREAM multimedia system," *IEEE Conference on Multimedia and Expo*, August 2002.

[2] Zoran Dimitrijevic, Raju Rangaswami, and Edward Chang, "Design and implementation of Semi-preemptible IO," *Proceeding of Usenix FAST*, March 2003.

[3] Zoran Dimitrijevic, Raju Rangaswami, and Edward Chang, "Virtual IO: Preemptible disk access," *Proceedings of the ACM Multimedia*, December 2002.

[4] Lan Huang and Tzi-cker Chiueh, "Implementation of a rotation-latency-sensitive disk scheduler," *SUNY at Stony Brook Technical Report*, May 2000.

[5] David M. Jacobson and John Wilkes, "Disk scheduling algorithms based on rotational position," *HPL Technical Report*, February 1991.

[6] Christopher R. Lumb, Jiri Schindler, Gregory R. Ganger, and David F. Nagle, "Towards higher disk head utilization: Extracting free bandwith from busy disk drives," *Proceedings of the OSDI*, 2000.

[7] Bruce L. Worthington, Gregory R. Ganger, and Yale N. Patt, "Scheduling algorithms for modern disk drives," *Proceedings of the ACM Sigmetrics*, pp. 241–251, May 1994.

[8] Eno Thereska, Jiri Schindler, John Bucy, Brandon Salmon, Christopher R. Lumb, and Gregory R. Ganger, "A framework for building unobtrusive disk maintenance applications," *Proceedings of the Third Usenix FAST*, March 2004.

[9] Zoran Dimitrijevic, David Watson, and Anurag Acharya, "Scsibench," *http://www.cs.ucsb.edu/∼zoran/scsibench*, 2000.

[10] B. L. Worthington, G. Ganger, Y. N. Patt, and J. Wilkes, "Online extraction of scsi disk drive parameters," *Proceedings of the ACM Sigmetrics*, pp. 146–156, 1995.

[11] Mohamed Aboutabl, Ashok Agrawala, and Jean-Dominique Decotignie, "Temporally determinate disk access: An experimental approach," *Univ. of Maryland Technical Report CS-TR-3752*, 1997.

[12] Nisha Talagala, Remzi H. Arpaci-Dusseau, and David Patterson, "Microbenchmark-based extraction of local and global disk characteristics," *UC Berkeley Technical Report*, 1999.

[13] Jiri Schindler and Gregory R. Ganger, "Automated disk drive characterization," *CMU Technical Report CMU-CS-00-176*, December 1999.

[14] Zoran Dimitrijevic, Raju Rangaswami, Edward Chang, David Watson, and Anurag Acharya, "Diskbench," *http://www.cs.ucsb.edu/∼zoran/diskbench/*, 2002.

[15] Christopher R. Lumb, Jiri Schindler, and Gregory R. Ganger, "Freeblock scheduling outside of disk firmware," *Proceedings of the Usenix FAST*, January 2002.

[16] Chris Ruemmler and John Wilkes, "UNIX disk access patterns," *Usenix Conference*, pp. 405–420, Winter 1993.

[17] A. N. S. I., "Scsi-2 specification x3t9.2/375r revision 10l," January 1995.

[18] Seagate, "Scsi interface, product manual 2," *http://www.seagate.com/support/disc/manuals/scsi/38479j.pdf*, April 1999.

[19] M. McKusick, W. Joy, S. Leffler, and R. Fabry, "A fast file system for unix*," *ACM Transactions on Computer Systems 2*, vol. 3, pp. 181–197, August 1984.

[20] S. J. Daigle and J. K. Strosnider, "Disk scheduling for multimedia data streams," *Proceedings of the IS&T/SPIE*, February 1994.

[21] Douglas Gilbert, "The Linux SCSI generic howto," *http://tldp.org/HOWTO/SCSI-Generic-HOWTO*, 2002.