Principles of Delay-Sensitive Multimedia Data Storage and Retrieval

JIM GEMMELL
Simon Fraser University
and
STAVROS CHRISTODOULAKIS
University of Waterloo

This paper establishes some fundamental principles for the retrieval and storage of delay-sensitive multimedia data. Delay-sensitive data include digital audio, animations, and video. Retrieval of these data types from secondary storage has to satisfy certain time constraints in order to be acceptable to the user. The presentation is based on digital audio in order to provide intuition to the reader, although the results are applicable to all delay-sensitive data. A theoretical framework is developed for the real-time requirements of digital audio playback. We show how to describe these requirements in terms of the consumption rate of the audio data and the nature of the data-retrieval rate from secondary storage. Making use of this framework, bounds are derived for buffer space requirements for certain common retrieval scenarios. Storage placement strategies for multichannel synchronized data are then categorized and examined. The results presented in this paper are basic to any playback of delay-sensitive data and should assist the multimedia system designer in estimating hardware requirements and in evaluating possible design choices.

Categories and Subject Descriptors: H.3.2 [**Information Storage and Retrieval**]: Information Storage; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*retrieval models*; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*animations, audio input/output, video* (*e.g., tape, disk, DVI*)

General Terms: Design, Performance

Additional Key Words and Phrases: Continuous media, delay-sensitive, real time

1. INTRODUCTION

Data is said to be delay sensitive when there are real-time deadlines for the presentation of successive units of the data to the user. Examples of data that would be classified as delay sensitive are digital audio, animations, and video. For digital audio to correctly reproduce sound, digital audio samples

Authors' addresses: J. Gemmell, Department of Computer Science, Simon Fraser University, Burnaby, British Columbia, V5A 1S6, Canada; S. Christodoulakis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada.

must be delivered to a digital-to-analog converter at a precise rate. In the cases of animation and video, real-time deadlines must be met for the display of successive images for motion to flow smoothly, without any pauses or interruptions.

This paper establishes principles of storage and retrieval of delay-sensitive multimedia data. In order to provide greater intuition to the reader, we discuss the retrieval requirements in terms of digital audio playback. However, the results given could equally apply to any similar delay-sensitive data, because the issues involved in digital audio are a superset of the issues involved with most other delay-sensitive data.

Very little work has been done in this area to date. Wells et al. and Yu et al. [12, 15] have examined a very uncommon scenario in which an optical disk is employed with extremely limited buffering. Anderson et al. [3, 4] have given a high-level description of a distributed system that includes delay-sensitive data, without describing in detail how real-time requirements for data retrieval are met. Abbot [1] has examined a storage placement strategy that yields a high probability of improved system throughput, but does not absolutely guarantee meeting real-time constraints. Park and English [7] give a strategy for using lower-quality, lower-bandwidth audio data when contention for bandwidth occurs. However, their method does not provide a guaranteed minimum quality.

In this paper we first develop a theoretical framework for describing the real-time requirements of retrieval and then consider storage placement strategies in the light of those results. Section 2 reviews the basics of digital audio. Sections 3–5 deal with a dedicated storage device: Sections 3 and 4 relate to single-channel (i.e., monophonic) playback, and Section 5, to multiple-channel playback. Section 6 then examines some approaches to a shared storage device, such as would be found in a multitasking or distributed system. Finally, Section 7 categorizes the possible storage placement strategies for synchronized multichannel playback and discusses their relative merits.

2. REVIEW OF DIGITAL AUDIO

Digital audio is the encoding of an audio signal as a series of symbols (numbers) that can be processed by a computer [11]. Much attention has been given to digital audio, especially since the advent of the compact disk, an optical disk that is used for storing digital audio recordings [2, 14]. Digital audio has been made more practical in recent years by mass storage technology that can handle the large amounts of data that may be involved in digital audio [10], and by the mass production of digital-to-analog and analog-to-digital conversion hardware.

At the same time, also in part due to improved mass storage technology, research and development have begun in the area of multimedia databases, which extend traditional text-only databases to include images, animations, video, and sound [5]. Electronic mail, once strictly text only, has been extended to include sound and images in such systems as NeXT Computing's workstation [9]. With powerful graphics workstations and mass storage devices becoming ever more cheap and powerful, multimedia will become the dominant form of man–machine communication [8].

Digitization of audio data is most commonly achieved by sampling an analog signal at fixed intervals and by storing the amplitude of each sample in binary form [6, 11]. A digital audio record therefore consists of a number of samples. Playback consists of feeding these samples into a digital-to-analog converter at the same rate at which they were sampled. We refer to this as the *consumption rate* of the playback system. The digital-to-analog converter then creates a signal that closely approximates the original. If samples are not delivered to the digital-to-analog converter at precisely the rate at which sampling occurred, the reproduced sound will not match the sampled sound. Systems may be designed in which samples come off the disk at this precise rate, but in most cases retrieval will be more irregular. When the retrieval rate is irregular, samples must first be buffered and then passed to the converter at the desired rate.

(Note that the term *read* is used in this paper to refer to retrieving data from the storage device, while the term *consume* is used to refer to data being consumed by the digital-to-analog converter.)

If we compare the real-time demands of digital audio playback with the real-time demands of most database systems, a significant difference is observed. In most database systems, the real-time demand consists of completing the retrieval of a requested record before a given deadline. The real-time demand of an audio system is that the retrieval of successive samples of the audio record meet successive deadlines. The time that it takes to retrieve the first sample (or set of samples for multichannel playback) is important, because it essentially is the time from the request for playback until some sound may be heard. However, it is not a real-time deadline, as a correct playback does not depend on it.

3. PRINCIPLES OF DELAY-SENSITIVE DATA RETRIEVAL

In this section we develop a general theoretical framework for studying the performance of any playback system with a dedicated storage device. This framework allows us to determine if the system can meet the real-time deadlines of playback, given a distribution of the data production rate from a secondary storage device. This production rate depends on the device characteristics, the data placement on the device, the compression rate achieved at any point in the playback stream (if employed), in addition to the number of samples recorded per second, and the granularity of the sample (number of distinct values that can be recorded). Therefore, the general framework developed in this section provides a set of tools for the multimedia database designer to study the effect of the above design and hardware selection decisions on the performance of the system.

We assume throughout this paper that data is read from the storage device in the order the data is consumed. Abbot [1] has explored the possibility of

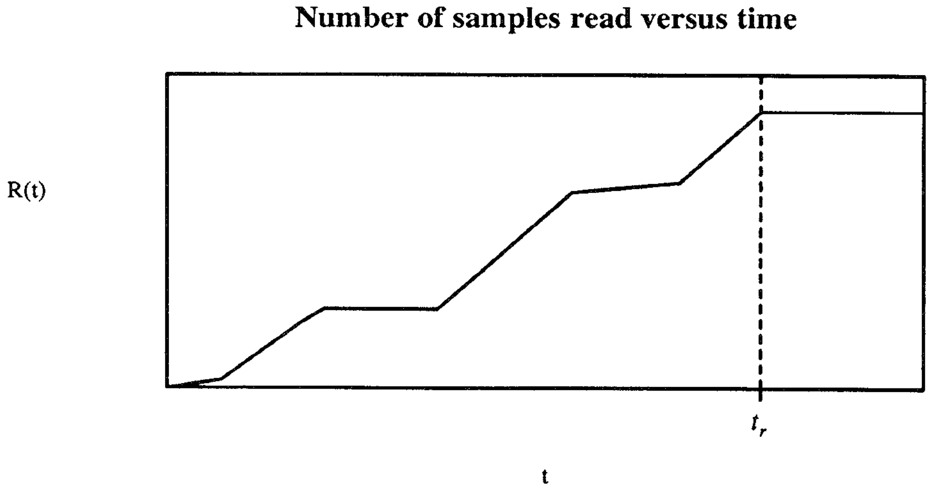

Fig. 1. $R(t)$: The reading function.

reading audio data out of order to reduce rotational delays on a disk. However, in general, reading out of order does not yield any such gain. In fact, it can be shown that, if the consumption order is known and the physical placement of the data may be specified, then reading in order will be optimal in terms of buffer requirements and throughput.

3.1 Minimizing Buffer Utilization and Start Time

In this section the basic requirements for the successful playback of a digital audio record from a dedicated storage device are explained. In addition, we analyze the minimal buffer requirements and the minimal start time for successful playback.

Let the total number of samples read up to and including time t be $R(t)$ (see Figure 1). The time chosen to be time zero should be chosen so that no samples have been read prior to that time; that is, for $t < 0$, $R(t) = 0$. Let the time that reading is completed be t_r, that is, the earliest time at which for all $t > t_r$, $R(t) = R(t_r)$.

$R(t)$ is a strictly nondecreasing function, because the number of samples read can never decrease. However, the slope of $R(t)$ will be zero whenever no samples are being read, for example, during seeks or during any waits for buffers to become available. Note also that when compression schemes are used $R(t)$ remains the number of decompressed samples. Achieving high compression appears as an apparent increase in the transfer rate of the storage device and, hence, the slope of $R(t)$ at that point. For example, under a differential compression scheme the slope of $R(t)$ would be relatively low when actual samples are read, and would increase only when differences are read.

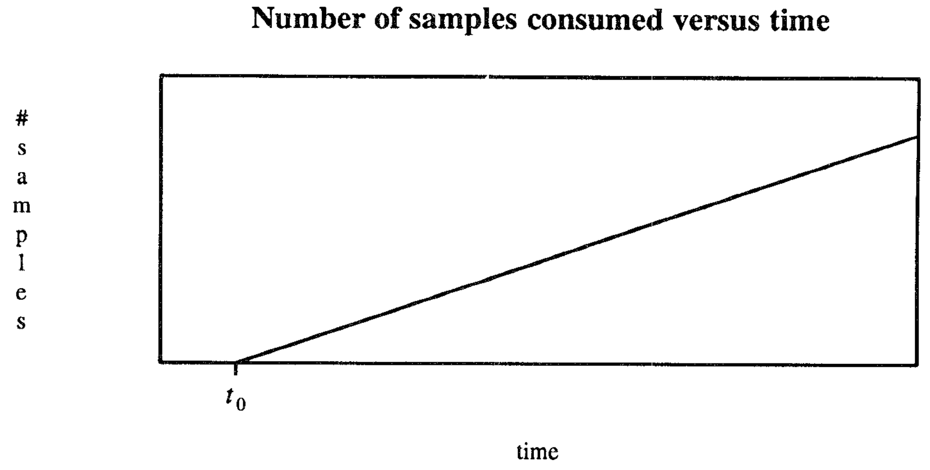

Fig. 2. $C(t)$: The consumption function.

Let the time that playback using the digital-to-analog converter begins be t_0. We refer to t_0 as the *start time.* Also, let the number of samples consumed by the digital-to-analog converter at time t be $C(t, t_0)$ (see Figure 2). If samples are consumed at r_c samples per second, then

$$C(t, t_0) = \begin{cases} 0 & \text{if} \quad t < t_0, \\ r_c(t - t_0) & \text{if} \quad t \geq t_0. \end{cases} \tag{1}$$

If the number of samples read is greater than the number consumed at any time, then the samples that have been read but not yet played back must be buffered. The number of samples that must be buffered at any given time is then $R(t) - C(t, t_0)$, which we denote as

$$B(t, t_0) = R(t) - C(t, t_0). \tag{2}$$

See Figure 3 for an example of the B, C, and R functions.

If buffer space must be allocated in advance, then enough space must be allocated for the maximum of $B(t, t_0)$. We refer to $B(t, t_0)$ as the *buffer utilization* at time t, and the peak value of $B(t, t_0)$ as the *maximum buffer utilization.*

If $B(t, t_0)$ is less than zero[1] in the range $t_0 \leq t \leq t_r$, this would mean that the digital-to-analog converter has consumed a sample that has not yet been read from the storage device, which is, of course, impossible. Otherwise,

[1] Strictly speaking, $B(t, t_0) = 0$ would also mean that something is being consumed that is not yet read. However, in this paper consumption is modeled as a continuous function, although in reality it occurs in discrete one-sample steps. Therefore, if the true discrete model would have rested at zero, our continuous model would have gone negative, and thus, we can use $B(t, t_0) < 0$ as the error condition.

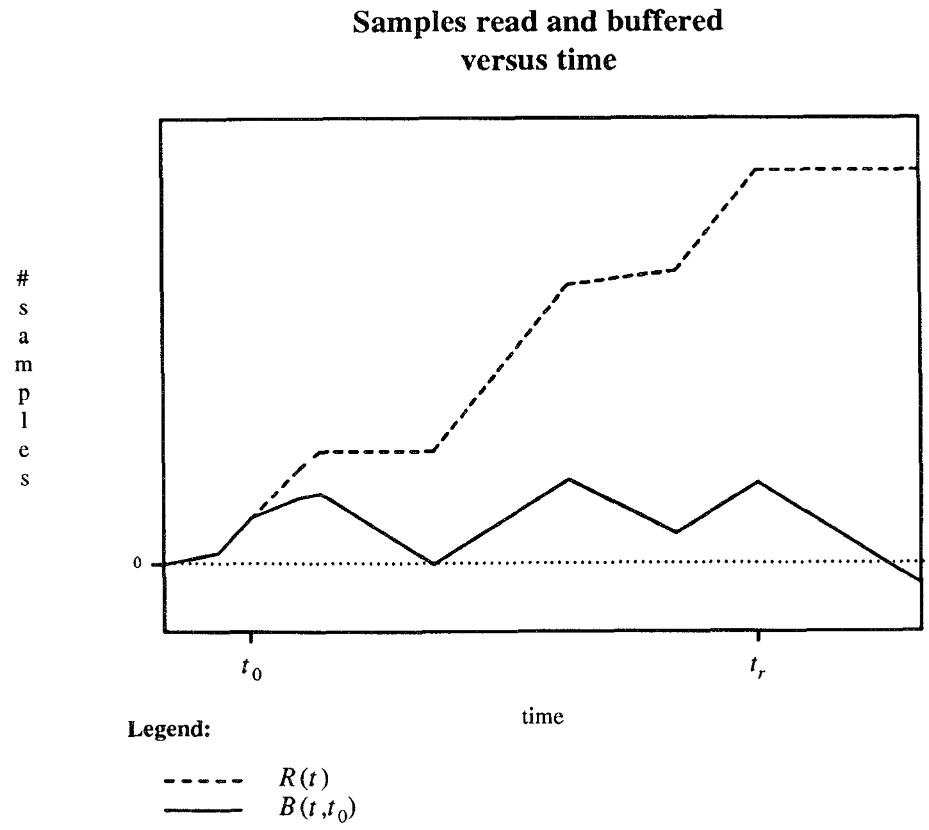

Fig. 3. $B(t, t_0)$: The buffer function.

$B(t, t_0)$ represents a workable playback and is referred to as a *playback solution*.

After the time t_r, any remaining samples will be consumed and $B(t, t_0)$ will reach zero eventually. The point at which $B(t, t_0)$ reaches zero after time t_r is the time at which playback is complete, because all samples have been read and consumed. Therefore, to find a workable playback, we must find a value of t_0 such that $B(t, t_0) \geq 0$ in the range $t_0 \leq t \leq t_r$. The buffer function is equal to the read function up to t_0, so it is equivalent to, say, $B(t, t_0) \geq 0$ for $0 \leq t \leq t_r$. Such a solution always exists, because if we set $t_0 = t_r$, we buffer all of the samples, which will always yield a solution. However, such a solution requires the maximum buffer space, as well as the longest start time. It is desirable to find a solution that minimizes both the buffer space and the start time. The following theorem demonstrates that solving for the minimum start time yields the minimum buffer space required:

THEOREM 1. *The solution using the minimum start time for playback also requires the least buffer space at any point in time.*

PROOF. Let the minimum value of t_0 giving a playback solution be t_w. Consider any other start time $y > t_w$. We know that the number of samples in the buffer at time t for playback beginning at t_w is

$$B(t, t_w) = \begin{cases} R(t) & \text{if} \quad t < t_w, \\ R(t) - r_c(t - t_w) & \text{if} \quad t \geq t_w \end{cases} \tag{3}$$

and that for playback beginning at y it will be

$$B(t, y) = \begin{cases} R(t) & \text{if} \quad t < y, \\ R(t) - r_c(t - y) & \text{if} \quad t \geq y. \end{cases} \tag{4}$$

Now consider any point in time t. If $t < t_w$, then $t < y$, because $y > t_w$. Therefore, both $B(t, y)$ and $B(t, t_w)$ have the value $R(t)$. If $t_w \leq t < y$, then $B(t, y) = R(t)$ and $B(t, t_w) = R(t) - r_c(t - t_w)$. Therefore, $B(t, y) \geq B(t, t_w)$. If $t > y$, then $B(t, y) = R(t) - r_c(t - y)$ and $B(t, t_w) = R(t) - r_c(t - t_w)$. Because $y > t_w$, once more, $B(t, y) \geq B(t, t_w)$. Therefore, at any time t, the buffer space required for playback beginning at time t_w is the same or less for playback beginning at time y. This means that beginning playback at time t_w is optimal in terms of buffer space. □

Now that we have demonstrated that the solution giving the minimum wait also gives the minimum buffer size, we will show how to find such a solution.

THEOREM 2. *The minimum value for t_0 may be found as follows (see Figure 4): First, consider $B(t, 0)$. If this function is nonnegative for $0 \leq t \leq t_r$, then $B(t, 0)$ is a solution, and the minimum value for t_0 is zero. Otherwise, let the minimum value for $B(t, 0)$, in the range $0 \leq t \leq t_r$, be at time t_{min} with $B(t_{min}, 0) = -m$. The intersection of $R(t)$ with $B(t, 0) + m$ is at the minimum value for t_0, yielding a workable playback.*

PROOF. If $B(t, 0)$ is a solution, then it must be optimal according to Theorem 1. Otherwise, let the minimum value for $B(t, 0)$, in the range $0 \leq t \leq t_r$, be at time $t_{\min}$ with $B(t_{\min}, 0) = -m$.

Suppose that a solution is to begin playback at time x. That is, $B(t, x)$ is at least zero for any time $0 < t < t_r$. If we compare $B(t, x)$ with $B(t, 0)$ in the range $x \leq t < t_r$, we see that

$$B(t, 0) = R(t) - r_c(t - 0) \tag{5}$$

and

$$B(t, x) = R(t) - r_c(t - x) = B(t, 0) + r_c x. \tag{6}$$

This shows us that, in the range $x \leq t < t_r$, $B(t, x)$ differs from $B(t, 0)$ by a constant, $r_c x$. That is, they have the same slope, which means they are parallel in this region.

Now suppose that the minimum value for $B(t, 0)$ occurs at a time greater than or equal to x; that is, $t_{\min} \geq x$. Because $B(t, x)$ is always at least zero for $0 < t < t_r$, we know that $B(t_{\min}, x) \geq 0$. Therefore, in the range $x \leq t < t_r$, the constant difference between the two equations is at least m if $t_{\min} \geq x$.

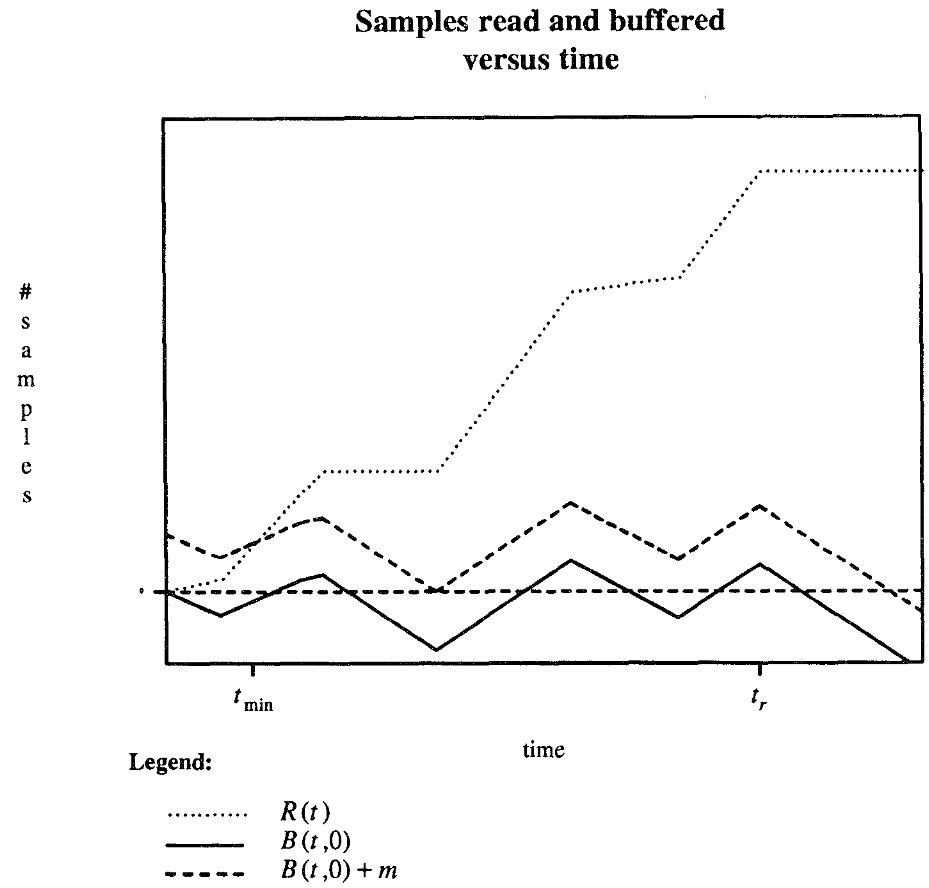

Fig. 4 Finding the minimal start time.

Now suppose that the minimum value for $B(t, 0)$ occurs at some value of $t_{\min} < x$. Because $R(t)$ is nondecreasing and nonnegative, it follows that $R(t_{\min})$ is at least 0. $B(t_{\min}, 0)$ is defined as being $-m$, so the functions differ by at least m at time $t_{\min}$. From $t_{\min}$ to x, $B(t, x)$ grows as $R(t)$ does, while $B(t, 0)$ grows only as fast as $R(t) - r_c t$. Therefore, the difference between $B(t, x)$ and $B(t, 0)$ remains at least m up to time x. We have already seen that there is a constant difference between the two equations in the range $x \leq t \leq t_r$. This difference must be at least m, because it is at least that value at time x.

Now consider the intersection of $R(t)$ with the function $B(t, 0) + m$. Let the intersection be at time t_i. $B(t, t_i)$ is a solution, because by definition it is the same as $R(t)$ up to time t_i, and after t_i it is greater than $B(t, 0)$ by m, ensuring that it has a value of at least zero. It remains to show that t_i is the minimum playback start time.

We have shown that there is a constant difference of at least m between any solution $B(t, x)$ and $B(t, 0)$ in the range $x \leq t \leq t_r$. $B(t, t_i)$ therefore has the least constant difference of any solution. Suppose there exists a solution $B(t, y)$ with $y < t_i$. By definition,

$$B(t, t_i) = \begin{cases} R(t) & \text{if} \quad t < t_i, \\ R(t) - r_c(t - t_i) & \text{if} \quad t \geq t_i, \end{cases} \tag{7}$$

and

$$B(t, y) = \begin{cases} R(t) & \text{if} \quad t < y, \\ R(t) - r_c(t - y) & \text{if} \quad t \geq y. \end{cases} \tag{8}$$

Consider the functions at time $t_{\min}$. It must be true that $t_{\min} \geq t_i$, because

$$B(t_{\min}, t_i) = B(t_{\min}, 0) + m = -m + m = 0, \tag{9}$$

and for the intersection point to occur at $R(t_i) < 0$ implies that $t_i < 0$.

By definition,

$$B(t_{\min}, t_i) = R(t) - r_c(t - t_i) = R(t) - r_c t + r_c t_i, \tag{10}$$

and

$$B(t_{\min}, y) = R(t) - r_c(t - y) = R(t) - r_c t + r_c y. \tag{11}$$

Because $y < t_i$, it follows that $B(t_{\min}, y) < B(t_{\min}, t_i) = 0$. Therefore, $B(t_{\min}, y) < 0$, which is a contradiction, because a playback solution cannot have a value less than zero until after time t_r. Therefore, t_i is the minimum playback start time. □

3.2 Implications of Blocked Data Reading

In the previous section, data was modeled as being available for consumption as soon as they were read. However, when the storage device requires that data be read in units of a certain size (usually called physical blocks, or sectors), this may not be true. For instance, on most disks, the smallest amount of data that may be read is a sector, which is typically about 512 bytes in size. Various error correction codes are stored with the sector, and in most cases they cannot be used to correct errors until the read of the entire sector is complete. In order to avoid confusion between the logical blocks that are discussed later and physical blocks, we follow disk drive terminology and refer to physical blocks as *sectors*.

There is a question, then, as to where the data is buffered before the error correction is applied. It may be buffered by the disk drive unit, by the disk controller, or possibly even in the computer's memory (although this is unlikely). The problem is how to model the read function when data is read in sectors.

Figure 5 show two possibilities for modeling the read function. The first graph shows the read function modeled as the number of samples read. The second shows the read function as the number of samples read and usable. Whereas the first graph rises with each sample read, the second only rises when a sector read is complete and when error correction can be applied. It then jumps suddenly to catch up with the first graph.

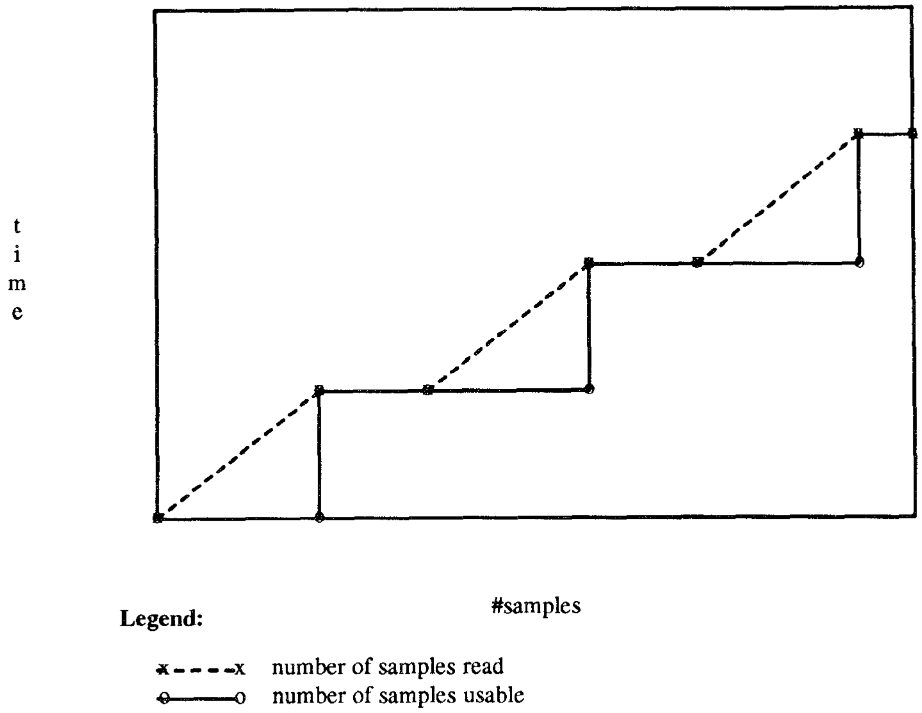

Fig. 5 Reading samples in sector units.

With the number of samples read and usable as the reading function, the minimum start time may be found using the results of the previous section. The number of samples read should then be used to calculate buffer usage.

Buffer space for the entire sector must usually be allocated before the read command for the sector is issued. Because this is usually true, the results that follow consider the number of sector-sized buffers that must be allocated in order to perform playback.

4. SPECIAL CASES AND TECHNIQUES FOR PLAYBACK SYSTEMS

Up to this point, we have developed a general theoretical framework for studying a playback system with a dedicated storage device. The framework is general, in that it does not make any assumptions about the way in which data are read from the storage device. However, this requires that the reading function, $R(t)$, is known or derived for the whole playback. In some special cases, the reading function can be considered uniform (or approximated to be uniform), and the study can then be simplified. In this section we analyze some special cases for which this is true.

In each example it is assumed that data is stored in sectors, with the read function not reflecting any increase until the entire sector has been read, and with buffer space for the sector allocated before the read command is issued.

In addition, it is assumed that data is not compressed (the results are also valid if the rate of compression is the same for every sector, which is equivalent to increasing the transfer rate of the storage device).

4.1 Continuous Playback

Consider the case of playing back a single channel of audio, in which reading is continuous. By continuous reading we mean that sectors are read one after another with no delay in between. Such a situation would occur when reading a contiguous record from a disk drive with a spiral format, when reading from a tape drive, or possibly when obtaining data from a network.

We present theorems for three cases:

(1) The transfer rate of the device is equal to the consumption rate.

(2) The transfer rate is less than the consumption rate.

(3) The transfer rate is greater than the consumption rate.

In each case we assume a storage device with sector size s_s and transfer rate r_t, and an audio record that can be read continuously, with a length of $l_a > 1$ sectors. Each theorem demonstrates a bound on the maximum buffer utilization (i.e., peak value of $B(t, t_0)$) and on the minimum number of sector-sized buffers that must be allocated.

THEOREM 3. *Given the assumptions described above and a consumption rate equal to the transfer rate of the storage device ($r_c = r_t$), then any playback algorithm must have a maximum buffer space utilization of at least s_s and must allocate at least two sector-sized buffers.*

PROOF. Data must be read before playback can begin. Because the data for a sector is not available until the read is complete, s_s data must be buffered before playback can begin.

Suppose no sectors are read during playback. Then they must all be buffered before playback begins. Because the size of the audio record is greater than one, at least two buffers are required. Suppose alternatively that sectors are read during playback. As one sector-sized buffer is being filled by reading, in order for playback to occur, there must be another buffer that contains data being consumed. Therefore, at least two buffers are always required. □

THEOREM 4. *Given the assumptions at the beginning of this subsection and a consumption rate less than the transfer rate of the storage device ($r_c < r_t$), then any playback algorithm must have a maximum buffer space utilization of at least*

$$u_{max} \geq (l_a - 1)\frac{s_s}{r_t}(r_t - r_c) + s_s, \tag{12}$$

and the number of sector-sized buffers required for playback is at least

$$n_b \geq \frac{l_a - 1}{r_t}(r_t - r_c) + 1. \tag{13}$$

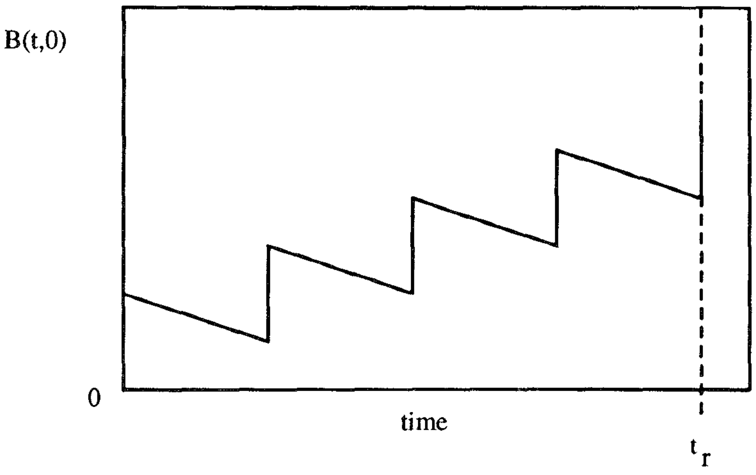

Fig. 6. Buffer function for Theorem 4.

Proof. Reading is continuous, but data only arrives after each sector read has completed. It takes s_s/r_t to read each sector, so the read function jumps by s_s at intervals of s_s/r_t. If we let time $t = 0$ be when data from the first sector becomes available for use, then $R(0) = ScSz$. The read function must then be

$$R(t) = s_s + \left\lfloor t\frac{r_t}{s_s} \right\rfloor s_s \tag{14}$$

and $t_r = (l_a - 1)(s_s/r_t)$. Therefore, the buffer function for a start time of zero must be

$$B(t,0) = s_s + \left\lfloor t\frac{r_t}{s_s} \right\rfloor s_s - r_c t. \tag{15}$$

During the read of a sector, the amount consumed is $r_c(s_s/r_t)$, and the buffer function falls by this amount. At the end of the read, when the data become available, it jumps by s_s. Because $r_c < r_t$, during every sector read the buffer function falls by less than a sector, and $B(t,0)$ increases in a zigzag pattern, as illustrated in Figure 6. Because $B(t,0)$ is always positive up to t_r, it is a solution. By Theorem 2, we know that this is optimal and that the starting time is then $t = 0$. It is easy to see that it reaches its maximum at time t_r and that this will be

$$B(t_r,0) = s_s + \left\lfloor t_r\frac{r_t}{s_s} \right\rfloor s_s - r_c t_r. \tag{16}$$

Because t_r comes at the end of reading, it must be a multiple of s_s/r_t, so the floor function can be dropped to obtain

$$u_{\max} = B(t_r,0) = s_s + t_r(r_t - r_c), \tag{17}$$

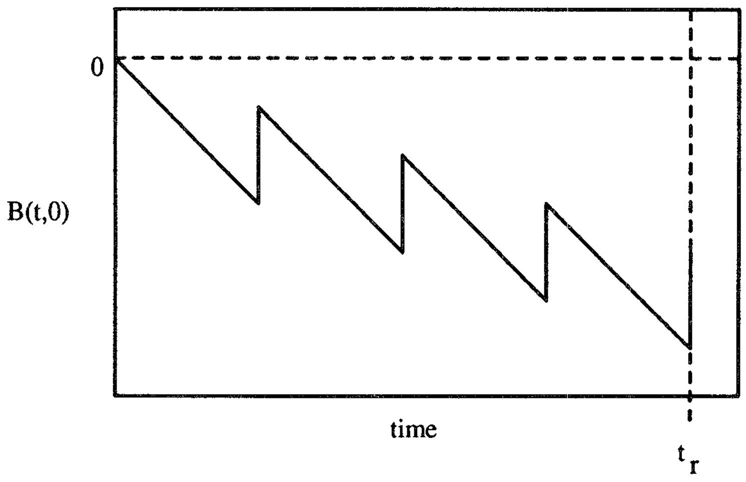

Fig. 7. Buffer function for Theorem 5.

and it must be true that

$$n_b \ge \left\lceil \frac{u_{\max}}{s_s} \right\rceil = \frac{t_r}{s_s}(r_t - r_c) + 1. \quad \square \tag{18}$$

THEOREM 5. *Given the assumptions at the beginning of this subsection and a consumption rate greater than the transfer rate of the storage device ($r_c > r_t$), then any playback algorithm must have a maximum buffer space utilization of at least*

$$u_{max} \ge \left\lfloor \frac{t_r(r_c - r_t) - s_s}{r_c} \left(\frac{r_t}{s_s} \right) \right\rfloor s_s. \tag{19}$$

In addition, the number of sector-sized buffers required for playback is at least

$$n_b \ge \left\lceil \frac{u_{max}}{s_s} \right\rceil. \tag{20}$$

PROOF. Let the read and buffering functions be adopted from the previous proof, except with time $t = 0$ being the moment when the first sector begins being read, so that $R(0) = 0$. The buffering function is then

$$B(t,0) = \left\lfloor t\frac{r_t}{s_s} \right\rfloor s_s - r_c t \tag{21}$$

and $t_r = l_a(s_s/r_t)$.

During every sector read, the buffer function falls by at least a sector, because $r_c > r_t$. This causes $B(t,0)$ to decrease in a zigzag pattern, as illustrated in Figure 7. It is easy to see that it reaches its minimum at the end of the last read, just before the sector data becomes available. This is at time t_r, and this minimum is $t_r(r_c - r_t) - s_s$. Using Theorem 2, $t_{\min}$ may be found by calculating the intersection of $R(t)$ with $B(t, 0) + t_r(r_c - r_t) - s_s$. This is found by solving

$$B(t, 0) + t_r(r_c - r_t) - s_s = R(t), \tag{22}$$

which, by the definitions of $R(t)$ and $B(t, 0)$, is equivalent to solving

$$\left\lfloor t_{\min}\frac{r_t}{s_s}\right\rfloor s_s - r_c t_{\min} + t_r(r_c - r_t) - s_s = \left\lfloor t_{\min}\frac{r_t}{s_s}\right\rfloor s_s, \tag{23}$$

which yields

$$t_{\min} = \frac{t_r(r_c - r_t) - s_s}{r_c}. \tag{24}$$

Now consider the buffer space utilization. The buffering function $B(t, t_{\min})$ is the same as the reading function up to time $t_{\min}$, so the maximum buffer utilization must be at least

$$B(t_{\min}, t_{\min}) = R(t_{\min}) = \left\lfloor \frac{t_r(r_c - r_t) - s_s}{r_c}\left(\frac{r_t}{r_c}\right)\right\rfloor s_s, \tag{25}$$

which requires that

$$\left\lceil \frac{R(t_{\min})}{s_s} \right\rceil \tag{26}$$

sector-sized buffers be allocated. (Note that this bound is not tight; the next read after time $t_{\min}$ may complete before a buffer has been emptied, so one more buffer may be required.) □

Therefore, we observe that, for the special case where the consumption rate is equal to the transfer rate of the device, playback can be done using two sector-sized buffers. In all other cases, the buffer space required is dependent on the length of the recording. In practice, it is undesirable to have buffer space dependent on the length of the audio record, because it will mean that only records up to a certain length can be played back with the given system memory. Therefore, it is desirable that a playback algorithm be able to handle records of arbitrary length.

It should be noted that, when the consumption rate is less than the transfer rate, samples are being read faster than they can be consumed. If somehow reading could be slowed down, then the buffer requirements would be reduced. For example, if the transfer rate was exactly twice the consumption rate, then a samples-sized gap could be left after each sample to reduce the effective transfer rate. Different strategies are possible, but the essential ingredient in all of them is the introduction of artificial delays, which is discussed in the next section.

4.2 Playback with Artificial Delays

In the previous section, it was noted that it is not practical to have buffer space requirements dependent on the length of the audio record. In this section, the same case of single-channel uncompressed playback is considered. It is assumed that the record could be played back continuously with no delays, but that artificial delays are introduced in order to limit buffer requirements. It is also assumed from this point on that the consumption rate is no more than the transfer rate of the device ($r_c \leq r_t$), because otherwise the buffer requirements must depend on the length of the audio record. The algorithms under consideration are those that can play back records of arbitrary length.

Delays can be created in many ways. For the purposes of this result, it does not matter how they are accomplished. The assumption will be that delays cannot be of arbitrary length, but must be in multiples of some minimum delay. An example of an artifical delay on a disk would simply be to stop reading. In this case, the length of the delay must be a multiple of the rotational delay.

THEOREM 6. *For a single-channel audio record stored in sectors of size s_s, which could be read contiguously, but in which delays of length d_{min} may be introduced, the number of sector-sized buffers required for playback of arbitrarily long records, with $r_t > r_c$, is at least*

$$n_b \geq \left\lceil \left(\frac{s_s}{r_t} + d_{min} \right) \frac{r_c}{s_s} \right\rceil. \tag{27}$$

PROOF. Suppose that the playback algorithm requiring the least number of buffers introduces no delays. For the algorithm to work over all lengths of records, it is apparent from the previous theorem that it must be the case that $r_t = r_c$. However, this violates our assumption, so delays must be introduced. Furthermore, these delays will necessarily come after the beginning of playback, because there is no possible gain from introducing delays prior to that.

Any time there is a delay, there must be enough data buffered to last for the duration of the delay, plus the time it takes to read a sector after the delay is complete. So, if the amount of data buffered at the beginning of a delay is b, then

$$b \geq r_c \left(d_{\min} + \frac{s_s}{r_t} \right), \tag{28}$$

and at least $\lceil b/s_s \rceil$ buffers must be allocated. □

The above bound relies on using all of the available buffers to hold data during delays. In practice, this may be difficult to implement, because as the last buffer is being filled another buffer is being emptied. The next theorem gives an algorithm that uses one buffer more than the bound presented above.

THEOREM 7. *For a single-channel audio record stored in sectors of size s_s, which could be read contiguously, but in which delays of length d_{min} may be* *introduced, and with $r_t > r_c$, there exists a playback algorithm using*

$$n_b = \left\lceil \left(\frac{s_s}{r_t} + d_{min} \right) \frac{r_c}{s_s} \right\rceil + 1 \tag{29}$$

sector-sized buffers.

Proof. Consider the following algorithm:

(1) Allocate $\lceil ((s_s/r_t) + (d_{\min}))(r_c/s_s) \rceil + 1$ sector-sized buffers.
(2) Begin reading the audio record sequentially, and fill the first buffer.
(3) Begin playback.
(4) Repeat until no sectors are left to be read:
  (i) If any buffer is empty, read the next sector and fill it.
  (ii) If no buffers are empty, perform an artificial delay.

The above is a solution if there is always data buffered, to be consumed from the time playback begins until all reading is complete. When playback begins, all buffers are full. When there are no delays, there will always be data buffered because $r_t > r_c$. Delays during playback only occur when there are no free buffers, that is, when $n_b - 1$ buffers are full, and one buffer is nonempty. The amount of data consumed during a delay is $d_{\min} r_c$. The amount of data in the full buffers is

$$\left\lceil \left(\frac{s_s}{r_t} + d_{\min} \right) \frac{r_c}{s_s} \right\rceil s_s \geq \frac{r_c}{r_t} s_s + d_{\min} r_c . \tag{30}$$

Therefore, at the end of any delay, there will be at least $(r_c/r_t)s_s$ still buffered. At the end of each delay, if there are still no free buffers, then another delay may be introduced with the above still holding. Otherwise, if there are free buffers, then the next sector will be read, and because at least $(r_c/r_t)s_s$ is still buffered, there will always be data buffered for consumption until the read is completed. □

4.3 Playback with Unavoidable Delays

In this section we consider playback in which reading is not continuous, but that includes some delays by necessity (e.g., seeks). Again, we assume that it is possible to introduce artificial delays. First, theorems are given that express certain bounds, and then results of a simulation verifying the bounds are given.

4.3.1 *Block Size and Buffer Allocation*

Theorem 8. *Consider a single-channel system that plays back audio records of arbitrary length, with the following properties:*

(1) *Data is read in blocks, each block consisting of a fixed number of sectors. The block size is $s_b = is_s$, where i is an integer.*
(2) *In between reading each block, a delay is incurred. The length of each delay is unpredictable, but is bounded above by d_{max}.*

(3) *The minimum length of an artificial delay is* d_{min}, *with* $d_{min} < d_{max}$.

(4) *The transfer rate of the storage device is greater than the consumption rate* $(r_t > r_c)$.

Then any algorithm that allocates a number of buffers independent of the record length must use a block size of at least

$$s_b \geq \left\lceil \frac{r_c d_{max}}{s_s} \left(1 - \frac{r_c}{r_t} \right) \right\rceil \tag{31}$$

sectors and must allocate at least

$$n_b \geq \left\lceil \frac{r_c}{s_s} \left(d_{max} + \frac{s_s}{r_t} + d_{min} \right) \right\rceil \tag{32}$$

sector-sized buffers.

PROOF. Consider the case where all delays are of maximum length, that is, $d_{\max}$. Let playback (consumption) begin at time t_0, and consider a block read after this time. Let it be the nth block read. As the read for the first sector of the nth block is completed, the amount buffered will rise. Just before this rise, the total amount of data read will be $(n - 1)s_b$. The total amount of data consumed by this time will be

$$(n - 1)\left(d_{\max} + \frac{s_b}{r_t} \right) r_c - t_0 r_c. \tag{33}$$

Because reading must always stay ahead of consumption,

$$(n - 1)s_b \geq (n - 1)\left(d_{\max} + \frac{s_b}{r_t} \right) r_c - t_0 r_c, \tag{34}$$

which yields

$$s_b \geq \frac{d_{\max} - \dfrac{t_0}{n - 1}}{\dfrac{1}{r_c} - \dfrac{1}{r_t}}. \tag{35}$$

We can take n to be arbitrarily large, so this will approach

$$s_b \geq \frac{d_{\max}}{\dfrac{1}{r_c} - \dfrac{1}{r_t}}. \tag{36}$$

Because the block size is in sector units, we must have

$$s_b \geq \left\lceil \frac{r_c d_{\max}}{s_s} \left(1 - \frac{r_c}{r_t} \right) \right\rceil s_s. \tag{37}$$

Now consider the buffer allocation requirements. Let the amount of data added to the buffer by reading a block during playback be

$$r_c\left(d_{\max} + \frac{s_s}{r_t} + \psi\right), \tag{38}$$

that is, the change in the buffer from function, from the moment just before the first sector of the block becomes available to the moment when the last sector of the block becomes available. It is assumed that playback begins when the first sector becomes available, so this is the amount in the buffer at the end of reading the first block. If $\psi \geq d_{\min}$, then we are done, so assume that $\psi < d_{\min}$.

Consider two blocks read after playback (consumption) has begun. Let the amount of data left in the buffer after the first block is read be

$$r_c\left(d_{\max} + \frac{s_s}{r_t} + \psi + \gamma\right). \tag{39}$$

This means that $r_c\gamma$ was left in the buffer previous to the read. γ may be zero, but is nonnegative. If $\psi + \gamma \geq d_{\min}$, then we are done, so assume that $\psi + \gamma < d_{\min}$.

Let $k = \lfloor \lceil (2\psi + \gamma)/d_{\min} \rceil + 1 \rfloor$. For all x it is true that $\lfloor x + 1 \rfloor > x$, so $kd_{\min} > 2\psi + \gamma$. Now let $\epsilon > 0$ be chosen such that $\epsilon < kd_{\min} - 2\psi - \gamma$.

Suppose that after the first block is read there is a delay of $d_{\max} + 2\psi + \gamma - kd_{\min} + \epsilon$, and after the second block is read there is a delay of $d_{\max}$. Note that the first delay is in fact less than $d_{\max}$, as the choice of ϵ and k ensures. Also, the first delay is in fact a nonnegative value, as k is at most 2, and is only that value when $2\psi + \gamma$ is at least $d_{\min}$.

Suppose also that no artificial delay is introduced before reading of the second block is complete. From the time that the first block completes reading until the time the first sector of the second block is available,

$$r_c\left(d_{\max} + 2\psi + \gamma - kd_{\min} + \epsilon + \frac{s_s}{r_t}\right) \tag{40}$$

sectors will be consumed. There was

$$r_c\left(d_{\max} + \frac{s_s}{r_t} + \psi + \gamma\right) \tag{41}$$

in the buffer when the first block completed reading; therefore, just as the first sector of the second block becomes available, there is

$$\begin{aligned} &r_c\left(d_{\max} + \frac{s_s}{r_t} + \psi + \gamma\right) - r_c\left(d_{\max} + 2\psi + \gamma - kd_{\min} + \epsilon + \frac{s_s}{r_t}\right) \\ &\quad = r_c(kd_{\min} - \epsilon - \psi) \end{aligned} \tag{42}$$

in the buffer. So after the second block completes reading, the amount buffered will be

$$r_c\left(d_{\max} + \frac{s_s}{r_t} + \psi\right) + r_c(kd_{\min} - \epsilon - \psi) = r_c\left(d_{\max} + \frac{s_s}{r_t} + kd_{\min} - \epsilon\right). \quad (43)$$

If k artificial delays had been introduced before the second block completed reading, then

$$r_c\left(d_{\max} + \frac{s_s}{r_t} - \epsilon\right) \quad (44)$$

would be left in the buffer. However, the next delay is of length $d_{\max}$, so by the time the first sector of the third block becomes available, $r_c[d_{\max} + (s_s/r_t)]$ would have been consumed. This would make the buffer function negative, which is an error in playback. Therefore, at most $k - 1$ delays could be introduced, reducing the buffer space to

$$r_c\left(d_{\max} + \frac{s_s}{r_t} + d_{\min} - \epsilon\right). \quad (45)$$

ϵ can be chosen to be arbitrarily small. However, partial samples cannot be utilized for consumption. Therefore, the buffer space must be at least

$$r_c\left(d_{\max} + \frac{s_s}{r_t} + d_{\min}\right). \quad (46)$$

And the number of sector-sized buffers must then be at least

$$n_b \geq \left\lceil \frac{r_c}{s_s}\left(d_{\max} + \frac{s_s}{r_t} + d_{\min}\right)\right\rceil. \quad \square \quad (47)$$

As in the case of Theorem 6, the bound for buffer allocation given in Theorem 8 is difficult to achieve in practice, because during playback there is always one buffer being emptied as one is being filled. Theorem 9 gives an algorithm that comes close to achieving the bound given in Theorem 8.

THEOREM 9. *Consider a single-channel playback system with the following properties*:

(1) *Data is read in blocks, each block consisting of a fixed number of sectors. The block size is* $s_b = is_s$, *where i is an integer.*

(2) *In between reading each block, a delay is incurred. The length of each delay is unpredictable, but is bounded above by* d_{max}.

(3) *The minimum length of an artificial delay is* d_{min}, *with* $d_{min} < T_d$.

(4) *The transfer rate of the storage device is greater than the consumption rate* $(r_t > r_c)$.

Then there exists an algorithm requiring block sizes of

$$\left\lceil r_c \frac{d_{max}}{s_s\left(1 - \frac{r_c}{r_t}\right)} \right\rceil \tag{48}$$

sectors and allocating

$$\left\lceil r_c \frac{d_{max}}{s_s\left(1 - \frac{r_c}{r_t}\right)} \right\rceil s_s + \left\lceil \frac{r_c}{s_s}\left(d_{min} + \frac{s_s}{r_t} \right) \right\rceil \tag{49}$$

sector-sized buffers.

Proof. The algorithm is as follows: Let the block size and number of buffers allocated be as above. Then

(1) Read the first block.
(2) Begin playback.
(3) Repeat until reading is completed:
  (i) Let the delay incurred be of length $T_d < d_{\max}$.
  (ii) While the amount still buffered is at least enough to satisfy consumption for an artificial delay and a block read,
    (a) perform an artificial delay.
  (iii) Read the next block

The block size satisfies the requirements of the theorem. It remains to show that the buffer allocation is sufficient and that the algorithm does constitute a playback solution. It is a playback solution if there is always data buffered, to be consumed from the time playback begins until all reading is complete.

Clearly there is data buffered when playback begins. The amount buffered after the read of any block is enough to satisfy consumption through the worst-case delay, and reading the next sector. Therefore, playback could only fail as a result of delays. However, delays are only performed when there is enough data buffered to last through a delay and a block read. Therefore, the above is a playback solution.

Now consider the buffer space utilized by the above algorithm. Before reading any block, there will always be less than

$$r_c\left(d_{\min} + \frac{s_s}{r_t} \right) \tag{50}$$

buffered; otherwise, another delay would have been issued to reduce the buffer space. Therefore, before reading the block, there may be

$$\left\lceil \frac{r_c}{s_s}\left(d_{\min} + \frac{s_s}{r_t} \right) \right\rceil \tag{51}$$

buffers in use. Reading the block requires at most

$$\left\lceil \frac{r_c d_{\max}}{s_s}\left(1 - \frac{r_c}{r_t}\right)\right\rceil \tag{52}$$

more buffers. Therefore, the maximum number of buffers required is

$$\left\lceil \frac{r_c d_{\max}}{s_s}\left(1 - \frac{r_c}{r_t}\right)\right\rceil + \left\lceil \frac{r_c}{s_s}\left(d_{\min} + \frac{s_s}{r_t}\right)\right\rceil. \quad \square \tag{53}$$

4.3.2 *Simulation Results for Uncompressed Playback with Unavoidable Delays.* The scenario described in the last two theorems can occur using a magnetic disk as the storage medium. Audio data can be stored in blocks, each block located on a single track. If blocks are stored in random locations, then a seek will be performed to each block. The seek will then be random, but will be bounded above by the maximum seek time of the disk drive.

Following the algorithm given in Theorem 9, a seek would be performed, and the read head moved over the first sector of the block to be read. If the proper conditions were satisfied, it would be read; otherwise, it would not be read, and a rotational delay would be allowed to occur. However, this cannot be done in practice, because by the time the calculation completes to see if the conditions are satisfied, the sector would have begun moving under the head, and it would be too late to read it. Therefore, for the purposes of the simulation, the algorithm was modified to the following:

(1) Read the first block.

(2) Begin playback.

(3) Repeat until reading is completed:

   (i) Perform a seek to the track the next block is on.

   (ii) While the amount still buffered is more than enough to satisfy consumption for a rotational delay, a block read, and any hardware overhead, wait.

   (iii) Read the next block.

This has the same properties in terms of buffer requirements. $d_{\max}$ in this case would be a maximum seek, plus a maximum rotation, plus any overhead delays. $d_{\min}$ would be a maximum rotation plus any overhead delays.

From the theorem, the block size required is

$$\left\lceil \frac{r_c d_{\max}}{s_s}\left(1 - \frac{r_c}{r_t}\right)\right\rceil \tag{54}$$

sectors. Therefore, if a block size of n sectors is chosen, with n being an integer, a consumption rate of

$$r_c = \frac{n s_s}{d_{\max}}\left(1 - \frac{r_c}{r_t}\right) \tag{55}$$

should be feasible to support, according to the theorem. The simulation results were compared to this value.

```
BEGIN
    TB = time to consume a block
    numsectors = number of sectors in a block
    /*** wait time is time that we may have to wait for new data to be
     *** available after issuing a read command
     ***/
    wait_time = Maximum rotation time + hardware overhead
              + time to read a sector
    'read' first block (actually verify)
    /*** buff_time is the amount of time it takes to consume the total
     *** amount of data read so far.
     ***/
    buff_time = TB;
    /* start playback */
    start_time = the current time
    WHILE there are more blocks to read
        seek to the track the next block is on
        temp =  buff_time - extra;
        do { /* wait as long as we can with having danger of failure */
            now = the current time
        } while ( now - start_time < buff_time - wait_time);
        'read' the next block (actually verify)
        /*** See if we ran out of data while the block was read.
         *** This would be the case if the amount buffered was
         *** not enough to last until the end of reading the
         *** first sector.
         ***/
        now = the current time
        1st_sector_read_time = now - (time to read numsectors-1 sectors)
        spare_time =  buff_time - (1st_sector_read_time - start_time);
        if (spare_time < 0)
            simulation fails  (exit)
        buff_time = buff_time + TB;
    ENDWHILE
    simulation is successful
ENDsimulation
```

Fig. 8. Pseudocode for the simulation of Theorem 9.

The routine that simulates the above algorithm is passed a list of blocks to read, the number of blocks in the list, the consumption rate to simulate, and the number of sectors per block. The pseudocode for it is given in Figure 8.

Code was written to generate two types of lists for the simulation routine; a worst-case list and a nonworst-case list. In each case blocks began at the beginning of a track, so that each block would lie entirely within one track. For a list which is not worst case, each block began on a random track. For a

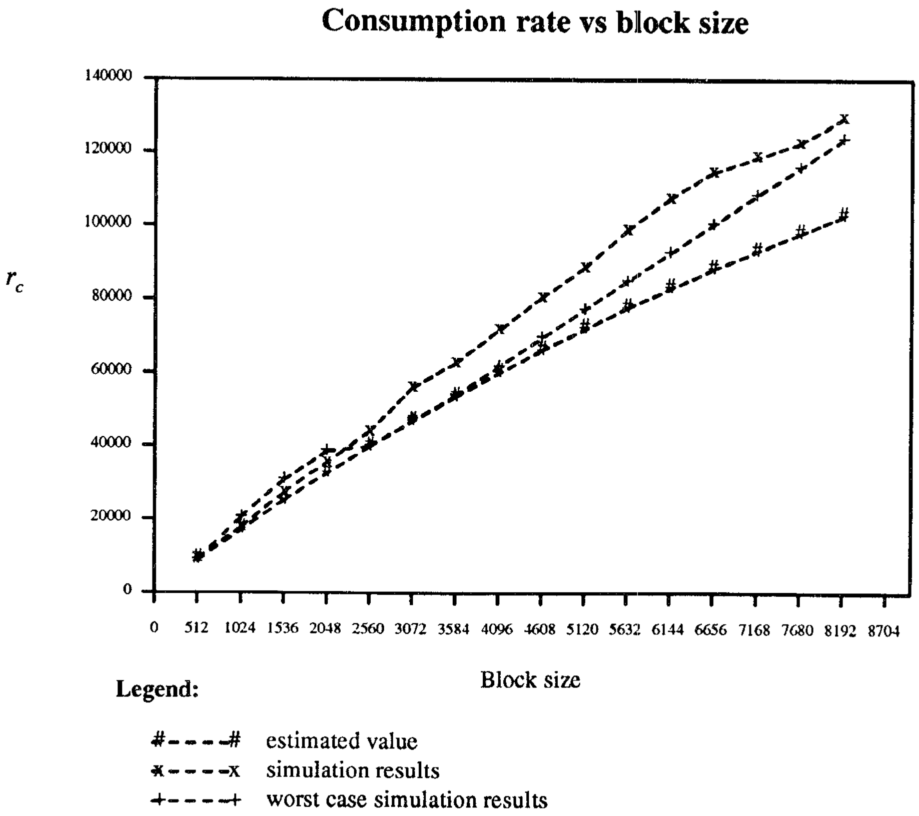

Fig. 9. Experimental results for Theorem 9 simulations. Maximum consumption rate (bytes/s) supportable by a given block size (bytes). Comparison of experimental results with predicted values.

worst-case list, blocks alternated between being on the inside and outside cylinders, with the actual track within the cylinder being random.

The simulation was performed using a 155-Mbyte SCSI hard drive attached to an IBM PC/AT compatible. The software was written under the DOS operating system, using Microsoft's C compiler, assembler, and linker. SCSI commands were sent to the drive using the software driver provided with the controller card. The driver contained a command that allowed SCSI command data blocks to be transferred, so that the drive could be directly under the experimental program's control, with no intervening algorithms applied. Timing was done using the Intel 8254-2 timer/counter chip built into the PC/AT, with an accuracy in excess of 1 μs.

Experimentally, it was found that a delay of at least 6 ms must occur before a sector may be read. Because of this, when issuing a read command while already on the correct track the possible delay is as high as a maximum rotation plus 6 ms. The rotational delay of the drive is at most 17 ms.

The maximum seek time is 40 ms. The transfer rate of the drive is 1.07 Mbytes/s. Sectors were 512 bytes.

The simulation was run using both the worst and nonworst cases, for various numbers of sectors per block. For each trial, 20 blocks were read. Each time, the simulation was run using various consumption rates until the maximum consumption rate for which no failures occurred in 20 trials was recorded. The graph in Figure 9 show the results of the simulations, with the predicted results indicated.

The simulation results meet or exceed the predicted results, verifying Theorem 9. In some cases, the worst-case results are better than the nonworst-case results. This is because the performance of each simulation depends more on the rotational delays encountered than on the length of seeks.

5. PROPERTIES OF MULTICHANNEL PLAYBACK

In the previous sections, we have studied the case of single-channel (i.e., monophonic) audio playback. Now we turn our attention to multichannel playback, in which several channels of audio are played back synchronously. A relatively simply playback system is studied in order to illustrate some important properties. Let the playback system have the following properties:

—The storage device is dedicated to the audio playback.

—The number of playback channels is n_c

—Each channel has the same consumption rate, r_c. The sum of the consumption rates is at most the transfer rate of the storage device, $n_c r_c \leq r_t$.

—Playback can be divided into *reading periods* during which s_b data are read for each channel. (Note that, although we use s_b to indicate the amount, the data need not be stored as a physical block for each channel. No assumptions are made as to how the data are physically stored.)

—During each reading period, there are delays of nonzero length that are bounded above by $d_{\max}$.

Let the following variables be defined:

k the ratio of the data transfer rate of the storage device over the total data consumption rate, that is, $r_t / n_c r_c$. Because we assume that $n_c r_c \leq r_t$, it must be the case that $k \geq 1$.

l_p the reading period length. This is the delay time plus the reading time.

We are partcularly interested in the reading period length and the block size. In general, the delays associated with retrieving the first block for each channel are similar to that for the remaining blocks. Therefore, the reading period length is a good indication as to how long the delay will be between a request for playback and the playback beginning, and it is desirable for it to be as low as possible. Similarly, s_b gives a good indication of the buffering requirements.

The reading period length is at most the maximum delay plus the time needed to transfer one block of data for each channel:

$$l_p \le d_{\max} + \frac{n_c s_b}{r_t}. \tag{56}$$

It is clear that for playback to work for records of arbitrary length there must be enough buffered in each reading period to satisfy consumption for the duration of a reading period; that is,

$$s_b \ge \frac{r_c}{l_p}. \tag{57}$$

It is possible that each reading period may be of maximum length, so

$$s_b \ge \frac{r_c}{d_{\max} + \dfrac{n_c s_b}{r_t}}, \tag{58}$$

which can be rewritten using k as

$$s_b \ge \frac{k r_c d_{\max}}{k-1} = \frac{r_t d_{\max}}{n_c(k-1)} \tag{59}$$

and which can be substituted into the formula for reading period length to obtain

$$l_p \ge d_{\max}\frac{k}{k-1}. \tag{60}$$

Solving for consumption rate yields

$$r_c \le \frac{s_b}{l_p} = \frac{1}{\dfrac{n_c}{r_t} + \dfrac{d_{\max}}{s_b}}, \tag{61}$$

or

$$r_c \le \frac{s_b}{d_{\max}}\left(\frac{k-1}{k}\right). \tag{62}$$

As expected, the above indicates that as $d_{\max}$ approaches zero the block size does as well. In practice, this means that the block size comes down to one, meaning that samples can be read off the storage device and transferred directly to the digital-to-analog converter.

Now consider the formulas for s_b[2] and l_p. Both of these contain a $1/(k-1)$ factor. This factor causes them to approach infinity near $k = 1$. Between 1 and 2, they drop very steeply and then begin to flatten out.

[2] It is possible that larger blocks may increase delays (e.g., on disks they may be more likely to cross cylinder boundaries), so that $d_{\max}$ is in fact a function of s_b. However, the contribution is small and only causes the effect discussed here to become more pronounced.

The interpretation of this is that it is very costly in terms of block size and reading period length to have the total data consumption rate very close to the transfer rate of the storage device. However, although significant gains can be made by keeping the total data consumption rate from being too close to the transfer rate, making it very small fails to yield improvements of the same order. For example, it would probably yield significant savings to keep the total data consumption rate less than one half the storage device data transfer rate, with reasonable decreases achieved by further reducing it to about one fourth the storage data transfer rate. Any further reduction, however, would probably have little effect.

Stated the other way around, if k is close to 1 then $d_{\max}$ must be close to zero. For this reason, it may not be feasible to do any seeks if the storage device data transfer rate is very close in value to the total data consumption rate.

Consider a real example. A CD-ROM with interleaved data can support two channels of 44.1-KHz sound, with 16 bits per sample. That is, $n_c = 2$, $r_c = 88{,}200$ bytes/s, $k = 1$. No seeking is done. Suppose we want to be able to do a half-second seek in every reading period. We need to increase k by decreasing r_c. If we decrease r_c by about 1 percent ($k = 1.01$), we find that we need about a 2-Mbyte block size. We also have a reading period length of about 50 seconds. If we decrease r_c by 25 percent ($k = 4/3$), we only need about 65 Kbytes, and our reading period length is 2 seconds. If we decrease r_c by 50 percent ($k = 2$), we only need about 20 Kbytes, and our reading period length is 1 second. See Figures 10 and 11 for graphs of block size and reading period length versus the ratio of supply rate over consumption rate.

This covers the case for a seek of 0.5 seconds, but one might ask what can be done to support an arbitrary seek time. By manipulating the equation for consumption rate, we can obtain

$$d_{\max} \le s_b\left(\frac{1}{r_c} - \frac{n_c}{r_t}\right). \tag{63}$$

So there are three ways to support a greater seek time:

(1) Decrease the consumption rate of each channel of audio.

(2) Increase the block size.

(3) Increase the data transfer rate of the storage device.

Decreasing the consumption rate gives a proportional increase in allowable seek time. This is a straightforward and effective method to allow for an arbitrary amount of seeking, but normally an application will have some sound quality requirements, which mean minimum consumption rate requirements.

Increasing the block size gives a proportional increase in the seek time we can support. However, it also increases the reading period length if the storage device data transfer rate remains the same. Increasing the data transfer rate of the storage device has some effect on the seek time we can support, but only to a limited extent. Because it appears in the formula as a

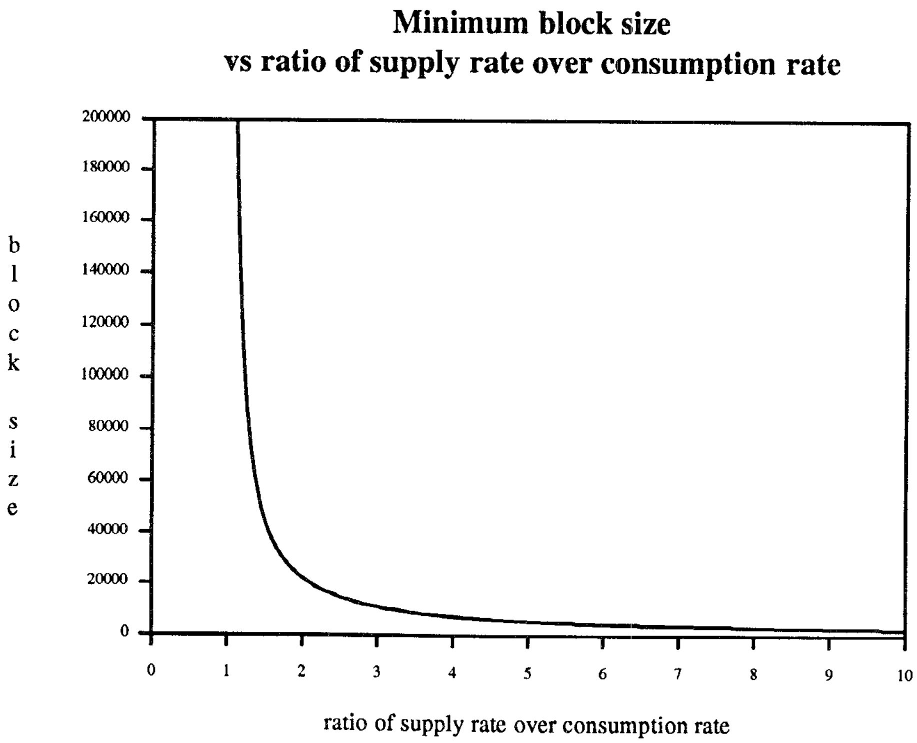

Fig. 10. Minimum block size (bytes) versus ratio of supply rate over consumption rate. Calculated for two audio channels, a supply rate of 88,200 bytes/s, and a delay time of 0.5 s per reading period.

negative inverse, it only helps to approach s_b / r_c asymptotically. Because the growth is asymptotic, increasing the transfer rate fails to be very effective after a certain point.

6. SHARING THE STORAGE DEVICE

Up to this point, we have assumed that the storage device is dedicated to the playback of a single audio record. This is a useful assumption for finding the maximum system performance for a single playback task. However, in multiuser or multitasking systems it is clearly undesirable to dedicate the storage device in this way. This is especially true if the device is being used to serve many clients over a network. In this section we extend the model given in the previous section to obtain two approaches toward dealing with shared storage devices: the free-time model and the channels model.

6.1 The Free-Time Model

For sharing of a storage device to occur as a playback task runs, the real-time demands of the audio task must be met without using the storage device constantly. In terms of the model given in the previous section, this means

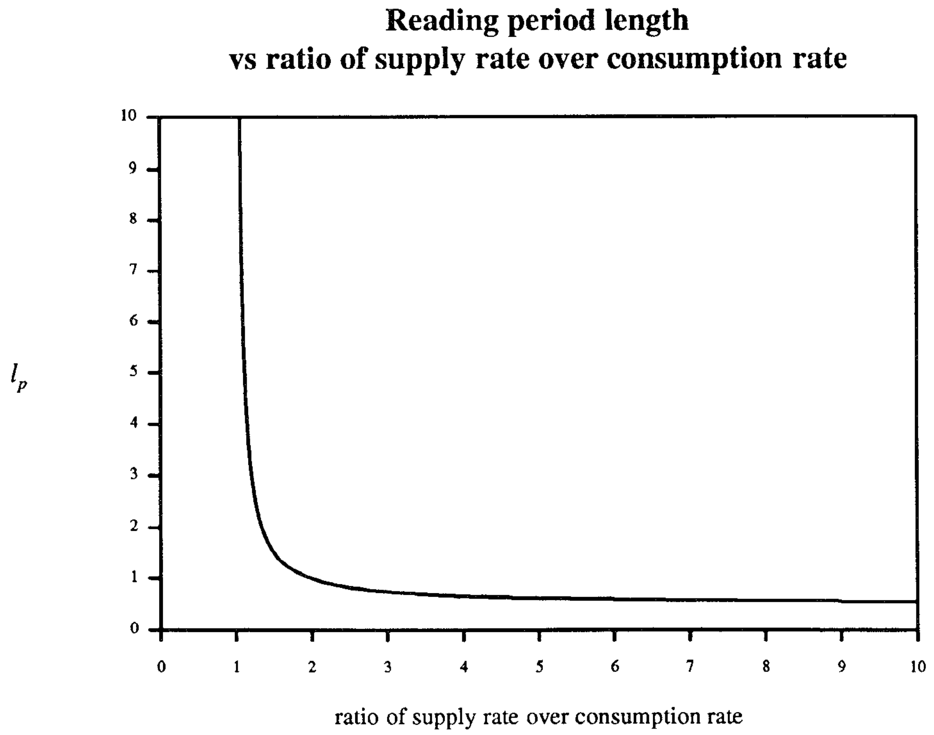

Fig. 11. Reading period length (in seconds) versus ratio of supply rate over consumption rate. Calculated for two audio channels, a supply rate of 88,200 bytes/s, and a delay tıme of 0.5 s per reading period.

that there must be some free time in every reading period when the audio playback task is waiting for buffer space to be freed by consumption and is not using the storage device either to seek or to read.

Merely providing free time, however, does not guarantee satisfactory results. It must be ensured that the free time is actually long enough to do some reading or writing for other tasks. The length of the reading period is also important, because it dictates how often this free time becomes available. Of course, increasing the free time must increase the length of the reading period, so again we are concerned about the length of free time.

Important as the length of free time per reading period is, the real indicator of how much sharing can occur is the fraction of free time per reading period. This value indicates how much of the storage device's time is really being freed for use by other tasks.

To redefine the model from the previous section, we first add some free time (t_f) to each reading period:

$$l_p = d_{\max} + \frac{n_c s_b}{r_t} + t_f. \tag{64}$$

This can be substituted into the equation for consumption rate, and solved for free time to obtain

$$t_f = s_b\left(\frac{1}{r_c} - \frac{n_c}{r_t}\right) - d_{\max}. \tag{65}$$

From this we can see that in order to increase the free time per reading period we can either increase the block size or transfer rate or decrease the consumption rate, number of channels, or delay time. For a given storage device, only the block size or delay time may be altered (by changing storage placement and playback strategies).

Now consider the fraction of time that the storage device is free. This is

$$\frac{t_f}{l_p} = 1 - r_c\left(\frac{n_c}{r_t} + \frac{d_{\max}}{s_b}\right). \tag{66}$$

This equation demonstrates that increasing the block size will increase the fraction of free time, as will decreasing the delay time. However, the growth of the fraction of free time follows the law of diminishing returns by altering these parameters. Therefore, the practical steps of adjusting the block size or delay time to increase the free time per reading period fails to yield useful results as the returns diminish.

6.2 The Channels Models

Another way to view the problem of sharing a storage device when an audio task is running is to consider all requests to the device as being for some channel of audio. An audio task could reserve some channels for playback use, while requests for nonaudio tasks would utilize any available nonreserved channels. It would be worthwhile to reserve some channels for exclusive nonaudio use in order to prevent starvation.

The first observation we can make from this simple model is that if all channels were audio channels the maximum number of channels would be $\lfloor r_t / r_c \rfloor$. This is because the total consumption rate cannot exceed the transfer rate of the storage device. However, because not all channels are for audio, this does not hold true. In fact, an arbitrary number of extra nonaudio channels can be supported. This is easy to see by considering the fact that the free-time length discussed above can be made arbitrarily large, supporting as many nonaudio reads as desired.

Freeing up this time to read nonaudio data will, of course, cost something, and the cost is increased block size for the audio channels. The following proof gives a lower bound on the extent of this cost:

THEOREM 10. *Consider a multitasking system using an even reading strategy for playback, with NPC channels of audio. Suppose that in each reading period n blocks of nonaudio data of size s_d are read. The audio block size must* *then be*

$$s_b \geq \frac{ns_d}{\frac{r_t}{r_c} - n_c}. \tag{67}$$

PROOF. An audio block must be long enough to supply data for the length of a reading period; that is, $s_b/r_c > l_p$. The reading period is spent reading audio blocks, reading data blocks, and incurring some delays. Ignoring the delays we have

$$\frac{s_b}{r_c} \geq \frac{n_c s_b + ns_d}{r_t}, \tag{68}$$

which yields

$$s_b \geq \frac{ns_d}{\frac{r_t}{r_c} - n_c}. \quad \square \tag{69}$$

The above result is useful in indicating how much of the storage device bandwidth should be used for audio tasks. We have already observed that we can have at most $\lfloor r_t/r_c \rfloor$ audio channels. Suppose that we have $n_c = (r_t/r_c) - x$ audio channels. The above result tells us that we would then need an audio block size that is at least the sum of all data block sizes divided by x; that is, $s_b \geq (ns_d/x)$. This means that the closer we come to using the maximum number of audio channels, the more costly it is to add other nonaudio channels.

For example, suppose that we used $(r_t/r_c) - 0.5$ audio channels; that is, the numer of audio channels is maximized, but there is still a small amount of bandwidth remaining. In this case, we would need an audio block size of $s_b \geq 2ns_d$, so that each audio block would have to be twice as large as the sum of all data blocks.

Suppose that we wish to keep our audio block size down to at most m, the size of a data block. Using the above results, we obtain $n_c \leq (r_t/r_c) - (n/m)$, or $n \leq m[(r_t/r_c) - n_c]$. That is, the number of nonaudio channels should not exceed m, the difference between the maximum number of audio channels and the actual number of audio channels.

We observe, then, that the greatest flexibility is obtained by not attempting to utilize fully the bandwidth of the storage device for audio. It was already demonstrated that for a dedicated storage device this is the most economical route. Sharing the storage device only exacerbates the problems associated with pushing the bandwidth limits.

7. STORAGE PLACEMENT STRATEGIES

This section considers several strategies for placing audio data on a storage device in order to play back multiple-channel audio synchronously. In Section 7.1 the possible placement strategies are categorized, and in Section 7.2 these categories are applied to some systems currently in use. Section 7.3 discusses issues particular to interleaving and indicates what sort of interleaving is best under certain conditions. Section 7.4 takes up the problem of placing logical blocks of audio in sectors on a storage device. Section 7.5 considers the impact of compression on placement.

7.1 Categories for Placement Strategies

Placement of data may be either interleaved or noninterleaved. An interleaved placement groups together data that are read at the same time from all of the channels. Notice that this implies that synchronization information is complete before placement and that changing synchronization involves moving the data on the storage device. Interleaving or noninterleaving may be combined with either a scattered or contiguous placement strategy. In a contiguous strategy, samples, in the order they will be read, are stored together in a contiguous fashion, one after another, on the storage device. In a scattered strategy, the data may be split up into blocks and placed in various locations on the storage device. The placement strategies discussed, then, are scattered interleaved, scattered, contiguous interleaved, and contiguous (a placement is assumed to be noninterleaved unless otherwise stated).

In *scattered placement*, data for each channel is split up into blocks. Each data block of a channel may lie anywhere on the storage device. In *contiguous placement*, all of the data for a particular channel is stored contiguously on the storage device, in the order it is read. In *contiguous interleaved placement*, it is assumed that several channels have already been synchronized. Instead of storing all of the data from one channel together as in the contiguous placement, interleaving places the data from all channels that are read at a particular time together. These groups of data to be read successively are then stored one after another, forming one large contiguous file. *Scattered interleaved placement* is the same, except that the large contiguous file is split up into pieces, which are then placed on the storage device as in the scattered placement.

There are various advantages and disadvantages to each strategy. Scattered placement allows more flexibility than contiguous placement, so that external fragmentation may be avoided. External fragmentation occurs when small gaps must be left on a storage device because no file is small enough to fill them. However, there is more overhead involved in keeping track of the locations of scattered blocks. In addition, continuous interleaved placements require much less seeking during playback than scattered interleaved placements.

Interleaving is used in an attempt to reduce seeking. By placing blocks that must be read at the same time side by side, minimum seeking between them is guaranteed. Note, however, that synchronization information must exist in order to do interleaving, and it cannot change without rewriting the data. Adjustment of synchronization is very flexible under a noninterleaved scheme. Another benefit of using noninterleaved placement is that when a certain sound is used several times it need only be recorded once on the storage device. With interleaved placement, it would have to be copied once for every time it is to be used.

With noninterleaved placement, we only need to consider the number of channels being played back. With interleaved data, we also need to consider the number of interleaved channels, which may be greater. We denote the number of interleaved channels by n_i and the number of playback channels by n_c. There are assumed to be at least two playback channels in an interleaved scheme, because otherwise interleaving would be meaningless.

7.2 Existing Placement Strategies

In this section some placement strategies that are currently in use are surveyed and identified according to the categories given in the previous section.

The best-known digital audio storage medium is the compact storage device (CD). As mentioned in the section reviewing digital audio, a CD is an optical storage device, which can hold about 72 minutes of stereo (two-channel) digital audio. Data is placed on the storage device in a spiral format, so that reading can be performed continuously, without any seeks. This spiral is divided into 2K sectors that contain the audio data and some additional error-correction data. Thus, CDs could be classified as using a contiguous interleaved placement strategy, with two channels of data.

A spin-off of the CD is compact storage device interactive (CD-I). The same physical storage device is used for this application, but data other than just audio are permitted, including machine instructions that allow user interaction. Of interest to this survey is that, besides including CD-type sound, CD-I also includes three other types, which are referred to as level A, B, or C sound. Level A uses about half the bandwidth of regular CD quality audio, level B about half that of level A, and level C about half that of level B. The bandwidth is cut by using a lower sampling rate and/or less bits to encode each sample. The CD-I standard allows for buffering of data (1-Mbyte system RAM) to be played back later, so the placement strategy is not fixed. However, Philips, a major player in developing the CD-I standard, indicates only interleaved data [8].

For systems using magnetic storage devices as the storage medium, little information regarding placement is available. However, Watkinson [10] indicates that in studio systems an attempt is made to keep the data contiguous where possible, although scattered placement may be necessary due to storage device fragmentation.

7.3 Options in Interleaving

When dealing with interleaved placement, the exact manner of interleaving must be considered along with the placement strategy. A contiguous section of the storage device containing data for each channel is read in each reading period, but the question remains; How is the data interleaved? One way of interleaving would be to store the data in such a way that the first sample is read for each channel, then the second sample for each channel, etc. Such an interleaving strategy is referred to here as *sample interleaving*, because interleaving occurs sample by sample. The other extreme would be collecting the data for each channel together, which is referred to as *channel interleaving*.

As mentioned in the previous section, a *block* is a unit of data read for a channel in a reading period. A *metablock* is the unit of data read for all channels in a reading period for interleaved strategies. Within a metablock, the data may or may not be divided up into blocks for each channel, depending on the interleaving strategy.

We now demonstrate the optimality of sample interleaving under certain conditions.

THEOREM 11. *For an interleaved placement strategy (scattered or contiguous) in which the number of playback channels is the same as the number of interleaved channels ($n_c = n_i$), it is optimal in terms of buffer space requirements and prefetching time to store the data in sample-interleaved fashion.*

PROOF. Suppose some other scheme of storing the interleaved data is optimal. Now compare the read functions for the optimal solution and for a sample-interleaved strategy. Let the read function and buffer function for the ith channel under the optimal solution be, respectively, $R_i(t)$ and $B_i(t, t_0)$. At any time t, a total of $\sum_{i=1}^{n_c} R_i(t)$ samples will have been read. Because sample interleaving distributes the reading evenly among the channels, at time t the amount read for any channel under the sample interleaving scheme is at least

$$\left\lfloor \frac{\sum_{i=1}^{n_c} R_i(t)}{n_c} \right\rfloor. \tag{70}$$

Therefore, because the buffer function is equal to the read function minus the consumption function, which is constant and the same for each channel, the buffer function for any channel under the sample-interleaving scheme must be at least

$$\left\lfloor \frac{\sum_{i=1}^{n_c} B_i(t, t_0)}{n_c} \right\rfloor. \tag{71}$$

Now consider any point in time t such that $B_i(t, t_0) \geq 0$ for all channels under the optimal scheme. At such a time, we must have

$$\left\lfloor \frac{\sum_{i=1}^{n_c} R_i(t)}{n_c} \right\rfloor \geq 0. \tag{72}$$

Therefore, all channels under the sample-interleaved scheme are nonzero in buffer space. Therefore, sample interleaving may be substituted as a playback solution with the same buffer space and prefetch time. □

This theorem is derived from the reasoning that, when seeks cannot be used to reduce reading time, sample interleaving is never any worse than any other interleaving strategy. However, in many cases seeking can be used effectively. If, for example, the number of playback channels is less than the number of interleaved channels, it may be worthwhile to divide the data into a block for each channel. One seek may then be performed for each playback channel, rather than many seeks to pass over unwanted data in any other interleaving scheme.

7.4 Placement within Sectors

This section discusses how placement is affected by sector size (recall that we use the term *sector* for a physical block, to avoid confusion with logical blocks). If the logical block size is such that it is an integer multiple of the sector size, then one may proceed straightforwardly. If this is not the case, then some difficulties arise. For example, suppose the desired logical block size is 1.5, the sector size. Two problems immediately present themselves:

(1) Because a block is the smallest unit to read, it is impossible to read just 1.5 blocks. Thus, the cost for reading the data may be higher. In general, when reading a logical block, at least $\lceil s_b / s_s \rceil$ sectors may need to be read.

(2) What is to be done with the extra 0.5 of a sector (assuming the logical block began on a sector boundary)?

We tackle the second problem first, as discussing it will make the properties of the first problem clear. If the logical block size is not an integer multiple of the sector size, then two strategies may be taken:

(1) The remainder of the sector may be simply left unused, so that each logical block is made up of an integral number of sectors. This is called a *padded* placement.

(2) No padding is added, and data from the next logical block is used to fill up the sector. This is called a *close-packed* placement.

Obviously, a close-packed placement uses the storage device space more effectively, so it is desirable if there are no negative drawbacks. In order to look at the implications of close packing, we consider interleaved and noninterleaved placements separately. In particular, we examine (1) the implications on reading time for a logical block and (2) buffer space utilization.

First consider close-packed, contiguous, noninterleaved placement. Two reading strategies are possible. The first strategy is to round up the logical block size to $\lceil s_b / s_s \rceil$ sectors. In this way, $\lceil s_b / s_s \rceil$ sectors are read for the channel in each reading period. Because this much must be read for any given block without close packing, there are no disadvantages for buffer space or reading time. Note that more data is being read than necessary, so artificial delays are introduced into the system while waiting for buffer space to become available. The second strategy is to read just as many sectors as are necessary in each reading period. Because the first strategy always reads too much, it is possible that after a certain amount of time the amount overbuffered will accumulate to the point where one (or more) less block(s) may be read in a reading period. The performance of the second strategy is even better than the first in terms of reading time and buffer space, because it is never ahead of the first in reading. Note, however, that, at least for the first logical block, it must read just as much as with the first strategy, so its peak buffer utilization and its worst-case reading time are the same. Therefore, although it may use less buffer space and take less reading time at some points during playback, it has peaks of reaching the same performance as the other strategy. In both cases, there would be no reason not to close pack the data.

The reasoning for scattered noninterleaved placement is identical, except that instead of two reading strategies there are actually two placement strategies. In the first strategy, groups of $\lceil s_b / s_s \rceil$ sectors are kept together in each scattered location; and in the second, at most that many are, whereas at some points it may be less. Again, there would be no reason not to close pack.

When dealing with interleaved placement, we must consider exactly what is to be close packed, because more than one channel is kept together. First, recall that sample interleaving was shown to be optimal in some cases. When sample interleaving is optimal and is employed, then playback is very much like playing back one channel, and the argument for noninterleaved data above may be used again to show that close packing is optimal. The only situations where sample interleaving would not be used is where it saves time to do seeks of individual channels within the interleaved record, and we assume in this situation that all of the data for each channel is kept separate within the metablock. In order to make the metablock an integral number of sectors, three approaches may be taken.

(1) The block size for each channel may be increased to an integral number of sectors, and hence, the metablock is also an integral number of sectors. This approach ensures that each block for each channel begins on a sector boundary, whereas the other two do not.

(2) Each channel may be increased just enough that the metablock is an integral number of sectors, whereas the individual channel blocks are not.

(3) Data belonging to the next metablock may be added to fill in the sector.

Approach (3) can be rejected immediately for scattered interleaved placement. By taking it, data for the next metablock may be widely separated, defeating the purpose of interleaving.

The first strategy spreads the blocks out a bit more than the second, which spreads them out more than the third. Having blocks spread out more may result in slightly higher seek (although still within the upper bounds given in this chapter).

When blocks are aligned on sector boundaries (approach (1)), reading $\lceil s_b/s_s \rceil$ sectors will always suffice for each channel. When blocks are not aligned, more blocks may need to be read. For example, suppose that each logical block consists of one-and-two-thirds sectors. The first logical block lies within the first and second sectors. The second logical block lies within the second, third, and fourth sectors, so three sectors must be read to retrieve it. In general, nonalignment may mean that $\lceil s_b/s_s \rceil + 1$ sectors need to be read for each channel. Therefore, alignment of each block on physical boundaries may cost slightly more in seek time, whereas nonalignment may cost slightly more in reading time and may require more buffer space.

Finally, using padding in any of the three cases would not improve performance in any way. We can conclude that close packing is optimal in all cases.

7.5 Effects of Compression on Placement

It is very common to use compression schemes on audio data. A compression scheme attempts to reduce the amount of data required to store some information by translating to a different form of encoding. In this section we examine the impact of compression on digital audio storage and retrieval.

7.5.1 *Compression and System Resolution.* Let us begin by defining the resolution of a delay-sensitive playback system as follows: The resolution of an audio playback system is the smallest amount of data that can be read and played back independently of the rest of the data.

A compression scheme should be chosen so as to not limit the resolution of the system. For example, it is undesirable if the compression scheme worked on the entire file, such that it needed to be read in entirety in order to be decompressed. This would mean that the entire file must be read no matter how little is desired for playback, and also that buffer space must have space for the whole file. An example is a simple "delta" compression algorithm, in which actual samples are not stored; rather, the difference of each sample from its predicted value, based on previous samples, is stored. If delta compression is employed across an entire file, reading would always have to proceed from the beginning of the file, even if only the last tenth is to be played back.

For reasons of resolution, it is desirable to treat each logical block as a file in its own right and to employ compression on it separately. However, general compression algorithms tend to perform better on larger blocks of data. Usually typical sector sizes will suffice [13]; so if compression is applied to either the sector or logical block, whichever is larger, compression performance should be good, and system resolution should be as high as possible.

A remark on the application to video is in order at this point. Many video compression schemes hinge on the calculation of differences from frame to frame of the same pixel or region. From our study in this paper, we might think of each region as a channel in a multichannel synchronized system. The conclusions for multichannel audio storage and retrieval strategies may then be applied.

7.5.2 *Compression, Block Size, and Close Packing.* We have already seen that compression should be performed on small units of data in order to ensure good system resolution. However, this does not tell us much about storage of compressed data. For example, suppose a logical block takes up exactly one sector and is compressed to 0.8 of a sector. Unless some form of close packing is employed, no savings in either storage device space or performance would be realized, because an entire sector must still be occupied and read for this logical block.

In terms of playback, there is no reason not to close-pack noninterleaved compressed data, just as for the noncompressed case. Compressed logical blocks should be placed one after another, without regard for sector boundaries. Note, however, that close packing of compressed data may lead to difficulties in an editing system. This is because when a logical block is edited it may no longer be the same size and may not fit in where the old block was close packed. In the worst case, this could lead to rewriting the entire file. However, we restrict our discussion here to playback-only systems.

In the case of interleaved placements, the options are more limited for compressed data than for noncompressed data. Because compression rates vary, it would be very difficult to extend the length of each block so as to make them all fill an integral number of sectors or so that the length of the metablock was an integral number of sectors. Three choices are then possible.

(1) Pad every block for every channel to make it an integral number of sectors.
(2) Close pack blocks within each metablock, and then pad each metablock to make it an integral number of sectors.
(3) Close pack blocks within each metablock, and close pack metablocks.

For contiguous interleaved data, there would be no reason not to close pack the metablocks. This would ensure that storage-space savings are realized. However, for scattered interleaved close packing, metablocks would be unacceptable because it would widely separate the data within metablocks, defeating the purpose of interleaving. For scattered interleaved data, approach (2) should be taken, because it will never waste any more space than approach (1), and may do better.

In conclusion, close packing should be employed for all compressed data except scattered interleaved, which should combine close packing and padding, as indicated by approach (2) above.

8. SUMMARY

The real-time requirements of digital audio playback are significantly different from the requirements of most "real-time" databases. In order to have a successful playback, the retrieval of successive samples must meet successive real-time deadlines.

In Section 3 we have developed a general theoretical framework for studying the performance of any playback system. We have demonstrated that playback can always be achieved, by buffering all of the samples. Knowing that playback can be done, the problem is then to do it with the smallest buffer space and start time. It was proved that for single-channel playback finding the minimum start time is equivalent to finding the minimum buffer space. Given the function that describes how many samples had been read from the storage device at a given time and the consumption rate of the samples by the digital-to-analog converter, a method for finding the minimum start time (and, hence, the minimum buffer space) has been given.

We have also studied the effects of reading data in sector units, and how to use the previous results in such a case. Reading data in sectors usually means that buffer space must be allocated before a read begins and that data from the sector is not available until the entire sector is read. All of the following results assumed that reading was performed on sectors.

The results of Section 3 form a set of design tools that can be used by multimedia system designers to study the effect of alternative designs and hardware selection decisions (such as data placement, compression schemes, devices with faster seek or transfer times) on system performance.

In Section 4 several special cases have been examined, and simpler results for finding minimum buffer space have been obtained. Section 4.1 gave bounds on buffering requirements for the case of single-channel uncompressed audio that can be read continuously. It was found that, unless the consumption rate is equal to the transfer rate of the device, the buffer requirements are dependent on the length of the audio record.

Having buffer requirements dependent on the length of the audio record is not desirable in most cases, and Section 4.2 has examined how this could be avoided by introducing artificial delays. A lower bound was found for buffer requirements, and an algorithm was demonstrated that had buffer requirements very close to the bound. Section 4.3 has extended this work to cover the case in which some unavoidable delays occur in addition to artificial delays introduced by the algorithm. Again, a lower bound was produced, and an algorithm with buffer requirements approaching the bound was demonstrated.

Section 5 has presented the properties of multichannel playback in order to illustrate some important relationships. It was observed that block size is indicative of buffer space requirements, while reading period length is indicative of start time delays. It was found that when nonzero seeks are encountered, block size and reading period length asymptotically approach infinity as the total consumption rate approaches the transfer rate of the storage device. Because of this, it is advisable to have some "headroom," with the transfer rate of the storage device being significantly higher than the total consumption rate. It was shown that, in order to support greater seek time, one may either decrease the consumption rate of each channel of audio, increase the block size read in each reading period, or increase the data transfer rate of the storage device.

In Section 6 these results have been extended in order to consider the case of a shared storage device. Two models have been presented to consider the special needs of a shared system. The first considered how an audio task might allow free time in its reading cycle for other tasks. It was shown that the absolute length of free time per reading period may be increased arbitrarily, but that it is more difficult to increase the percentage of free time. The second model considered all requests for data on the storage device to belong to a channel of audio playback. This model has been useful in illustrating the effects of bandwidth utilization, and similar to Section 5, it was found that it is not economical to push the bandwidth limits of the device too far.

Section 7 took up the topic of storage placement strategies for multichannel synchronized playback. Strategies were categorized as either being interleaved or noninterleaved, and contiguous and scattered. Interleaved and contiguous strategies achieved higher performance at the expense of flexibility and disk fragmentation. The problem of placing logical blocks of data in physical blocks was then analyzed for both compressed and noncompressed data.

Because the basic real-time requirements of digital audio playback are entirely analogous to other data types such as video and animations, the results presented here extend to delay-sensitive data in general. The basic principles for retrieving and storing delay-sensitive multimedia data have been described in this paper. These principles can be used by a system designer to estimate hardware requirements and to evaluate design strategies.

REFERENCES

1. Abbot, C. Efficient editing of digital sound on disk. *J. Audio Eng. Soc. 32*, 6 (June 1984), 394-402.
2. Anazawa, T., Hayashi, H., Inokuchi, K., Takahashi, Y., Takasu, A., Yamamoto, K., Todoroki, S., and Yazawa, H. A historical overview of the developments of PCM/digital recording technology at Denon. Presented at the AES 7th International Conference of the Audio Engineering Society (Toronto, Ontario, May 14-17, 1989).
3. Anderson, D. P., Govindan, R., and Homsy, G. Abstractions for continuous media in a network window system. In *Proceedings of the International Conference on Multimedia Information Systems '91* (Singapore, Jan. 16-18, 1991). McGraw-Hill, New York, 1991, pp. 273-298.
4. Anderson, D. P., Tzou, S., Wahbe, R., Govindan, R., and Andrews, M. Support for continuous media in the DASH system. In *Proceedings of the 10th International Conference on Distributed Computing Systems* (Paris, May 28-June 1, 1990). pp. 54-61.
5. Christodoulakis, S., Ho, F., and Theodoridou, M. The multimedia object presentation manager of MINOS: A symmetric approach. In *ACM Proceedings of SIGMOD.* ACM, New York, 1986, pp. 295-310.
6. Nakajima, H., Doi, T., Fukuda, J., and Iga, A. *Digital Audio Technology.* Tab Books, Blue Ridge Summit, Pa., 1983.
7. Park, A., and English, P. A variable rate strategy for retrieving audio data from secondary storage. In *Proceedings of the International Conference on Multimedia Information Systems '91* (Singapore, Jan. 16-18, 1991). McGraw-Hill, New York, 1991, pp. 135-146.
8. Philips International. *Compact Disk Interactive: A Designer's Overview.* Donnelley and Sons, Harrisonburg, Va., 1988.

9. THOMPSON, T., AND BARAN, N. The NeXT computer. *Byte 13*, 12 (Nov. 1988), 158–175.
10. WATKINSON, J. Digital audio recorders. *J. Audio Eng. Soc. 36*, 6 (June 1988), 501–508.
11. WATKINSON, J. *The Art of Digital Audio* Focal Press, Boston, Mass , 1988
12. WELLS, J., YANG, Q., AND YU, C. Placement of audio data on optical disks. In *Proceedings of the International Conference on Multimedia Information Systems '91* (Singapore, Jan. 16–18, 1991). McGraw-Hill, New York, 1991, pp 123–134
13. WILKES, D. Data compression for randomly addressed files. Master's thesis, Dept. of Computer Science, Univ. of Toronto, Ontario, 1985.
14. YAMAMOTO, T. Optical recording technology: Can optical disks be applicable to digital audio workstations? Presented at the AES 7th International Conference of the Audio Engineering Society (Toronto, Ontario, May 14–17, 1989).
15 YU, C., SUN, W., BITTON, D., YANG, Q , BRUNO, R., AND TULLIS, J. Efficient placement of audio data on optical disks for real-time applications. *Commun. ACM 32*, 7 (July 1989), 862–871.

Received June 1990; revised February 1991; accepted July 1991