Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Design and Analysis of an Efficient Buffer Scheduling Scheme for VOD System*

Y. C. Chen and S. C. Rong
Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan, ROC

ABSTRACT

Due to the fast advance in communication and computer system technologies in 1990s, multimedia communication applications emerge rapidly. A popular example is the video-on-demand (VOD), which is perceived as a highly marketable service for entertainment. In this paper we discuss the design and analysis of an efficient buffer scheduling scheme for VOD system. Our scheme uses a round-robin like buffer scheduling approach combining with a disk scheduling method–Scan, which greatly reduces the buffer requirement up to a half comparing with solely Scan if the number of video streams is large. Using our proposed scheme, it is able to serve more subscribers compared with either Scan or round-robin method under the same system configuration. We also prototype a VOD system based on our scheme for performance evaluation.

1. Introduction

Due to the fast advance in communication and computer technologies in 1990s, multimedia communication applications emerge rapidly. A popular example is the video-on-demand (VOD), which is perceived as a highly marketable service for entertainment. In this paper we discuss the design and analysis of an efficient buffer scheduling scheme for VOD system. Our scheme uses a RR (round-robin)[2] like buffer scheduling approach combining with a disk scheduling method–Scan, which greatly reduces the buffer requirement up to a half if the number of subscribed streams is large.

A video server has to perform a number of functions[10,13] such as admission control, real-time data retrieval, disk scheduling, and stream-oriented data transmission, as well as support these functions found in a VCR(video cassette recorder). Since the video data needs to be delivered to the viewers with minimized jitters, the bursty nature of disk accesses mandates that intermediate buffer memory is required to transform the bursty data into a continuous stream, which must be guaranteed to have well controlled jitters.

We have prototyped a VOD system based on a real-time kernel[4,5,6,14] and the UDP protocol[15,16] using our proposed scheme. The system is shown to support 3 clients simultaneously, which already reaches the hardware limitation (PC486 DX2-66, IDE adapter, ISA Bus, Ethernet)[12]. Our design should be able to support more streams if both disk data-transfer rate and network transmission rate are higher.

In Section 2, we present our disk scheduling scheme. In Section 3, the mathematical analysis of our scheme is discussed. Section 4 addresses the performance measurement of the system prototype, and Section 5 concludes this work.

2. Design Approach

Our video server consists of three main modules (tasks): *Admission*, *Producer*, and *Consumer*. Assuming k streams are to be supported, we have to prepare a $k+1^{th}$ buffer, called *extra buffer* in addition to k buffers to perform our buffer scheduling algorithm. Before we proceed, some terminologies are defined:

(1) **Data-block:** Amount of data that the video server retrieves for each stream during a round, here we let the buffer size be identical to the data-block size.
(2) **Disk service time:** This is the data transfer time plus the disk-access overhead such as seek-time and rotation latency.
(3) **Round and round-length:** The process that a video server retrieves a set of data-blocks for all subscribed

* This work is supported in part by Computer and Communications Lab., ITRI under contract no. K4-84021

streams is called "a round". The time duration of a round, including the operating system overhead and disk service time, is called the round-length.

(4) **Stream retrieve latency(SRL):** This is the time interval between two consecutive requests to read data.

(5) **Stream send latency(SSL):** It is the time interval between two consecutive requests to send data-blocks.

A disk scheduling algorithm decides how to retrieve video data for each stream. Data-placement algorithms[3,8] that inherently reduce latencies are used in conjunction with disk scheduling algorithm. To ease the implementation, continuous data-placement is used for each stream. Although Scan is considered more efficient than RR, however, if we also consider buffer scheduling, there exists a trade-off between the disk overhead (and the round length) and the required buffer size. In the following, we will examine this trade-off issue.

2.1 Disk Scheduling Algorithms – RR and Scan

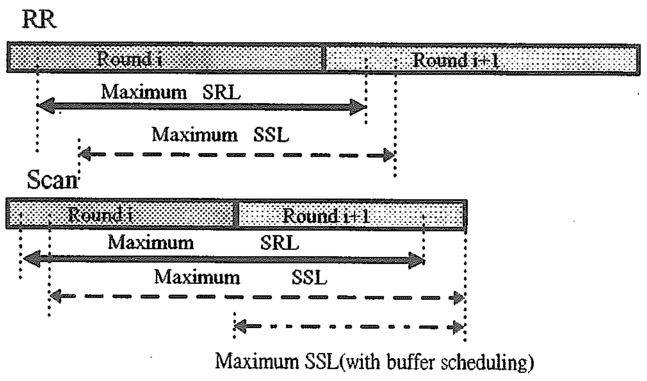

Figure 1. SRL and SSL in RR and Scan.

Both RR and Scan can reduce seek time[7]. The Scan is considered more efficient than RR because Scan has less frequent disk head movement. On the other hand, the order to service video streams is fixed during each RR round, thus its SRL is upper-bounded by one round-length. While with Scan method, the order is changing alternatively for each round, thus SRL is upper-bounded by two round-lengths. Generally SSL has the same value as SRL with the same disk scheduling algorithm(see Figure 1). So SSL is upper-bounded by two round-lengths using Scan, which means that Scan needs a double-sized buffer for each subscribed stream to satisfy the consuming need of nearly two rounds; while RR only needs a single-sized buffer in the same situation. To resolve this trade-off issue, several algorithms have been proposed. Among them, the grouped sweeping scheme (GSS) is the most popular one. In GSS, each round is partitioned into groups which may contain more than one stream. These groups use RR internally, while the Scan method is used external to groups. By optimally deriving the number of groups, the server can balance the cutback of round length against data-block size.

2.2 Scan with Proposed Buffer Scheduling Algorithm

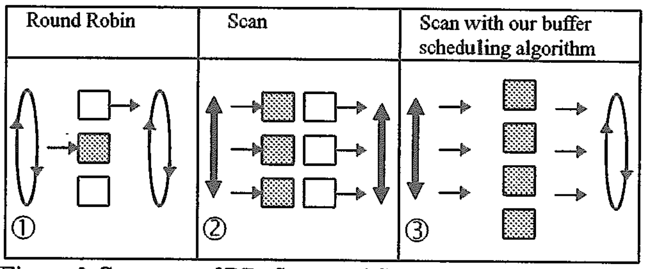

Figure 2. Concepts of RR, Scan and Scan and our algorithm.

Let the number of streams be k. In RR, the video server just needs k buffers, while in Scan, the video server needs $2k$ buffers. Since the SRL and SSL of Scan are bounded by two rounds, video data transmission must start at the end of each round, otherwise the client buffer will be starving. Since the order of buffer filling is always same as that for buffer sending, SSL is bounded by SRL (see ①,② of Figure 2). What we try to improve is reducing SSL to only one round while still keeping SRL unchanged in Scan. Our algorithm reduces the required number of buffers from $2k$ to $k+1$.(see ③ of Figure 2). The key idea is that data sending speed can be controlled according to data reading speed so that we need only to provide an extra buffer in addition to k buffers. During each round, we know for each stream

Server throughput = Data-block size/Round-length

and from the previous section, we have

Round-length = Disk service time + OS overhead.

Here OS overhead includes interrupt latency and scheduling latency.

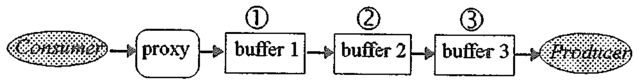

Figure. 3. The operation of the proxy.

In our design, each buffer is associated with a unique proxy. By triggering the proxy attached to an empty buffer, the *Consumer* can tell the *Producer* which buffer is empty and can be filled again. Figure 3 shows that *Consumer* freed three buffers (in the sequence of ③,②,①) and triggered their corresponding proxies, *Producer* can pick up an empty buffer through the proxy ID it received.

2.3 Task Priority and Synchronization

We assign the highest, the lower and the lowest priority to tasks *Admission*, *Producer*, and *Consumer* respectively. When *Producer* issues a *fread*( ) system call, it enters an event-wait state, then the kernel selects *Consumer* to run if it is in ready state. *Consumer* issues a non-blocking *sendto*( ) system call to transmit data. Once a buffer has been emptied during a round, *Consumer* issues a non-blocking *trigger*( ) to send a proxy to *Producer*, which may be unblocked either before *Consumer* empties a buffer and sends a proxy, or after. If the former happens, the kernel preempts *Consumer* and run *Producer*, and *Consumer* will be put into ready queue. If no free buffer is available, *Producer* will enter the message-wait state waiting for *Consumer*, otherwise *Producer* can service the next stream. If the latter happens, *Producer* reads data into the buffer that *Consumer* has freed. *Admission* is responsible for receiving commands from clients. The disk seeking order will be re-computed once a new request is admitted or an existing stream is terminated.

There are three entities in our video server: File ID, Stream ID, and Buffer ID which define four mapping relations (see Figure 4). Among them, admission process will setup mapping relation between the video file ID and the stream ID (FileToStream) immediately after it receives an initialization command from a client. In addition to be a mapping relation, FileToStream also acts as a flag. With the mapping relation, *Producer* can tell whether a video file is subscribed. Initially, the relation (table) entries of FileToStream are set to NOC (a flag, which means "No Clients"), while *Producer* does nothing but checks the table entries. As soon as a play command is received, *Admission* will signal *Producer* to start reading the video file by setting its associated FileToStream entry to the stream ID. *Producer* will retrieve data from disk as soon as the FileToStream flag of a video file is set. Meanwhile, *Admission* also handle further requests from other clients.

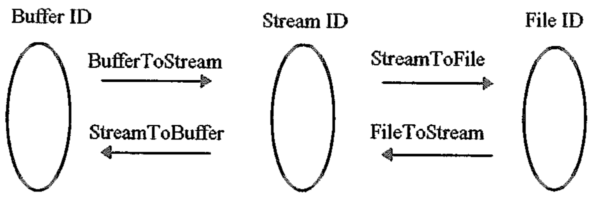

Figure 4. Mapping relations between different entities

2.4 Proposed Buffer Scheduling Algorithm

Figure 5 shows our proposed buffer scheduling algorithm, and its related block diagram is shown in Figure 6. If k clients connect to the server simultaneously, in the first round, there are $k+1$ empty buffers, and *Producer* fills up k buffers. In the next round, *Producer* can always find an empty buffer (extra buffer) to fill. Since consuming rate = $S_{db}/(T_{ps}+T_{nt})$, where S_{db} is the data-block size, T_{ps} is the time to segment a data-block into packets (packetization) and move them to the network adapter, and T_{nt} is the transmission time of a data-block in the network interface.

```
Initially. trigger all the proxicies associated with their buffers respectively.
Producer :
for each round
{  for each file i subscribed   /*Producer loops file IDs using Scan*/
     find stream ID j associated with file i   /*j=FileToStream[i]*/
   wait for  the proxy ID k associated with an empty buffer k
   mark StreamToBuffer[j]=k
   mark BufferToStream[k]=j
   read file i into buffer k  }
send the message to Consumer   /* at the end of one round */
Consumer :
wait message from Producer and reply it
for each stream i that has data in buffer /*Consumer loops stream
                                                IDs using RR */
{  find Buffer ID j associated with buffer i  /* j=StreamToBuffer[i]*/
   send content of buffer j to the client of stream i
   mark BufferToStream[j]=Empty
   mark StreamToBuffer[I]=Empty
   trigger proxy ID j associated with buffer i to Producer }
```

Figure. 5 Proposed buffer scheduling algorithm

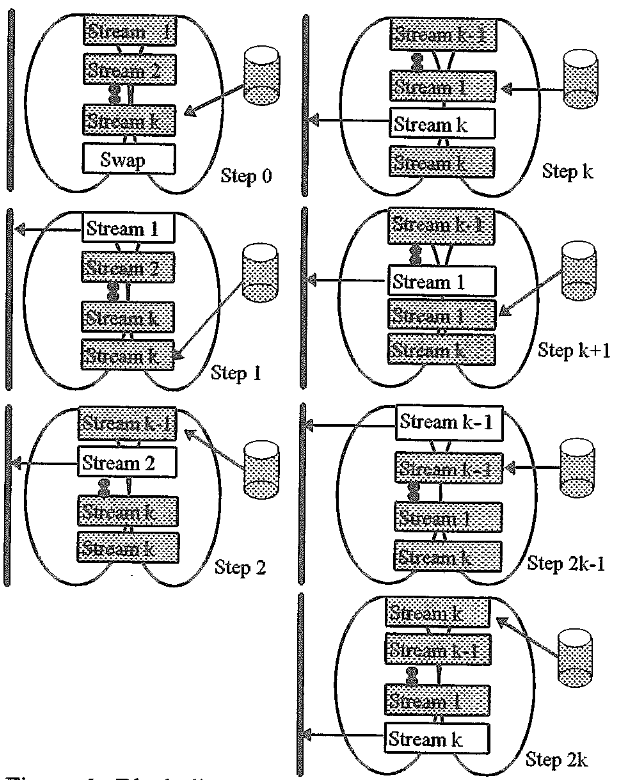

Figure 6. Block diagram of proposed buffer scheduling algorithm.

We can select a proper data-block size and compute its T_{ps} and T_{nt} so that $T_{ps}+T_{nt}$ is smaller than the reading time of a data-block, then the consuming rate will be faster than the producing rate. *Consumer* will have already released a buffer of the first stream in the previous round before *Producer* fills out the extra buffer, and *Producer* can select this released buffer as the next empty buffer to fill the data. In normal situation, it is very unlikely that there is no empty buffer available for *Producer*. Even though, it still can wait until a proxy from *Consumer* is received. This may occur only when the network traffic load is very high which in turn causes a relatively small consuming rate.

3. Mathematical Analysis

Symbol	Explanation	Unit
R_{vr}	Video playback rate	frames/sec
R_{dr}	Disk data transfer rate	Kbyte/sec
S_{db}	Size of a data-block	Kbyte
S_{vf}	Average size of a video frame	Kbyte/frame
S_{tk}	Size of a track	Kbyte/track
τ	Service time per round	second
T	Number of tracks	tracks
$l_{min\text{-}seek}$	Track-to-track seek time	second
l_{rot}	Rotation latency	second
$l_{max\text{-}seek}$	Maximum seek time	second

Table 1 Symbol table.

In this section, we perform the mathematical analysis of disk scheduling, also we derive equations to compute the maximum number of the video streams that a video server can support based on our buffer scheduling scheme. Table 1 is the symbol table used in our discussion.

3.1 Disk Characteristic Curve

The effective transfer rate of a hard disk is about 1-3 MByte/s (IDE) or 4-6 MByte/s (SCSI), this is much higher than the average playback rate of a single compressed video stream, e.g., 0.2 MByte/s for MPEG-1[9], and 0.42MByte/s for MPEG-2[18]. Suppose that the video server retrieve data blocks for n stream. Let $R_{vp1}, R_{vp2}, \ldots, R_{vpn}$ be their playback rates, and $S_1, S_2 \ldots$, and S_n be data-blocks for these streams respectively. Also assume that the disk storage space is divided into T tracks, with each track being able to hold at most B data-blocks. To simply the analysis, we approximate the seek time by :

$$l_{seek}(t) \approx a + b \times t \qquad (1)$$

where t is the distance in terms of tracks, a and b are constants, and $l_{seek}(t)$ denotes the seek-latency for the disk head to move across t tracks. Since the real curve should be nonlinear[17], we call Equation (1) *disk characteristic curve*. Let $l_{max\text{-}seek}$ denotes the seek time that the disk head moves from the innermost track to the outermost track or vice versa, and l_{rot} denotes the rotation latency. From Equation (1), we know:

$$l_{max\text{-}seek} = l_{seek}(T) \approx a + b \times T \qquad (2)$$

$$l_{min\text{-}seek} = l_{seek}(1) \approx a + b \qquad (3)$$

3.2 Single Stream Analysis

If the size S_{db}^i of the data-block S_i is retrieved during each round, then its retrieval time will depend on the number of tracks spanned by this data-block, because the video server stores successive data-blocks of a stream into adjacent tracks. Since the size of each track on the disk is S_{tk}, the data-block of S_i can span at most ($\lceil S_{db}^i / S_{tk} \rceil + 1$) tracks. Once the disk head is positioned on the first track of the data-block to be read, it may have to move to an adjacent track at most $\lceil S_{db}^i / S_{tk} \rceil$ times for reading the data, each move costs a seek latency of $l_{\min-seek}$. The latency required for locating a data-block from a track is bounded by l_{rot}, and the total transfer time of a data-block is S_{db}^i / R_{dr}. Hence, using Scan algorithm, the data-block of stream S_i can be retrieved with the upper bound:

$$\left\lceil S_{db}^i \big/ S_{tk} \right\rceil \times l_{\min-seek} + \left(\left\lceil S_{db}^i \big/ S_{tk} \right\rceil + 1 \right) \times l_{rot} + \frac{S_{db}^i}{R_{dr}} \qquad (4)$$

3.3 Multiple Streams Analysis

3.3.1 Seek Time

Assume that the video server is serving n streams, then for each round, the video server will retrieve n data-blocks for the n streams in one sweep. During a sweep, the disk head has to be repositioned at least n times, once for switching to a different data-block. Assume that each reposition causes the disk head to move t_i tracks. From Equation (4), these n moves cost seek time l_{seek}^1:

$$l_{seek}^1 = a \cdot n + b \cdot \sum_{i=1}^{n} t_i \qquad (5)$$

Furthermore, once the disk head is positioned on the first track of the data-block for a stream S_i, it may have to move at most $\lceil S_{db}^i / S_{tk} \rceil$ adjacent tracks in order to read all the data of this data-block. From Equation (1), this costs time $l_{min\text{-}seek} \times \lceil S_{db}^i / S_{tk} \rceil$, thus retrieving all n data-blocks for n streams costs $l_{\min-seek} \times \sum_{i=1}^{n} \left\lceil S_{db}^i \big/ S_{tk} \right\rceil$. Applying Equation (3), we know that the retrieval of all n data-blocks needs maximum seek time l_{seek}^2:

$$l_{seek}^2 \le (a + b) \times \sum_{i=1}^{n} \left\lceil \frac{S_{db}^i}{S_{tk}} \right\rceil . \qquad (6)$$

The total seek latency incurred for each round is bounded by the value of $l_{seek}^1 + l_{seek}^2$, which is equal to l_{seek}^{all}:

$$l_{seek}^{all} \le a \cdot n + b \cdot \sum_{i=1}^{n} t_i + (a + b) \cdot \sum_{i=1}^{n} \left\lceil \frac{S_{db}^i}{S_{tk}} \right\rceil ,$$

$$\Rightarrow l_{seek}^{all} \le a \cdot (n + \sum_{i=1}^{n} \left\lceil {S_{db}^{i}}/{S_{tk}} \right\rceil) + b \cdot (\sum_{i=1}^{n} t_i + \sum_{i=1}^{n} \left\lceil {S_{db}^{i}}/{S_{tk}} \right\rceil)$$
$$\le a \times (n + n) + b \times (T + n),$$
$$= (2a + b)n + bT \,. \quad (7)$$

Here we assume that the value of $\lceil S_{db}^{i}/S_{tk} \rceil$ is no greater than one because a reasonable data-block size will not be greater than the track size, as we can see in Section 4, thus the number of single-track moves for reading a data-block should be no more than one.

3.3.2 Rotation latency and data transfer time

Since S_{db}^{i} of data-block S_i may span at most ($\lceil S_{db}^{i}/S_{tk} \rceil$ +1) tracks, the total latency incurred while accessing data-blocks of all the streams in a round is thus bounded by

$$Rot_{all} = l_{rot} \cdot \sum_{i=1}^{n} \left(\left\lceil \frac{S_{db}^{i}}{S_{tk}} \right\rceil + 1 \right). \quad (8)$$

The total data transfer time of these n data-block is

$$Tx_{all} = \sum_{i=1}^{n} \frac{S_{db}^{i}}{R_{dr}} \quad (9)$$

3.3.3 Disk service time

Finally, using Scan, the disk head will stay in the final position of current round until the next round starts. At the beginning of the next round, data-block S_n, which is positioned at either the outermost or the innermost track among all subscribed video files, will become the first data-block to be read immediately. So during each round, the total service time τ is bounded by the maximum value of $l_{seek}^{all} + Rot_{all} + Tx_{all}$:

$$\tau \le (2a + b) \times n + b \times T + 2n \times l_{rot} + \sum_{i=1}^{n} \frac{S_{db}^{i}}{R_{dr}}. \quad (10)$$

3.3.4 Maximum number of allowed streams

During a round, the video server may retrieve n data-blocks for all n streams, and one round length can not exceed the minimum playback duration among $S_{db}^{1}, S_{db}^{2} \ldots$, and S_{db}^{n}. With Scan disk scheduling method, we get

$$(2a + b)n + bT + 2n \times l_{rot} + \sum_{i=1}^{n} \frac{S_{db}^{i}}{R_{dr}} \le \min_{i \in [1,n]} \left(\frac{S_{db}^{i}}{R_{vp} \cdot S_{vf}} \right) \quad (11)$$

If we can determine the value of $S_{db}^{1}, S_{db}^{2} \ldots$, and S_{db}^{n}, then we can calculate the maximum value of n by solving the Equation (11). The simplest way is to set $S_{db}^{1} = S_{db}^{2} = \ldots = S_{db}^{n} = S_{db}$, then we get:

$$(2a + b)n + bT + 2nl_{rot} + n \cdot \frac{S_{db}}{R_{dr}} \le \left(\frac{S_{db}}{R_{vp} \cdot S_{vf}} \right),$$

$$n \le \left(\frac{S_{db}}{R_{vp} \cdot S_{vf}} - b \cdot T \right) / \left(2a + b + 2l_{rot} + \frac{S_{db}}{R_{dr}} \right). \quad (12)$$

4. Performance Measurement

The hard disk used in our video server is Fujitsu M2622T. Its characteristics are listed in Table 2. Here we use the Norton Utility *SYSINFO* to get the information of the seek time and transfer rate. From Equation (5) and (6), we know that $l_{seek}(1)$ is the track to track seek time for adjacent tracks and $l_{seek}(T)$ is the maximum seek time, thus we get:

$$\begin{cases} a + b = 3.84 \\ a + 849\, b = 11.66 \end{cases}$$

After solving above joint equation, we get $a = 3.831$ and $b = 0.009$. Since disk drives cannot be accurately modeled analytically[17], we can only calculate the seek time through approximation.

Capacity (MByte)	267
Tracks	849
Heads	10
Sectors	63
Track Size (KB)	314.4
Sector Size (KB)	0.5
Peak Transfer Rate (KB/s)	1214.8
Avg. Transfer Rate (KB/s)	772.41
Spin Rate (rpm)	4500
Maximum Seek Time (ms)	11.66
Track to Track Seek Time (ms)	3.84
Maximum Rotation Latency (ms)	13

Table 2. Hard disk characteristics of our video server.

Data size	Transfer Time(sec)	Transfer rate(Kbyte/sec)
100	0.136156	734.449447
200	0.278144	719.052821
300	0.382134	785.065831
400	0.506122	790.323180
500	0.600113	833.176938
Avg. transfer rate (Kbyte/s)		772.413643

Table 3. The test result of average disk transfer rate

We know that the actual average disk transfer rate (tested in application level) is lower than the peak transfer rate (tested with the BIOS int13h) due to OS overhead and the data locality. We measure our average disk transfer rate on the QNX operating system by reading data of different sizes as shown in Table 3.

4.1 Estimation of the Maximum Number of Streams

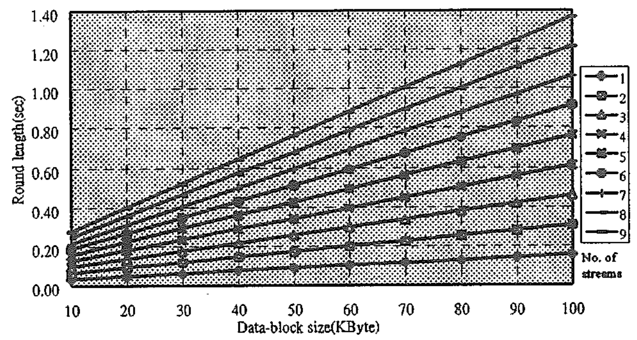

Figure 7. Round length v.s. different data-block sizes and number of streams.

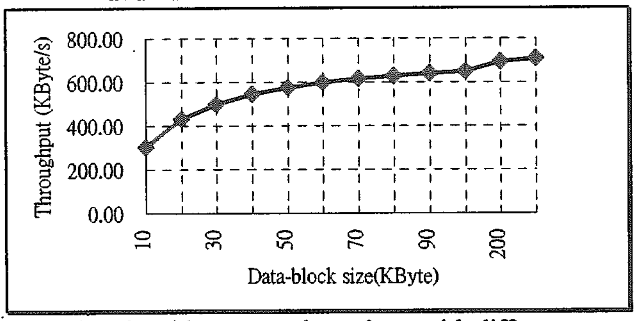

Figure 8. The video server throughput with different data-block size.

First, we substitute the parameters measured in the above section into Equation (10), and find out the relation between the round length and data-block size, as illustrated in Figure 7. The round-length grows with the data-block size. This is because the larger the data-block, the more time it will spend to retrieve data. Second, we divide the data-block size by round-length and draw the relation between the throughput and data-block size in Figure 8. We can see when the data-block size is smaller than 100KByte, the throughput grows quickly with increasing data-block size, while it grows slowly when the data-block size is larger than 100KByte. We say that 100KByte is the saturation point here and it is used as our data-block size.

Data-Block size(KB) / Frame-Rate (frame/s)	10	30	50	60	70	80	90	100	200	300
12	4	6	7	8	8	8	8	8	9	9
15	3	5	6	6	6	6	6	7	7	7
18	2	3	4	5	5	5	5	5	6	6
20	2	4	4	4	5	5	5	5	5	5
22	2	3	4	4	4	4	4	4	5	5
30	1	2	3	3	3	3	3	3	3	3

Table 5. The maximum number of streams with different frame rate requirements.

We can calculate the minimum playback time periods with different frame rate requirements and data-block sizes by Equation (11). From Equation (12), we can figure out the maximum number of streams in our server with different data-block sizes, which are listed in Table 5. The relationship between these parameters are shown in the Figure 9, from which we can find that our buffer scheduling algorithm is able to support more than 12 streams if effective disk transfer rate is higher than 3 MByte/s.

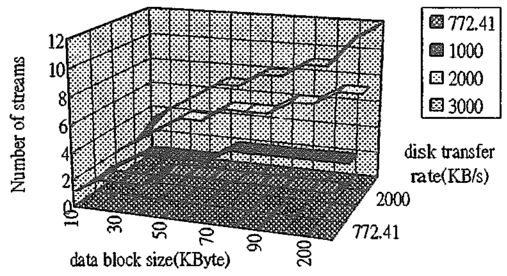

Figure 9. The maximum number of streams with different parameters.

4.2 Measurement of SSL and SRL

We will compare our buffer scheduling algorithm with general Scan disk scheduling algorithm, all tests performed in this section proceed with three streams.

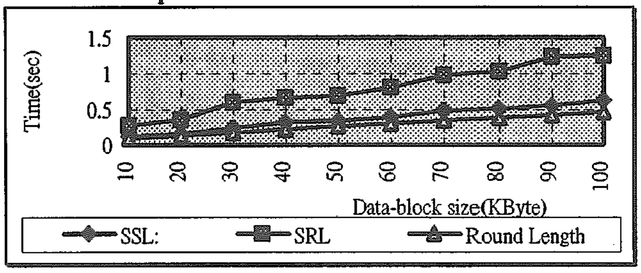

Figure 10. Comparison of round length with SSL and SRL using proposed buffer scheduling and Scan.

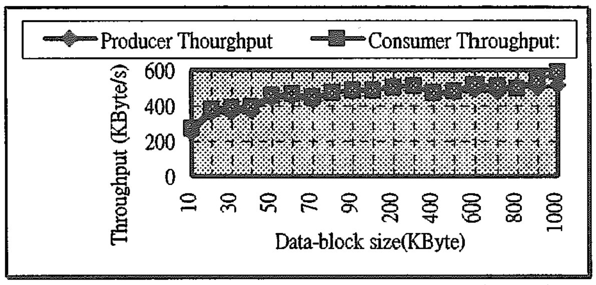

Figure 11. The producer and consumer throughput.

Regarding the SRL and SSL in our system, Figure 10 shows that the SRL is about two round-lengths and SSL is almost equal to the round-length with three streams. This indicates that our buffer scheduling algorithm successfully reduces SSL from two round-lengths to one when it is combined with the Scan. Next, we will examine the throughput of our system.

The throughput can be measured in two aspects. The producer throughput is measured by dividing the total data read in each round by a half of SRL, and the consumer throughput is measured by dividing the total data sent in each round by SSL(one round length). Figure 11 shows that these two throughputs are almost equal, except few differences caused by the slack time[1] of IPC. Here we use the producer throughput as our video server throughput.

In Figure 12, we compare the calculated throughput with the measured throughput. We find that these two curves increase rapidly with increasing data-block size. Our video server throughput is smaller than the theoretical value due to the overhead in the real system, for example, access to the client information data base, performing buffer scheduling algorithm, and checking out the status of the video files and streams. Besides, the disk caching, the context switching of operating system, slack time caused by inter-process communication, and the time Producer waiting for an empty buffer are also affecting factors.

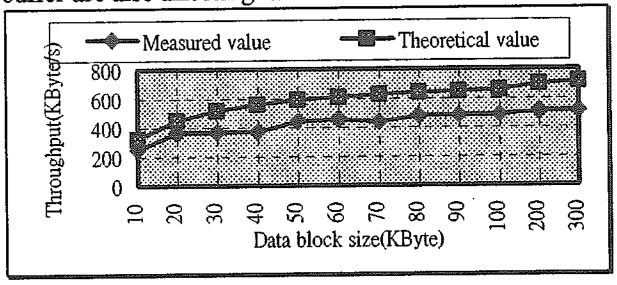

Figure12. Comparison of measured and theoretical server throughput.

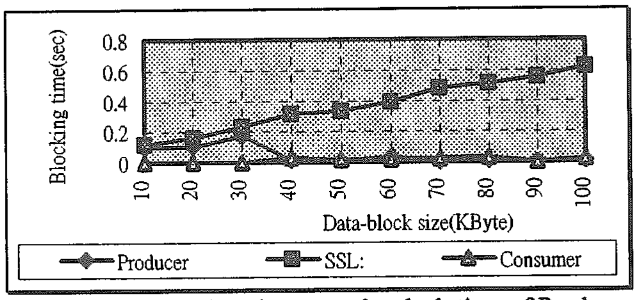

Figure 13. Comparison between the slack time of Producer and Consumer with SSL

Figure 13 shows the comparison of round length with slack time[1]. We can see that the Producer slack time is very high when data-block size is small, but once the data-block is beyond 40KByte, the slack time, comparing with round-length, will become small enough and can be ignored. We can explain this situation by examining Figure 14, in which the disk transfer rate is much higher than the throughput when the data-block size is smaller than 40KByte, that causes *Producer* to wait longer for *Consumer*. However, it does not ~~degrade our system~~ performance too much, because the system throughput should not be induced from the best case but rather the average cases. In other words, we can not count on the peak disk transfer rate to do the admission control.

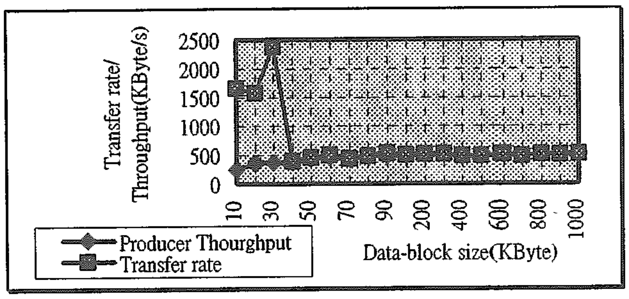

Figure 14. Comparison of producer throughput and transfer rate with different data-block sizes.

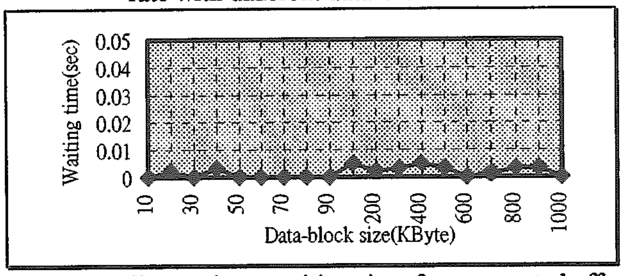

Figure 15. The producer waiting time for an empty buffer.

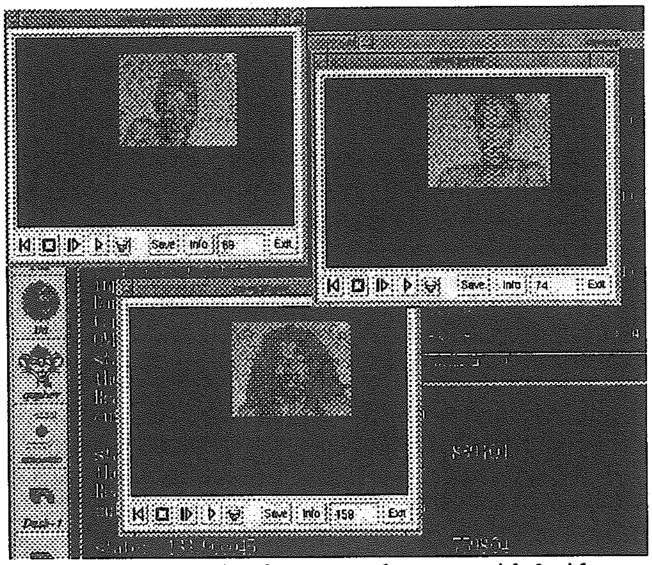

Figure 16. A scenario of prototyped system with 3 video streams.

Figure 15 shows the statistics of *Producer* waiting for an empty buffer. We can see that the waiting time is always less than 5ms, and it affects the performance of our system slightly. This demonstrates that our buffer scheduling algorithm is feasible and robust.

From the remarks above, we can conclude that in order to maximize the video server throughput and the number of streams to be supported, the data-block size should be properly set to the saturation point so that the buffer allocated is not too large to waste memory resource or too small to under-utilize the server performance[11]. In our system, the saturation point is about 100KBytes, which means 2MBytes buffer should be allocated if the system will support 10 streams with traditional Scan disk scheduling algorithm. Our buffer scheduling algorithm can reduce the buffer requirement from 2MBytes to 1.1MBytes, and if the system can support more streams, the advantage of our buffer scheduling algorithm will be more obvious.

Figure 16 demonstrates a scenario with three streams connecting to our video server simultaneously. We use X-windows system to show all three scenes on the same host through X-protocol. The three streams are decoded on different machines using software MPEG-1 decoder.

5. Conclusion and Future Work

An efficient buffer scheduling scheme combined with Scan for video server design is discussed. We analyze this algorithm mathematically and implement it in our video server based on a real-time micro-kernel. From the experiment, we show that our algorithm effectively reduces the buffer requirement. We also investigate how the size of data-block affects the server throughput. Since the system buffer is limited, with the proposed buffer scheduling algorithm, we can reduce the buffer requirement and hence support more streams.

Since only one disk drive equipped in our video server, the *producer* throughput seems to be the bottleneck. It is possible to adopt a faster disk controller to resolve this problem. For example, an advanced fast SCSI can have 20 MByte/s raw data transfer rate, which means it is possible to support up to 50 MPEG-1 video streams using our buffer scheduling scheme. This issue is left for further study.

References:

[1] D. Anderson, Y.Osawa, and R.Govindan, "A File System for Continuous Media," ACM Trans. on Computer Systems, Vol.10, No.4, Nov.1992, pp. 311-337.

[2] H. M.Vin and P.V. Rangan, "Admission Control Algorithms for Multimedia On-Demand Servers," Network and Operating System Support for Digital Audio and Video, Proceedings of Third International Workshop, Nov. 1992, pp.56-68.

[3] P. V. Rangan and H. M. Vin, "Efficient Storage Techniques for Digital Continuous Multimedia," IEEE Trans. on Knowledge and Data Engineering, Vol.5, No.4, Aug.1993, pp. 564-573.

[4] K. K. Ramakrishnan et. al., "Operating System Support for a Video-On-Demand File Service," Network and Operating System Support for digital Audio and Video, Proceedings of 4th International Workshop, Nov. 1993, pp. 216-227.

[5] D. Hildebrand, QNX Software Systems Ltd., "A Scalable Microkernel POSIX OS for Realtime Systems", Embedded Computer Conference, April 1993.

[6] J. Nieh, J. G. Hanko, J. D. Northcutt, and G. A. Wall, "SVR4UNIX Scheduler Unacceptable for Multimedia Applications," Network and Operating System Support for Digital Audio and Video, Proceedings of 4th International Workshop, Nov. 1993, pp. 41-53.

[7] A. L. Narasimha Rddy and J. C. Wyllie, "I/O Issues in a Multimedia System," Computer, Vol. 27, No. 3, Mar.1994, pp. 69-74.

[8] J. Gemmell and S. Christodoulakis, "Principles of Delay Sensitive Multimedia Data Storage and Retrieval," ACM Trans. Information Systems, Vol.10, No.1, Jan.1992, pp. 51-90.

[9] Draft International Standard, "Coded Representation of Picture, Audio and Multimedia/Hypermedia Information", ISO/IEC 11172, Dec. 1994.

[10]D. J. Gemmell, H. M.Vin, D. D. Kandlur, P.V. Rangan, and L. A. Rowe, "Multimedia Storage Servers: A Tutorial," IEEE Computer Magazine, May 1995, pp. 40-49.

[11]K. M. Nichols,"Performance Studies of Digital Video in a Client/Server Environment", Network and Operating System Support for Digital Audio and Video, Proceedings of Third International Workshop, Nov. 1992, pp. 81-91.

[12]R. Keller, W. Effelsberg and B. Lamparter,"Performance Bottlenecks in Digital Movie Systems", Network and Operating System Support for Digital Audio and Video, Proceedings of 4th International Workshop, Nov. 1993, pp. 161-171.

[13]P. Lougher and D. Shepherd, "The Design and Implementation of a Continuous Media Storage Server," Network and Operating System Support for Digital Audio and Video, Proceedings of Third International Workshop, Nov. 1992, pp. 69-80.

[14]QNX Software systems Ltd. "QNX 4 Operating System-System Architecture", July 1993.

[15]QNX Software systems Ltd. "QNX 4 TCP/IP-Programmer's Guide", August 1993.

[16]QNX Software systems Ltd. "QNX 4 TCP/IP-User's Guide", August 1993.

[17]C. Ruemmler and J. Wilkes, "An In troduction to Disk Drive Modeling," IEEE Computer Magazine, March 1994, pp. 17-28.

[18]Draft International Standard, "Coding of Moving Pictures and Associated Audio: Systems," ISO/IEC 13818-1, Nov. 1994.