Techniques to Quantify SCSI-2 Disk Subsystem Specifications for Multimedia*

Shahram Gandeharizadeh, James Stone, Roger Zimmerman

Department of Computer Science
University of Southern California
Los Angeles, California 90089

March 20, 1995

Abstract

Magnetic disk technology has established itself as the mass storage device of choice in the commercial arena. Most often, the operating system of a hardware platform that employs this device (e.g., a Unix™-based workstation) hides its physical attributes by conceptualizing it as an array of blocks. This abstraction expedites program development time because the programmer is no longer concerned with the working details of the mass storage device. Moreover, the final program is portable as long as a new target platform provides an identical abstraction of its magnetic disk drive.

This paradigm is effective for those applications with no real-time constraints. However, certain applications (e.g., multimedia) cannot tolerate significant variations in the service time of the disk drive. They must estimate the service time of the disk accurately in order to both support real-time constraints of the application and schedule the disk effectively in the presence of multiple requests. Assuming a Unix-based workstation, this study reports on our experimental techniques to identify the physical details of a magnetic disk drive. The models constructed based on the results obtained from these experiments can be used by a program to satisfy its real-time constraints.

1 Introduction

The magnetic disk drive technology has benefited from more than two decades of research and development. It has evolved to provide a low latency (in the order of milliseconds) and a low cost per megabyte of storage (approximately 50 cents at the time of this writing). It has become common place with annual sales in excess of 30 billion dollars [oST94] (far exceeding that of both CD Drives and Tape recorders whose combined sales is less than 10 billion dollars). Both workstations and personal computers employ magnetic

---

*This research was supported in part by the National Science Foundation under grants IRI-9110522, IRI-9258362 (NYI award), and CDA-9216321, a grant from DOD/Intelligence Community, and a Hewlett-Packard unrestricted cash/equipment gift.
disks as non-volatile storage devices. Their operating systems provide the application programmer with the abstract perspective that the storage capacity of this device is a linear array of fixed size blocks. Such a system simplifies program development by allowing the programmer to focus on the task at hand instead of the physical attributes of the disk drive. Nonetheless, a number of data intensive applications (e.g., multimedia) must estimate the service time of a disk in order to satisfy their real-time constraints. This study focuses on the Unix operating system (due to its popularity) and investigates techniques to identify the physical characteristics of a magnetic disk in order to estimate its service time. These estimates can be used to develop, implement, and evaluate a real-time application. Our techniques employ no proprietary information and are quite successful in developing a methodology to estimate the service time of a disk. However, these techniques do identify a few attributes of magnetic disks that are difficult to explain (without proprietary information).

Our proposed techniques are a contribution because they demonstrate that a programmer does not need to abandon the benefits of the development environment provided by an operating system to support multimedia applications (or other applications with a real-time constraint). A program is portable as long as it consumes the physical workings of a magnetic disk drive as its input parameters. Each time a program is ported to a new platform, the system administrator computes the physical characteristics of the new disk drive using our techniques and provides them as input to the program.

Techniques to estimate the disk service time have been proposed in [RW94, Wil77]. The previous studies proposed models that provide an average estimate of the disk service time. These models were targeted for use in simulation studies to yield realistic results. Our models are designed to estimate the service time of a disk drive to enable an application program to satisfy its real-time constraints. As such, our models do not underestimate the service time of a disk drive. This may reduce the overall processing capability (throughput) of a system, however, this limitation can be eliminated (once a program is debugged) by introducing a specialized software layer that tailors the system to the physical details of a disk drive.
Moreover, we report on the scalability characteristics of the SCSI-2 bus. This information has not been reported by the previous literature.

The rest of this paper is organized as follows. Section 2 provides an overview of a target platform configured with the Unix operating system that employs a magnetic disk drive. Next, Section 3 focuses on multimedia information systems and describes why it is important to estimate the service time of a disk drive to maximize the numbers of simultaneous displays of video clips supported by the system. In Sections 4 and 5, we focus on the HP-UX operating system and develop several experimental techniques to identify the physical characteristics of the disk drive. Using this information, Section 6 develops a methodology to estimate the service time of the disk drive. We present brief conclusions and future research directions in Section 7.

2 Overview of the Target Development Environment

![Figure 1: Architecture of the Target Environment](image)

An abstract description of our target environment is provided in Figure 1. The system bus provides a very high performance (nanosecond latency and more than 25 MBytes per second transfer rate once the bus arbitration overhead is considered). The \textit{SCSI (Small Computer System Interface)} bus supports a peak
transfer rate of 5, 10, or 20 MBytes per second depending on whether it is Narrow, Fast, or Fast&Wide, respectively. The disk subsystem is central to this study and is detailed in Section 2.1. Section 2.2 provides an overview of our target operating platform.

2.1 Overview of the SCSI-2 Disk Subsystem

This section starts with an overview of a magnetic disk drive. Subsequently, it describes the SCSI-2 host bus adapter and how it interfaces with multiple disk drives.

2.1.1 Overview of a Magnetic Disk Drive

A magnetic disk drive is a mechanical device, operated by its controlling electronics. The mechanical parts of the device consist of a stack of platters that rotate in unison on a central spindle (see [RW94] for details). Presently, a single disk contains one, two, or as many as sixteen platters (see Figure 2). Each platter surface has an associated disk head responsible for reading and writing data. Each platter is set up to store data in a series of tracks. A single stack of tracks at a common distance from the spindle is termed a cylinder. To access the data stored in a track, the disk head must be positioned over it. The operation to reposition the head from the current track to the desired track is termed seek. Next, the disk must wait for the desired data to rotate under the head. This time is termed rotational latency time.
To meet the demands for a higher storage capacity, disk drive manufacturers have introduced disks with zones. A zone is an contiguous collection of disk cylinders whose tracks have the same storage capacity. Tracks are longer towards the outer portions of a disk platter as compared to the inner portions; more data may be recorded in the outer tracks. While zoning increases the storage capacity of the disk, it produces a disk that does not have a single transfer rate. The multiple transfer rates are due to (1) the variable storage capacity of the tracks and (2) a fixed number of revolutions per second for the platters.

2.1.2 Overview of the SCSI-2 Bus Operation

The SCSI-2 standard is both a bus specification and a command set to efficiently use that bus (see [Ded94]). Figure 1 shows how the SCSI host bus adapter (also called initiator) links the system bus (based on any standard or proprietary bus) to the SCSI-2 bus. The storage devices (also called targets) are attached to the SCSI bus. The goal of the SCSI standard was to provide a device independent abstraction for different storage devices to the system. For example, the only characteristic that differentiates among magnetic disk drives of different vendors is their maximum capacity.

The SCSI devices and host bus adapters in our systems are of two varieties, Fast (10 MBytes/second transfer rate) and Fast & Wide (20 MBytes/second transfer rate). The data path of a fast SCSI bus is one byte wide and it can support a maximum of eight devices due to its addressing scheme. One of them is the host bus adapter itself. Every SCSI device has a fixed priority that directly corresponds to its SCSI ID. This number ranges from 0 to 7 in value. The host bus adapter usually is assigned ID number 7 because it corresponds to the highest priority\(^1\). A fast & wide SCSI-2 bus can transfer two bytes of data in parallel and support up to sixteen devices (numbered 0 to 15). Because of the wider data path, the bus bandwidth is twice that of the narrow version.

The transfer rate of a typical magnetic disk is 2 to 4 MBytes per second. The higher bandwidth of a

---

\(^1\) The wide SCSI specification was introduced after the narrow version. Consequently, the priorities in terms of SCSI IDs are 7...0, 15...8, with 7 being the highest and 8 the lowest.
Figure 3: Timing of overlapped read operations with SCSI devices that implement the disconnect/reconnect feature.

SCSI bus can be utilized because many disks implement the disconnect/reconnect feature. Figure 3 shows how the read operations of several disks can be overlapped. After receiving a command from the host bus adapter, the target disconnects itself from the bus and performs the seek operation offline. It then waits until the first of the requested blocks rotates underneath the disk head. When the actual read operation starts, the data is not immediately sent over the SCSI bus. Instead it is stored in a small buffer. Whenever this buffer is full, the disk reconnects to the bus and transfers the accumulated data to the host bus adapter in a burst that matches the maximum speed of the SCSI bus. Hence, a device that executes a long read may be connected to the bus for only 20% to 40% of the time. This leaves room for other devices on the same bus to receive and execute commands concurrently. With this technique of overlapped reads and writes the overall throughput of the system can be improved. However, the SCSI bus starts to become a bottleneck beyond a certain number of disks. This causes requests referencing the lowest priority disk to observe the worst service time because disks gain access to the bus based on priority.

2.2 Overview of the Unix Operating System

Unix, of which HP-UX is a particular implementation, is a general-purpose operating system. Several of its features are central to this study. These features are designed to make general system programming and operation easier. However, at times they hinder our experiments with a magnetic disk drive.
The access time of a mechanical disk drive is much slower than that of system memory. Therefore, Unix maintains the most recently referenced pages in a memory structure, termed a buffer pool, in order to minimize the number of accesses to the disk drive. If a referenced page is resident in the buffer pool, no I/O request is issued to the disk. While this is a useful service, its presence impacts our experiments because it might translate a disk page request to a memory reference.

Unix provides a consistent interface to devices. All devices are abstracted as files, which may be opened, read from, written to, and closed using standard programming language functions. An addition to the operating system, termed a device driver, is provided for each specific device to present this abstraction. Access to these device files uses one of two methods, either through a block interface (which employs the buffer pool), or through a raw interface (which bypasses the buffer pool).

The Unix system is both multi-tasking (more than one process may be simultaneously executing at a time) and multi-user (more than one user may be simultaneously executing processes). In addition to user processes, the operating system has processes of its own that run without direct user intervention. These system processes periodically perform various housekeeping tasks for the system, and cannot be preempted by user processes.

3 Continuous Display of Video Objects Using a Magnetic Disk Drive

Video objects must be retrieved and displayed at a fixed rate in order to ensure their continuous display. Otherwise, their display will suffer from frequent disruptions and delays, termed hiccups. To simplify the discussion, assume that the bandwidth required to display an object \( R_C \) is lower than the bandwidth of the disk drive. To support a continuous display of a video object \( X \) using a magnetic disk drive, several studies [CI93, Pol91, BGMJ94] have proposed the following approach. First, object \( X \) is partitioned into \( n \) subobjects: \( X_1, X_2, \ldots, X_n \). The size of each subobject is a function of the bandwidth required to ensure a continuous display and the bandwidth of the disk drive. This size is fixed for objects with the same
bandwidth requirement. When an object $X$ is referenced, the system stages its first block (say $X_1$) in memory and initiates its display. Before the display of $X_1$ completes, the system retrieves $X_2$ into memory in order to ensure a continuous display. This process is repeated until all blocks of an object have been displayed.

![Diagram](image)

**Figure 4: Time Period**

Since the bandwidth of the disk exceeds $R_C$, the system can support simultaneous displays of several objects. Figure 4 demonstrates the continuous display of $n$ objects. In this figure, the time to display a block is termed a *time period*. This time is partitioned into slots, with each slot corresponding to the retrieval time of a block from the disk drive. The number of slots in a time period defines the number of simultaneous displays that can be supported by the system. For example, a block size of 750 KBytes corresponding to a MPEG-1 compressed movie ($R_C = 1.5$ Mbps) has a 4 second display time ($T_p = 4$). Assuming a magnetic disk with a transfer rate of 24 Mbps ($t_{fr} = 24$ Mbps) and maximum seek time of 35 milliseconds, 14 such blocks can be retrieved in 4 seconds. Hence, a single disk supports 14 simultaneous displays. As demonstrated by this example, in order to maximize the number of simultaneous displays, the system must accurately estimate the worst seek time and transfer rate of the disk in order to schedule requests effectively.

8
The previous example is a simplification because it assumes a constant transfer rate for the disk. As described in Section 2.1.1, the platters of currently available disk drives are partitioned into zones that have a varying number of sectors. This results in different transfer rates for different zones. The system designer should take the zoning information into account to best utilize a disk drive for the retrieval of continuous media objects. Otherwise, the designer must assume the worst case transfer rate that will result in a considerable waste of the disk bandwidth for all but one (the slowest) zone.

If the disk bandwidth is multiplexed among different displays (see Figure 4), the disk heads must be repositioned to a different location for each subobject. This results in a wasteful seek operation that consumes the time $T_{\text{seek}}$ (see Figure 4) yet transfers no data. This time is a function of the distance (in cylinders) that the disk head must travel. The designer can enhance the utilization of the disk drive by estimating the seek time accurately as opposed to using the worst case value.

In addition to ensuring a continuous display, a system designer must consider the performance requirements of an application. The bandwidth of a single disk drive might be insufficient to support the number of simultaneous displays required by an application. One may assume a multi-disk platform consisting of $D$ disks to resolve this limitation. However, notice from Figure 1 that the SCSI-2 bus is a shared resource. Increasing the value of $D$ will not increase the number of simultaneous displays when the SCSI-2 bus becomes a bottleneck resource. (See the discussion of Section 2.1.2.)

For this study we undertook experiments to learn about the SCSI-2 disk I/O subsystem: the SCSI-2 bus, zoning and the seek profile of several SCSI-2 disk drives. However, all the disks available to us have cache memory on the controller board. Although the cache operation normally only improves the performance of a disk drive, we wanted to be sure about its impact on our measurements so we devised a way to selectively enable and disable it. The effect of the cache operation is mentioned in the following sections whenever it is relevant to the results.
3.1 Test Platforms

Table 1 lists the configuration of the platforms that were used for the tests in this study.

<table>
<thead>
<tr>
<th>System Type</th>
<th>System 1</th>
<th>System 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Workstation HP 9000/720</td>
<td>Workstation HP 9000/735</td>
</tr>
<tr>
<td></td>
<td>PA-RISC, 50 MHz</td>
<td>PA-RISC, 99 MHz</td>
</tr>
<tr>
<td></td>
<td>32 MB</td>
<td>144 MB</td>
</tr>
<tr>
<td>CPU</td>
<td>Built-in (Fast)</td>
<td>Built-in (Fast &amp; Wide)</td>
</tr>
<tr>
<td>Main Memory</td>
<td>Quantum PD425S, 412 MB</td>
<td>Hewlett-Packard C2247, 1 GB</td>
</tr>
<tr>
<td>SCSI-2 System Disk</td>
<td>Hewlett-Packard C2247-300, 1 GB</td>
<td>Seagate ST31200W, 1 GB</td>
</tr>
<tr>
<td>SCSI-2 Auxiliary Disk(s)</td>
<td>HP-UX 9.05</td>
<td>HP-UX 9.05</td>
</tr>
<tr>
<td>Operating System</td>
<td>HP-UX cc &amp; GNU gcc</td>
<td>HP-UX cc &amp; GNU gcc</td>
</tr>
<tr>
<td>C Compiler</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Physical Network Connection</td>
<td>Single-user</td>
<td>Single-user</td>
</tr>
<tr>
<td>Operation Mode</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Configuration of test platforms.

4 HP-UX Low-level SCSI Programming Interface

In order to analyze the characteristics of a disk drive we disabled the Unix buffer manager to ensure that all the read and write requests in our test programs would be executed by the physical disk devices. In addition, we issued seek commands to the magnetic disks as well as enabled or disabled the on-device read caches. In the following, we describe how this was achieved using the HP-UX operating system.

4.1 Bypassing Input/Output Buffering

Unix does not buffer I/O requests to a disk if it is accessed as a raw device. In order to use a device in raw mode its corresponding character special file must be opened. For example, in HP-UX these files are located in the `/dev/rdsk` directory and might appear as:

`/dev/rdsk/c207d2s0`

After a disk is opened as a raw device, the standard C library calls `read()` and `write()` can be used to access the data on the device. Note, however, that this level perceives a disk as a linear array of blocks. If
a Unix file system exists on the disk, it can be destroyed by accidentally overwriting inode tables or other crucial information.

4.2 Executing Direct SCSI Commands

Normally SCSI devices are controlled by a device-type-specific driver. This device driver then presents the usual device-as-a-file abstraction to the programmer. With HP-UX, however, it is possible to explicitly send SCSI commands to a device through the ioctl() system call. This system call requires the following parameters:

- A file descriptor that was obtained by opening the character special device with the open() system call.
- A call identification. The ioctl() system call is very generic and can be used to perform a variety of functions on character special files. In HP-UX the macro SIOC_ID0 identifies the programmer’s intent to directly execute a SCSI command.
- A data structure that contains the input and output parameters for the request. In HP-UX the fields of a structure of type sctl_io must be filled in.

For HP-UX some of the necessary information on how to set up a SCSI ioctl call is contained in the scsi.h header file in the /usr/include/sys directory. Additional information can be found in the system manual pages of diskio, scsi, and scsi_ctl. The references described so far only address the question of how to execute a SCSI ioctl call. The syntax and semantics of the actual SCSI commands are defined in the SCSI standard document [ANS94]. There are three types of SCSI commands: those of length 6, length 10, and length 12 bytes. The first byte of the SCSI command is called the operation code and uniquely identifies the SCSI function. The rest of the bytes are used for parameter passing. The standard also allows vendor-specific extensions for some of the commands which are described in the vendor’s technical manual(s) of the device in question. For example, for our experiments involving the Seagate ST31200W disk drive, we consulted the technical manuals [Sea94b] and [Sea94a].
Figure 5: Structure of the caching page as defined in the SCSI standard for direct access devices.

4.2.1 Example: Enabling and Disabling the On-Device Read Cache

Many of the newer magnetic disk drives are equipped with cache memory on the controller board. The SCSI standard [ANS94] specifies that it should be possible to enable and disable the cache function. If an on-device cache exists, then the related parameters are grouped together into a block termed a caching page. Figure 5 shows the structure of the cache parameter page from the SCSI standard document [ANS94]. Two control bits within this page are used directly to enable and disable caching:

- **WCE** (byte 2, bit 2) - Write Cache Enable: This bit enables the write caching when set to one.
- **RCD** (byte 2, bit 0) - Read Cache Disable: This bit disables the read caching when set to one.

Figures 6 and 7 show sample code fragments in C on how to issue direct SCSI commands. The function `GetCacheParams()` in Figure 6 uses the SCSI MODE SENSE command to retrieve the caching page. The function `SetCacheParams()` depicted in Figure 7 first calls `GetCacheParams()` to read the current cache related values and then issues a MODE SELECT command to write them back to the device after possibly changing the values of the WCE and RCD control bits.
static
int GetCacheParams (fd, buff)
/* This function issues a MODE SENSE(6) command to the device whose */
/* file descriptor is fd. It returns the parameter page that de- */
/* scribes the cache. The returned data consists of two parts: */
/* - a 4 byte mode parameter sense header and */
/* - the 12 to 20 byte cache parameter page. */
/* These two parts are returned in a ‘mode_6_buff_no_bd’ structure.*/  
int fd;    /* File descriptor. */
struct mode_6_buff_no_bd *buff;     /* Caching page. */
{
    /* The following is adapted from 'man scsi_ctl': */
    memset(buff, 0, BUFF_LEN);   /* Clear reserved fields. */
    memset(&sctl_io, 0, sizeof(sctl_io)); /* Input data is expected. */
    sctl_io.flags = SCTL_READ;    /* MODE SENSE(6) command. */
    sctl_io.cdb[0] = CMDmode_sense; /* Disable block descriptor. */
    sctl_io.cdb[2] = 0x00 | CACHING_PAGE; /* Ask for caching page. */
    sctl_io.cdb[3] = 0x00;        /* Reserved. */
    sctl_io.cdb[4] = BUFF_LEN;  /* Allocation length. */
    sctl_io.cdb[5] = 0x00;       /* Control. */
    sctl_io.cdb_length = 6;       /* 6 byte command. */
    sctl_io.data = buff;         /* Data buffer location. */
    sctl_io.data_length = BUFF_LEN; /* Maximum transfer length. */
    sctl_io.max_msecs = 10000;   /* Allow 10 seconds for command. */
    if (ioctl(fd, SIOC_IO, &sctl_io) < 0) {   /* Request was invalid. */
        printf("Error: request was invalid!\n");
        return(-1);
    }
    else if (sctl_io.cdb_status == S_GOOD) {    /* Device is ready. */
        return(0);
    }
    else {  /* Unknown state or device is not ready. */
        printf("Error: unknown state device is not ready!\n");
        return(-1);
    }
}

Figure 6: Sample code fragment to retrieve the cache parameter page of a disk drive.

## 5 Experimental Techniques

### 5.1 Quantifying the Overhead of Seek Operations

To maximize disk utilization, read and write operations must be carefully scheduled. The more information the scheduling algorithm has about the seek profile of a disk, the “tighter” the disk operations can be scheduled (otherwise worst case values must be assumed) and the higher the overall utilization of the system. A test program that measures the seek profile of a magnetic disk can therefore provide useful
int SetCacheParams(fd, bWce, bRce)

/* This routine can set/clear the WCE (Write Cache Enable) and the RCD */
/* (Read Cache Disable) bits on a SCSI-2 disk. It will return 0 on */
/* success and -1 on error. */
int fd; /* File descriptor. */
int bWce; /* 1 - enable, 0 - disable write cache. */
int bRce; /* 1 - enable, 0 - disable read cache. */
{
    unsigned char parm_len;
    struct mode_6_buff_no_bd buff;

    /* Get the current cache page parameters. */
    if (GetCacheParams(fd, &buff) != 0) {
        parm_len = buff.mode_parms.length + 1;
        if (parm_len > BUFF_LEN) {
            printf("Error: buffer size mismatch!
");
            return(-1);
        }
    }
    else return(-1);

    /* Set/reset the cache control flags. */
    /* bWce: 1 enables cache, therefore we must SET WCE. */
    if (bWce) buff.caching_page.control_bits |= WCE;
    else buff.caching_page.control_bits &= ~WCE;
    /* bRce: 1 enables cache, therefore we must CLEAR RCD. */
    if (bRce) buff.caching_page.control_bits &= ~RCD;
    else buff.caching_page.control_bits |= RCD;

    /* Now write the page. */
    memset(&sctl_io, 0, sizeof(sctl_io)); /* Clear reserved fields. */
sctl_io.flags = 0x00; /* No input data is expected. */
sctl_io.cdb[0] = CMDmode_select; /* MODE SELECT(6) command. */
sctl_io.cdb[1] = 0x00 | PF; /* Page format. */
sctl_io.cdb[2] = 0x00; /* Reserved. */
sctl_io.cdb[3] = 0x00; /* Reserved. */
sctl_io.cdb[4] = parm_len; /* Parameter list length. */
sctl_io.cdb[5] = 0x00; /* Control. */
sctl_io.cdb_length = 6; /* 6 byte command. */
sctl_io.data = &buff; /* Data buffer location. */
sctl_io.data_length = parm_len; /* Maximum transfer length. */
sctl_io.max_msecs = 10000; /* Allow 10 seconds for command. */
    if (ioctl(fd, SIOC_IO, &sctl_io) < 0) /* Request was invalid. */
        printf("Error: request was invalid!
");
        return(-1);
    else if (sctl_io.cdb_status == S_GOOD) /* Device is ready. */
        return(0);
    else /* Unknown state or device is not ready. */
        printf("Error: unknown state device is not ready!
");
        return(-1);
}

Figure 7: Sample code fragment to enable/disable read caching of a disk drive.
The I/O library function call lseek() of the C programming language cannot be used to move the disk arms directly because it only updates an internal data structure. Only the next call to read() will actually execute both the seek and the read operation. However, timing this call to read() will also include rotational latency delays that will bias the measured seek time. Fortunately, the SCSI-2 Standard [ANS94] specifies an explicit seek operation. Through the low-level SCSI interface described in Section 4, we were able to force the disks to perform seek operations without reading any data. With this command, the disk operation is finished as soon as the heads reach the target track.

The test strategy to measure the seek profiles of the magnetic disks in System 1 and System 2 consisted of a loop that executed the following four steps during each iteration:

1. Seek to logical block address (LBA) zero.
2. Get the initial time.
3. Seek to LBA \( X \).
4. Calculate the total seek time by subtracting the initial time from the current time.

The value \( X \) in step 3 was varied from 0% to 100% of the disk capacity in 1% increments. For every distinct value of \( X \), the four steps were repeated 20 times to get an average value. Figure 8 shows the seek profiles of the disks in System 1 and System 2.

We can make the following observations from the seek profiles. The seek times are shown as a function of the number of logical blocks traversed (or the percentage of disk capacity). While this is related to the number of physical cylinders, we could not access the physical cylinders directly, except for the Seagate ST31200W disk (see the notch page discussion in Section 5.2). Since the seek parameter is a number of logical blocks, disks that are zoned produce different seek profiles when seeking inward as compared to outward (see Figure 8e). This is because a zone with a higher density will contain fewer cylinders for a given number of logical blocks when compared to a low density zone. A seek traversing a higher density
Figure 8a: Quantum PD425S disk on System 1 (Fast SCSI interface).

Figure 8b: Hewlett-Packard C2247-300 disk on System 1 (Fast SCSI interface).

Figure 8c: Hewlett-Packard C2247 disk on System 2 (Fast & Wide SCSI interface).

Figure 8d: Seagate ST31200W disk on System 2 (Fast & Wide SCSI interface).

Figure 8e: Seeking inward vs. outward with Seagate ST31200W disk.

Figure 8: Seek profiles for magnetic disk drives from different vendors. All seeks were initiated from logical block address zero.
zone will involve a smaller distance, reducing the service time of the operation. Seek graphs reported by manufacturers are often in terms of cylinders. In that case, the seek curves in both directions are identical. However, the number of cylinders per zone is not always provided by the disk manufacturers. (Section 5.2 describes a technique to compute zoning information by conducting experiments.)

Figures 8b-d show the seek profile of current, widely available 1 GByte disk drives from Seagate and Hewlett-Packard. Their maximum seek time is roughly 20 msec. The slightly older Quantum 412 MByte disk of Figure 8a executes a full-stroke seek in about 26 msec. The seek profiles as shown in these figures can be used as system design parameters to fine tune the real-time scheduling of disk I/Os. During our experiments, both test systems were in single-user mode. If the machines operate in normal, multi-user mode, system activities such as network I/Os can cause delays due to interrupt processing or task switches. These artifacts must be taken into account when designing a system.

5.2 Measuring the Transfer Rate

The different zones of a disk present differing data transfer rates to the system. We wanted to characterize the disk based on its number of zones, and the transfer rate, storage capacity, and boundaries of each zone. Our experiments undertook to measure these disk parameters.

The fundamental assumption was that differences in observed transfer rates from the disk would reveal the desired zoning information. In these experiments, we employed the raw interface to the disk to bypass HP-UX kernel buffering. There was no need to use the low-level SCSI operations (with the exception of the command to enable/disable the disk-controller read cache) as described in Section 4.2 for these experiments; the standard C library functions \texttt{read()} and \texttt{lseek()} were used.

The experiments were conducted on the disks on our two platforms by performing an initial \texttt{lseek()} to logical block zero, and then sequentially reading a fixed number of bytes from the disk, while timing each \texttt{read()}. The sequential reads were continued with no intervening \texttt{lseek()}s until the end of the disk
was reached. This entire process was repeated 10 times for each fixed read size. The read size was varied between iterations; sizes of 128K, 512K, and 1 MBytes were used. Read sizes were selected to be sufficiently large so as to negate the effect of the disk-controller cache. The program stored the start and end times for each read; only after all the reads for each size iteration were completed did the program calculate the read times. We used this strategy in an effort to minimize the impact of additional processing during the sequential reads. We computed the maximum, minimum, and average of the measured times.

The transfer rate experiments were performed using HP-UX in a single user mode. The experiments were duplicated with the disk-controller read-cache enabled and disabled so that we might understand the possible effects due to caching operations.

Figure 9a: Quantum PD425S disk on System 1 (Fast SCSI interface).

Figure 9b: Hewlett-Packard C2247-300 disk on System 1 (Fast SCSI interface).

Figure 9c: Hewlett-Packard C2247 disk on System 2 (Fast & Wide SCSI interface).

Figure 9d: Seagate ST31200W disk on System 2 (Fast & Wide SCSI interface).

Figure 9: Transfer rate profiles for magnetic disks from different vendors. The read cache was on in all cases.
The transfer rate profiles are shown in Figure 9, while zone information from our experiments is tabulated in Tables 2-5. The presence of zones, manifested as horizontal segments with differing transfer rates, may be seen in the transfer rate profiles. The number of zones may be determined by the number of horizontal segments in the plot. The actual transfer rates themselves can be estimated from these plots, but may also be determined directly from the data files.

It is of interest to note that these plots are for a read size of 1 MBytes, as this read size produces the cleanest plots for the determination of the number of zones and their sizes. However, the plots for the smaller read sizes showed slightly higher transfer rates as well as an increased scatter in the values of the transfer rates. This is because within a given zone, a large read size will incur approximately the same number of track and head switches during its retrieval, whereas the retrieval of a smaller read block may show a larger variance in these values because it may sometimes reside entirely within a given track and other times on a cylinder boundary.

In our experiments, the HP C2247 unexpectedly produced differing transfer rates with cache on and off, while the transfer rates of the Quantum PD42S5 and Seagate ST31200W were unaffected by cache status. From our experiments with cache control (see Section 5.3), we feel that there is some unknown (to us) interaction in the HP C2247 between the status of cache enable/disable and other cache-related disk operations such as read-ahead/prefetch or zero-latency read that changes the observed transfer rate.

The manufacturer of the HP C2247 publishes the numbers of zones for its device. The computed

<table>
<thead>
<tr>
<th>Zone #</th>
<th>Size (MB)</th>
<th>Transfer Rate (MBps)</th>
<th>Start (MB)</th>
<th>End (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>64.0</td>
<td>2.02</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>61.0</td>
<td>1.90</td>
<td>64</td>
<td>125</td>
</tr>
<tr>
<td>2</td>
<td>57.0</td>
<td>1.82</td>
<td>125</td>
<td>182</td>
</tr>
<tr>
<td>3</td>
<td>53.0</td>
<td>1.66</td>
<td>182</td>
<td>235</td>
</tr>
<tr>
<td>4</td>
<td>49.0</td>
<td>1.52</td>
<td>235</td>
<td>284</td>
</tr>
<tr>
<td>5</td>
<td>45.0</td>
<td>1.38</td>
<td>284</td>
<td>329</td>
</tr>
<tr>
<td>6</td>
<td>41.0</td>
<td>1.26</td>
<td>329</td>
<td>370</td>
</tr>
<tr>
<td>7</td>
<td>36.0</td>
<td>1.13</td>
<td>370</td>
<td>406</td>
</tr>
</tbody>
</table>

Table 2: Zone information of Quantum PD42S5 disk.
<table>
<thead>
<tr>
<th>Zone #</th>
<th>Size (MB)</th>
<th>Transfer Rate w/ Read Cache (MBps)</th>
<th>Transfer Rate w/o Read Cache (MBps)</th>
<th>Start (MB)</th>
<th>End (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32.4</td>
<td>3.80</td>
<td>2.88</td>
<td>0</td>
<td>32.3</td>
</tr>
<tr>
<td>1</td>
<td>112.0</td>
<td>3.66</td>
<td>2.78</td>
<td>324</td>
<td>435</td>
</tr>
<tr>
<td>2</td>
<td>76.0</td>
<td>3.49</td>
<td>2.69</td>
<td>436</td>
<td>511</td>
</tr>
<tr>
<td>3</td>
<td>77.0</td>
<td>3.33</td>
<td>2.60</td>
<td>512</td>
<td>588</td>
</tr>
<tr>
<td>4</td>
<td>71.0</td>
<td>3.15</td>
<td>2.49</td>
<td>589</td>
<td>659</td>
</tr>
<tr>
<td>5</td>
<td>145.0</td>
<td>2.84</td>
<td>2.30</td>
<td>660</td>
<td>804</td>
</tr>
<tr>
<td>6</td>
<td>109.0</td>
<td>2.52</td>
<td>2.08</td>
<td>805</td>
<td>913</td>
</tr>
<tr>
<td>7</td>
<td>89.0</td>
<td>2.19</td>
<td>1.85</td>
<td>914</td>
<td>1002</td>
</tr>
</tbody>
</table>

Table 3: Zone information of Hewlett-Packard C2247-300 disk.

<table>
<thead>
<tr>
<th>Zone #</th>
<th>Size (MB)</th>
<th>Transfer Rate w/ Read Cache (MBps)</th>
<th>Transfer Rate w/o Read Cache (MBps)</th>
<th>Start (MB)</th>
<th>End (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32.4</td>
<td>3.40 / 3.52</td>
<td>2.96 / 3.06</td>
<td>0</td>
<td>32.3</td>
</tr>
<tr>
<td>1</td>
<td>112.0</td>
<td>3.17 / 3.28</td>
<td>2.78 / 2.89</td>
<td>324</td>
<td>435</td>
</tr>
<tr>
<td>2</td>
<td>76.0</td>
<td>3.04 / 3.13</td>
<td>2.69 / 2.78</td>
<td>436</td>
<td>511</td>
</tr>
<tr>
<td>3</td>
<td>77.0</td>
<td>2.92 / 3.02</td>
<td>2.60 / 2.68</td>
<td>512</td>
<td>588</td>
</tr>
<tr>
<td>4</td>
<td>71.0</td>
<td>2.78 / 2.87</td>
<td>2.49 / 2.56</td>
<td>589</td>
<td>659</td>
</tr>
<tr>
<td>5</td>
<td>145.0</td>
<td>2.54 / 2.61</td>
<td>2.28 / 2.34</td>
<td>660</td>
<td>804</td>
</tr>
<tr>
<td>6</td>
<td>109.0</td>
<td>2.27</td>
<td>2.07</td>
<td>805</td>
<td>913</td>
</tr>
<tr>
<td>7</td>
<td>89.0</td>
<td>2.02</td>
<td>1.84 / 1.88</td>
<td>914</td>
<td>1002</td>
</tr>
</tbody>
</table>

Table 4: Zone information of Hewlett-Packard C2247 (Fast & Wide) disk.

<table>
<thead>
<tr>
<th>Zone #</th>
<th>Size (MB)</th>
<th>Transfer Rate (MBps)</th>
<th>Start (MB)</th>
<th>End (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14.0</td>
<td>4.17 / 4.22</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>1</td>
<td>67.0</td>
<td>4.08</td>
<td>140</td>
<td>206</td>
</tr>
<tr>
<td>2</td>
<td>45.0</td>
<td>4.02</td>
<td>207</td>
<td>251</td>
</tr>
<tr>
<td>3</td>
<td>46.0</td>
<td>3.97</td>
<td>252</td>
<td>297</td>
</tr>
<tr>
<td>4</td>
<td>44.0</td>
<td>3.88</td>
<td>298</td>
<td>341</td>
</tr>
<tr>
<td>5</td>
<td>42.0</td>
<td>3.83</td>
<td>342</td>
<td>383</td>
</tr>
<tr>
<td>6</td>
<td>41.0</td>
<td>3.74</td>
<td>384</td>
<td>424</td>
</tr>
<tr>
<td>7</td>
<td>96.0</td>
<td>3.60</td>
<td>425</td>
<td>520</td>
</tr>
<tr>
<td>8</td>
<td>37.0</td>
<td>3.50</td>
<td>521</td>
<td>557</td>
</tr>
<tr>
<td>9</td>
<td>52.0</td>
<td>3.36</td>
<td>558</td>
<td>609</td>
</tr>
<tr>
<td>10</td>
<td>35.0</td>
<td>3.31</td>
<td>560</td>
<td>644</td>
</tr>
<tr>
<td>11</td>
<td>34.0</td>
<td>3.22</td>
<td>645</td>
<td>678</td>
</tr>
<tr>
<td>12</td>
<td>46.0</td>
<td>3.13</td>
<td>679</td>
<td>724</td>
</tr>
<tr>
<td>13</td>
<td>32.0</td>
<td>3.07</td>
<td>725</td>
<td>756</td>
</tr>
<tr>
<td>14</td>
<td>31.0</td>
<td>2.99</td>
<td>757</td>
<td>787</td>
</tr>
<tr>
<td>15</td>
<td>42.0</td>
<td>2.85</td>
<td>788</td>
<td>829</td>
</tr>
<tr>
<td>16</td>
<td>55.0</td>
<td>2.76</td>
<td>830</td>
<td>884</td>
</tr>
<tr>
<td>17</td>
<td>37.0</td>
<td>2.62</td>
<td>885</td>
<td>921</td>
</tr>
<tr>
<td>18</td>
<td>25.0</td>
<td>2.53</td>
<td>922</td>
<td>946</td>
</tr>
<tr>
<td>19</td>
<td>34.0</td>
<td>2.43</td>
<td>947</td>
<td>980</td>
</tr>
<tr>
<td>20</td>
<td>20.0</td>
<td>2.33</td>
<td>981</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 5: Zone information of Seagate ST31200W (Fast & Wide) disk.
number of zones using experiments matched those provided by the manufacturer with 100% accuracy. Our measured transfer rates for Seagate’s and HP’s transfer rate are not identical to those advertised by the manufacturers. This is because the advertised numbers correspond to raw data transfer from the disk medium and include all bits read from the disk surface (including the address bits and the ECC bits for error correction). However, our measurements are based on the data bits read and include the overhead of SCSI adapter, SCSI bus transfer, disk controller, disk seeks, and head switches.

For the Seagate ST31200W disk model we were able to verify our experimental results about zones and transfer rates with accurate configuration data that was obtained directly from the devices. The ST31200W disk stores configuration data about notches that can be queried through a SCSI MODE SENSE command. A notch is the SCSI terminology for a zone. This information includes: 1) the number of platters, 2) the number of zones, and 3) both the number of cylinders and the number of logical blocks per zone. It is obtained in a similar manner as the cache information (see Section 4.2). Using this information, we can develop analytical models to compute the service time of each zone (see Table 6).

We can make the following observations when comparing Tables 5 and 6:

- We failed to identify two zones (i.e., zone 8 of Table 6 was believed to be part of zone 7 in Table 5 and zone 18 of Table 6 was believed to be part of zone 16 in Table 5) because they differ only in one block per track and, therefore, their transfer rate is almost the same.
- We measured the rest of the zone boundaries very accurately (see Figure 10).
- The measured transfer rate is consistently about 10% lower than calculated. This may be due to the arbitration (i.e., reconnect) overhead of the SCSI-2 bus protocol.
- Table 6 enables us to devise a mapping from the logical blocks to the physical cylinders. This information could be used to adjust the seek profile such that it is identical for inward and outward head movements.

5.3 On-Device Cache Operation

In order to support our transfer rate experiments with the disk-controller cache enabled as well as disabled, we undertook to discover the methods by which we could send commands to the underlying SCSI hardware.
Table 6: Zoning information of a Seagate ST31200W disk, obtained by reading the notch parameter page.
Since the Unix operating system provides a device-independent abstraction of files for accessing devices, it was necessary for us to circumvent some standard Unix facilities as detailed in Section 4.

The experiments were designed to determine if we could control the operation of the disk-controller cache by observing the behavior of the transfer rate over a small portion of the disk. We expected to see a much higher transfer rate (essentially that of the SCSI bus) when read requests could be satisfied using the disk-controller cache instead of the disk medium.

![Figure 11a: Quantum PD425S disk on System 1 (Fast SCSI interface).](image)

![Figure 11b: Hewlett-Packard C2247-300 disk on System 1 (Fast SCSI interface).](image)

![Figure 11c: Hewlett-Packard C2247 disk on System 2 (Fast & Wide SCSI interface).](image)

![Figure 11d: Seagate ST31200W disk on System 2 (Fast & Wide SCSI interface).](image)

![Figure 11: Effect of read caching for magnetic disks from different vendors.](image)

The design of the experiment was to perform a timed read from the raw character device, thereby eliminating the influence of the kernel buffer pool. Starting at 1 KBytes, increasingly larger reads were performed in increments of 1 KBytes up to a maximum of 250 KBytes. The reads were performed from
block address zero, with 10 iterations recorded for each read size. So that we might experience disk-controller read cache hits when the read cache was enabled, the experiment first performed a read that was not included in the data set times. Minima, maxima, and averages were calculated for each set of ten iterations and plotted in Figure 11. These tests were duplicated with the disk-controller read cache enabled and disabled.

The influence of the disk-controller read cache is clearly evident in the figures, showing as the segment of the line with a small but non-zero slope starting from the origin. This slope defines the maximum rate of data transfer per disk. These calculated values are in Table 7, compared with the corresponding maximum SCSI bus transfer rate.

Also evident in these figures is the point at which the cache ceases to be of value, as indicated by the sharp rise in the time required to satisfy a request. The value of the horizontal axis at this point is the size of the read cache allocated to our process (disk controllers can segment the cache to provide separate cache memory for separate processes). While we did perform tests with the read cache off as well, we have presented only those with the cache on, since they showed more interesting behavior.

Several other features in Figures 11a-d are of interest. The most obvious is the overall shape of the plot, consisting of a step function of the time required to read a fixed amount of data. It is curious that the difference in time between one plateau and another is equal to the rotational period of the disk. While we have no clear explanation of this, the correspondence is so close that it should be attributed to some phenomenon directly related to the rotation of the disk. Using these time differences, we calculated the

<table>
<thead>
<tr>
<th>Disk</th>
<th>Measured Transfer Rate (MBytes/sec)</th>
<th>Maximum SCSI Transfer Rate (MBytes/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seagate ST31200W</td>
<td>12.2</td>
<td>20</td>
</tr>
<tr>
<td>HP C2247</td>
<td>14.4</td>
<td>20</td>
</tr>
<tr>
<td>HP C2247-300</td>
<td>3.9</td>
<td>10</td>
</tr>
<tr>
<td>Quantum PD425S</td>
<td>3.8</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 7: Maximum Observed SCSI Transfer Rates
rotational speed of the disk. These calculated rotational speeds compared to the manufacturer’s values are in Table 8. Using the Seagate disk, we attempted to disable some of the features we thought may influence the disk behavior, such as prefetch/readahead. Our desired configuration is one in which both prefetch and cache are disabled; however, our attempts to produce this state always produced the same results as if the cache were enabled.

The presence of a handful of peaks in Figure 11 is not reproducible from one run to another. They are a consequence of system activity occurring during a timed event and are not an artifact of the disk hardware. System 2, which was operated in a standalone mode, experienced far fewer of these peaks than did System 1, which remained physically connected to a network. Since System 2 is also faster than System 1, it may be the case that System 2 can process other system activity with little impact on our results.

### 5.4 Measurement of SCSI-2 Scalability

We conducted experiments to quantify the scalability characteristics of the SCSI-2 bus. In these experiments, we varied the number of disks (Seagate ST31200W) from one to eleven and the size of a read request from 128 KBytes to 5 MBytes. The requests were performed entirely within one of two zones, either the outermost zone (highest transfer rate) or the innermost zone (lowest transfer rate).

The tests were performed by having a master Unix process fork a child process for each disk request. The child processes performed all necessary initialization and then waited for a synchronization signal from the master. Upon receipt of the signal, each child issues twenty read requests to its assigned disk,

<table>
<thead>
<tr>
<th>Disk</th>
<th>RPM (estimated)</th>
<th>RPM (Mfg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seagate ST31200W</td>
<td>5413</td>
<td>5411</td>
</tr>
<tr>
<td>HP C2247</td>
<td>5400</td>
<td>5400</td>
</tr>
<tr>
<td>Quantum PD425S</td>
<td>3608</td>
<td>3600</td>
</tr>
</tbody>
</table>

Table 8: Observed vs. Manufacturer’s Specified Disk Rotational Speeds
one after another. After completion of the reads, the child process averaged the measured service times, converted them to transfer rates, and then wrote them to an output file. All tests were performed with the disk-controller read cache enabled.

<table>
<thead>
<tr>
<th>Transfer Rate [MBytes/sec]</th>
<th>Disks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1  2  3  4  5  6  7  8  9  10 11</td>
</tr>
<tr>
<td>Innermost Zone</td>
<td>2.23 4.47 6.70 8.93 11.17 13.35 14.95 15.54 15.53 15.49 15.55</td>
</tr>
<tr>
<td>Outermost Zone</td>
<td>4.01 8.00 11.93 14.89 16.12 16.96 17.29 17.64 17.36 16.97 16.92</td>
</tr>
</tbody>
</table>

Table 9: Total SCSI-2 bus transfer rate.

Table 9 presents our results for 1 MByte read requests issued to both the innermost and outermost zones for varying numbers of disks. In the innermost zone, transfer rate increases linearly up to seven disks, and then starts to flatten out as the bus becomes a bottleneck. We observed similar behavior for read requests to the outermost zone, except that the linear region extends to only four disks.

Figure 12: Standard deviation of transfer times vs. number of disks.

Figure 12 depicts the increase in the standard deviation of the read time as the number of disks is increased. This deviation is dependent on the number of disks, the zone that a block is read from, and the priority of the SCSI disk (as determined by its SCSI id). In the outermost zone, when more than four simultaneous read requests issued, the variation in the measured service times becomes too large to model accurately. Reads from the innermost zones on all disks showed small amounts of standard deviation up
to six disks. This should come as no surprise because the bandwidth of the bus is fixed, and the innermost zone has a lower transfer rate than the outermost one.

These experiments demonstrate that modelling the service time of a SCSI-2 disk is difficult when the SCSI-2 bus becomes saturated.

6 Methodology to Estimate the Service Time

In order to verify our methods for characterizing disk properties, we combined our results obtained from the seek test, the transfer rate test, and the cache experiments into a model to predict the service time for specified read requests. Our model can estimate the service time of the disk subsystem accurately as long as the maximum number of active disks does not exceed four. We employed this model to compare measured service time to predicted service time. Our model is designed to never underestimate the actual service time. For the multimedia example (Figure 4), the time period required to read blocks from the disks would never exceed the estimates using our model.

Figure 13: Comparison of measured vs. modeled seek profile for the Hewlett-Packard C2247 (wide) disk drive.
6.1 Seek Time

The model was produced by fitting a curve to the seek profile obtained for the inward direction. The curve was broken into two parts, joined at 4% of disk capacity. The 0-4% range was simulated with seek time being a square root function of the seek distance, while the range from 4% to 100% was represented as seek time being a linear function of seek distance (see [RW94]). Constants were chosen to produce the following seek time equations as well as the curve seen in Figure 13:

\[
   t = \begin{cases} 
   1.5 + 2.65 \times \sqrt{S} & \text{0%-4% Capacity} \\
   6.3 + 0.127 \times S & \text{4%-100% Capacity} 
   \end{cases}
\]

In the above equations, \( t \) is the seek time in milliseconds, and \( S \) is the seek distance expressed in percentage of disk capacity.

6.2 Rotational Latency

Added to the model was the calculated value for the rotational latency of the disk. Even though the rotational latency will often be less than this value (one-half of the rotational latency on the average), choosing the full value means we will never underestimate this variable latency.

6.3 Transfer Rate

The transfer rates for the observed zones on the disk were taken from one of the corresponding Tables 2, 3, 4, or 5. For example, for the HP C2247 (Fast & Wide) disk drive, we used column 3 in Table 4. Before adding the transfer rates to the model, we derated them by 5%. Again, this derating was performed to ensure that we would never underestimate the disk service time.

6.4 SCSI-2 Scalability

We used the results from Section 5.4 to limit the number of disks for which our model is valid. Based on the results obtained reading in the outermost (highest transfer rate) zone, we limit the applicability of our
Table 10: Comparison of measured vs. predicted times for starting in Zone 0 and reading in Zone 0

<table>
<thead>
<tr>
<th>Read Size</th>
<th>Measured Time (msec)</th>
<th>Predicted Time (msec)</th>
<th>Error (percent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64K</td>
<td>22.7</td>
<td>34.8</td>
<td>53</td>
</tr>
<tr>
<td>128K</td>
<td>56.3</td>
<td>57.0</td>
<td>1.2</td>
</tr>
<tr>
<td>256K</td>
<td>89.5</td>
<td>101.5</td>
<td>13</td>
</tr>
<tr>
<td>512K</td>
<td>180.7</td>
<td>190.4</td>
<td>5</td>
</tr>
<tr>
<td>1024K</td>
<td>356.8</td>
<td>368.2</td>
<td>3</td>
</tr>
</tbody>
</table>

model to four disks. While up to six disks can be supported if all disks are reading from the innermost (slowest transfer rate) zone, our intended applications must use a constant number of disks. This limitation requires that we choose the minimum number of disks that can be supported at all times.

6.5 Measurements

Since we wanted to simulate the conditions under which a disk would be used, the following procedure was employed to obtain the measured service time:

1. Seek to a specified LBA (logical block address) on the disk using the low-level SCSI commands. This physically positions the disk arms.

2. Perform an `lseek()` to the start LBA of the desired data. No disk arm movement results, but the internal file pointer is updated and disk arm movement will result upon issuing a `read()` command.

3. Obtain the start time.

4. Perform a `read()` for the desired data. The disk arms will then seek to the desired location, the platter will rotate until the desired data is under the disk heads, and finally the data will be read.

5. Compute the elapsed time.

This procedure was embedded within a loop and performed 20 times for each set of parameters. These parameters consisted of the initial arm position, the start point of the read, and the number of bytes to read.

The results of both the measured and predicted times, as well as relative errors, are presented in Tables 10 and 11 and in Figure 14. From Figure 12 we can see that, for up to four disks, the service times
Table 11: Comparison of measured vs. predicted times for starting in Zone 0 and reading in Zone 7.

<table>
<thead>
<tr>
<th>Read Size</th>
<th>Measured Time (msec)</th>
<th>Predicted Time (msec)</th>
<th>Error (percent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64K</td>
<td>34.2</td>
<td>64.7</td>
<td>89</td>
</tr>
<tr>
<td>128K</td>
<td>78.2</td>
<td>100.5</td>
<td>28</td>
</tr>
<tr>
<td>256K</td>
<td>145.4</td>
<td>172.0</td>
<td>18</td>
</tr>
<tr>
<td>512K</td>
<td>294.4</td>
<td>315.0</td>
<td>6</td>
</tr>
<tr>
<td>1024K</td>
<td>565.6</td>
<td>601.0</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 14: Relative errors (i.e., over-estimations) of modeled vs. measured service times for the Seagate ST31200W disk drive.

for multiple disks are the same as the service times for a single disk. As can be seen from Tables 10 and 11, we have a model that does not underestimate the measured service time of a disk, yet provides a reasonable approximation to the measured service time. A more comprehensive comparison was done for the Seagate ST31200W disk and the results are shown in Figure 14. It is evident that the estimates in Tables 10 and 11 and Figure 14 are worse for the smaller read sizes. This is because a small read request has a short transfer time, and our estimate is influenced by our choice of including one full rotational period in our model.
7 Conclusions and Future Research Directions

In this study we used a low-level SCSI programming interface to identify physical attributes of off-the-shelf magnetic disk drives using HP-UX based workstations. This was possible without access to proprietary information. We were able to gather information about the seek characteristic, the zoning, and the on-device cache operation for several different SCSI-2 disks. The experimental results were used to outline a methodology to estimate the service time of a disk read request. This methodology can be employed in multimedia applications in order to best utilize magnetic disk drives while guaranteeing a continuous media display.

Our experimental techniques yield a model that predicts the service time of a single disk quite well. The main limitation of extending our model to multiple disks is the uncertainty involved in the scalability and contention involved using the SCSI-2 bus. If we were to extend our model past four disks without underestimating the disk service time, we would have had to grossly overestimate the disk service time. This would waste the bandwidth of the SCSI-2 disk subsystem.

This study further demonstrates that a programmer can have fine-grained control over SCSI devices without having to abandon a convenient, off-the-shelf programming environment. However, it also shows the limitations of such an approach. The multi-user, multi-tasking nature of Unix introduces infrequent but hard to control delays in the disk service time. The system designer will have to evaluate these trade-offs and weigh the convenience of using standard hardware and software against the stringency of high disk utilization and low hiccup probability of the target application. If the highest possible number of simultaneous displays and absolute hiccup-free retrieval are required, then special-purpose hardware and software must be employed. If, on the other hand, a lower number of displays can be tolerated (by using the service time estimation techniques outlined in this paper) then the use of off-the-shelf tools can greatly reduce the system development effort. Moreover, if there are only a few hardware and operating system dependent routines, an application port to a different platform will be simplified.
Also, while the experiments that we conducted for this study showed promising results, they do not have the complexity of a real application. The implementation of a system that supports the display of continuous media based on the ideas presented in this study will therefore be another future research direction.

References


