TR-94-007

Continuous Media Recording for Video Conferencing

Meng-Huang Lee, Chia-Hsiang Chang
Meng-Chang Chen, Jan-Ming Ho
Ming-Ta Ko, Yen-Jen Oyang, Kuo-Hui Tsai

Continuous Media Recording for Video Conferencing

Meng-Huang Lee[1,3] Chia-Hsiang Chang[1,2]
Meng-Chang Chen[1,2] Jan-Ming Ho[1,2]
Ming-Ta Ko[1,2] Yen-Jen Oyang[3] Kuo-Hui Tsai[1,2]

Abstract

In this paper, we present a new real-time file system architecture which is optimized for supporting continuous media recording in video conference applications. We show that this new design optimizes simultaneously several performance criteria, i.e., minimum buffer size, maximum I/O bandwidth utilization, maximum I/O throughput, minimum I/O latency and minimum response time. Design of this file system is based on mathematical analysis of a FCFS I/O server for streams of continuous media. It uses a reservation-based admission control algorithm to manage system resources, i.e., I/O bandwidth, buffers, and disk space. Theoretical performance guarantees are also given. Preliminary experiments on PC 486's running Unix SVR4.0 version 4.0 with a slightly modified file system illustrate promising results.

1 Background

A distributed multimedia application can be viewed as exchanging multimedia data streams among n parties where each party generates zero or more data streams (such as video/audio data and shared work space). Multimedia data are often periodic samples of analog data and thus their processing and delivery usually have to be completed within a certain amount of time (real-time requirement). Multimedia data is also known as *continuous media.* Typical distributed multimedia application examples are video libraries (video-on-demand) and video conference . On a video library application, a server is used to store video movies and clients are allowed to connect to the server simultaneously for retrieving video movies. For this application, the server is capable of retrieving data from files, receiving commands (e.g. to freeze a video frame), and

[1] Institute of Information Science, Academia Sinica, Taiwan, R.O.C.
[2] This work has been supported in part by the National Science Council under Grant NSC83-0408-E-001-001.
[3] Department of Information Engineering and Computer Science, National Taiwan University, Taiwan, R.O.C.

sending data to networks in real-time. A video conference allows multiple user to communicate visually with each other through the network. A completed video conference system facilitates multiple groups to work simultaneously, and it must support movie playback and on-line conference recording. These new types of services can be made possible through a real-time network with real-time file servers. A real-time system guarantees timing correctness as well as logic correctness. Traditional network and file system servers are not designed or optimized for this objective, and thus it opens an opportunity for new research directions and possible new products with a potentially large market. Other applications are entertainment movies, educational documentaries, advertisements, etc.

A file system designed to support recording and playback of continuous media [12, 3, 4] [19, 9] [13, 10] [14, 15] [2, 22, 16] [8, 6, 1, 21] [18, 7, 20, 11] [17, 5]. e.g., video and audio data, is characterized by continuous recording and retrieval of multiple data streams at periodical intervals, and usually involves sequential access of large files at a high I/O bandwidth. For example, without compression the bandwidth of a video stream is around 15M bytes per second and that of CD quality audio is around 150KBps to 176KBps. In addition, the real-time nature of continuous media requires the file system to guarantee this bandwidth through the entire media recording or playback process. This guarantee is much more difficult to maintain when multiple streams are being recorded or played, where each stream is competing for I/O bandwidth, system buffers, and disk space. These system resources must be carefully managed to avoid over-commitment. Thus, when a media stream is to be opened, the amount of resources it needs must be checked against the currently available resources in the system. A continuous media stream is admitted only if enough resource is available. Otherwise, the request is rejected. This call admission process is analogous to that of a public telephone system. If the system admits a certain phone call request, it guarantees service quality during the call. Otherwise, the phone call request is simply rejected.

A continuous media file system basically consists of four design issues: admission control, buffer management, I/O scheduling, and disk layout, though buffer management issues are not explicitly addressed. Disk layout strategy refers to the definition of physical block size of each stream, placement of each block, and usage of multiple disks. I/O scheduler determines when and how to schedule the set of active requests to maximize I/O throughput, or to simplify performance analysis. The buffer management strategy determines the amount of buffers to allocate, and the buffer architecture, i.e., whether the buffers are used independently by each continuous media stream, or are shared by a group of streams, etc. The admission controller negotiates with the above three components to decide whether to accept a new request to playback or to record a certain continuous media stream or not. It guarantees that the time to process the I/O requests accumulated in the previous service cycle does not exceed the length of a service cycle. It also guarantees that the number of I/O requests flowing into the buffers equals that flowing out of the buffer, so that the total number of I/O requests accumulated in buffers is bounded. Notice that these performance guarantees must remain true in the worst cases. Average-case guarantees are not sufficient.

In most of these literatures, the I/O requests are processed in periodical batches at fixed periods T_c. T_c is referred to as the *service cycle*. During each service cycle except the first, the file system concurrently queues the incoming I/O requests to buffers, and processes I/O requests queued in the previous service cycle. The system guarantees that the total amount of data l_j requested by the incoming streams in the previous service cycle j are processed by the I/O controller within the current service cycle $j+1$.

Obviously, the worst case I/O latency, i.e., the time between a data block is requested until its corresponding physical disk block is accessed, is $2T_c$. System response time, i.e., the time interval from the user requests for a continuous media service until its first data block is processed by the system, is between 0 and T_c for write requests, and is between $2T_c$ and $3T_c$ for read requests, depending on detailed implementations and the arrival time of user request. Buffer requirement is between l_{max} and $2l_{max}$ depending on variations in scheduling policies and implementations, where $l_{max} = \max_{j=1}^{\infty} l_j$. Since the I/O requests are processed in batches, a higher I/O throughput is usually achievable. The basic ideas are to minimize disk seek time by either arranging data of a continuous stream in contiguous blocks or making proper use of the batched

SCAN algorithm and the use of RAID (redundant array of inexpensive disks) to multiply disk throughput.

But the I/O throughput could not be fully allocated to the continuous streams since the total amount of data for I/O usually varies from cycle to cycle. l_j is a constant for all service cycles j only when the service cycle T_c is chosen as a multiple of the period of each stream, i.e., if T_c is a multiple of the least common multiplier $LCM\{T_i\}_{i=1}^{m}$ of the periods T_i of the ith continuous streams, where m denotes the total number of streams. Output bandwidth must be allocated to the input streams according to the maximum data requirement per service cycle to account for the fluctuation in the aggregate data rate, i.e., l_j/T_c. In service cycles where the total amount of data is less than the maximum, the output bandwidth is left under-utilized. In other words, output bandwidth cannot be fully utilized unless T_c is chosen as $LCM\{T_i\}_{i=1}^{m}$. Note that an increase in the service cycle also implies an increase in buffer size.

2 Our Approach

In this paper, we address the problem of designing a real-time file system for recording of video conference applications on a desktop workstation running Unix operating systems. These applications require a write-once real-time file system for continuous media, and the recorded data is usually playback together. For applications requiring browsing through the individual continuous streams, media editing tools must be provided which is beyond the scope of this paper.

We also notice that in a video conference application, though it is usually necessary to record a complete set of audio streams, a complete record of the entire fully animated video streams is not necessary. A series of still-picture samples of the entire video streams at a lower rate and lower resolution, say a 1/4 screen-size picture per second instead of 30 full-screen pictures per second, is usually sufficient. In other words, existing PC technology is able to support these continuous media applications up to a certain degree of quality. In table ?? and ??, we list bandwidth requirements of audio and video streams under different compression schemes, resolutions and frame rates.

	Sample Size	
Compressed Ratio	Speech	CD quality audio
1	8K	176K
2	4K	88K
4	2K	44K
8	1K	22K

Table 1: Bandwidth of varieties of audio streams.

Plain/JPEB/MPEG	Frame Size			
Frame rate	full	1/4	1/16	1/64
30	15M/1.5M/200K	4M/400K/50K	1M/100K/13K	235K/23.5K/3.2K
25	12M/1.2M/166K	3M/300K/42K	750K/75K/11K	188K/18.8K/3K
20	9.6M/960K/144K	2.4M/240K/36K	600K/60K/9K	150K/15K/2.2K
15	7.2M/720K/100K	1.8M/180K/25K	450K/45K/6.2K	112K/11.2K/1.6K
10	4.8M/480K/67K	1.2M/120K/17K	300K/30K/4.2K	75K/7.5K/1K
5	2.4M/240K/34K	600K/60K/8.5K	150K/15K/2.1K	38K/3.8K/500
1	480K/48K/6.6K	120K/12K/1.7K	30K/3K/425	7.5K/750/106
1/2	240K/24K/3.3K	60K/6K/825	15K/1.5K/206	3.8K/380/52

Table 2: Bandwidth requirements of a video stream.

We also take a rough measurement of the throughput of the Ethernet connecting PC's running Unix operating systems using TTCP package. The test results are list in table 1. TTCP was written by Mike Muuss at the Ballistics Research Laboratory and has been modified at Silicon Graphics. It is a test program for evaluating the performance of a TCP connection. We also measured the throughput of a SCSI hard disk, Seagate ST-2383N, on our PC's. Its manufacturer's specifications is given in table 2, and performance data as measured by Coretest program developed by Core Inc. is also given in table 2.

We also measure I/O throughput of a Unix file system on a PC/486- with 16M RAM using the *performance test of sequential file I/O* program, also known as *IOZONE*, developed by Bill Norcott and a procedure which

segment	from (PC model)	to (PC model)	throughput (Kbytes/sec)	comments
same	486/33 (TH)	486/66	426	(3c503 card)
same	486/66	486/33	388	
same	486/33 (TH)	486/33	366	
no	486/33 (TH)	486/33	260	(gateway is 486/66)

Table 3: Throughput of TCP connections on PC's.

Hard Disk Type	SEAGATE ST-2383N	
Formatted Capacity	300 MByte	
Spindle Speed	3600 RPM	
	Manufacturer's Specification	Coretest Result
Average Latency	8.33 msec	8.33 msec
Track to Tract Seek time	3 msec	9.8 msec
Transfer Rate	2.25-2.75 MByte/sec	1.3 MByte/sec
Max Full Seek time	33 msec	
Average Access Time	14 msec	

Table 4: Manufacturer's specifications and benchmark of Seagate ST-2383N.

probes directly into Unix SVR4 kernel. Both shows a throughput of around 120 to 124 Kbytes per second. Note that the maximum block size of this kernel is only $2k$ bytes on the system currently running in our laboratory.

Furthermore, on an open multitasking computer system, CPU switches contexts among multiple user tasks and is not always available for I/O processing. Practical programming models and system constraints should also be taken into account.

For example, we also take practical Unix programming styles into consideration, i.e., the process-kernel computing model in which the kernel manages every I/O related activities and user processes sit upon the kernel to perform complicated processing and computing. As a common practice, Unix operating system also provides kernel buffers to hold file data submitted by user processes. Control returns to user processes upon successful allocation of kernel buffer and data transfer of user data into these buffers. The process is blocked in case of running out of kernel buffers. This type of operation is usually called *asynchronous I/O mode*.

We then model the real-time file scheduling problem as a *zero-response time buffered-I/O problem* in which kernel buffer requirement is minimized. This approach effectively decomposes the real-time I/O scheduling problem into two subproblems: performance guarantees for real-time user processes, and design of a file system for maximum throughput. In other words, we study the problem of providing sufficient amount of kernel buffer such that user processes are guaranteed to submit file data to kernel without being blocked due to a shortage in buffer resource. This model is particular useful if the user process also has some other computing subtasks, e.g., handling and processing of multiple I/O threads. As will be shown later, our theoretical analyses guarantee that if the disk I/O server consumes data in the kernel buffer fast enough, real-time response of the user process is guaranteed automatically.

To achieve maximum I/O throughput, we modify the source code of Unix kernel so as to allocate a region of $200M$ bytes on the hard disk used exclusively for storing data blocks of continuous streams. These free disk space is online allocated to the continuous streams through the conventional *inode* indexing mechanism on a *first-come first-serve (FCFS)* basis. It guarantees minimum disk seek time in transferring each consecutive block of data. We also defer the output of the inode structures until the end of recording in order not to perturb continuous I/O process.

We can then take full advantage of the original UNIX file system which is designed for optimal flexibility and efficiency (through the incorporation of, say, SCAN algorithm for I/O scheduling).

2.1 An admission control/resource allocation algorithm

In the previous section, we introduce our file system design based on a FCFS I/O server. This server is data-driven, i.e., data is sent to the disk whenever data is available. It is different from the previous continuous media file systems which are basically batch processing systems. The worst-case queuing behavior of this FCFS I/O server under periodical workload is analyzed in section 5, the appendix. It shows that the maximum queue length of this server if properly controlled is no greater than the summation of number of I/O requests per period of each active continuous streams. Note that this upper bound on the maximum queue length is tight and is guaranteed even when total utilization of I/O bandwidth is 100%. This theorem is the foundation of our admission control algorithm as will be presented in the following.

Let's denote m as the number of continuous streams currently allowed to the system. A task τ_i for recording a stream of continuous media is modeled as the release of a series of I/O (*write*) requests for a fixed number of blocks at constant time intervals. A task τ_i is thus described by four parameters, i.e., $\tau_i = (T_i, N_i, \delta_i, D_i)$, where T_i is the *period* of τ_i, N_i is the *number of blocks* to be written to disk at the beginning of each period T_i, δ_i is the *release time* of τ_i, and D_i is the duration of recording for τ_i. Total number of block frames in the system buffer is denoted as N_B, total number of block frames in the disk allocated to real-time data as N_D, and the throughput of the I/O server is denoted as $1/T_0$, . Note that, as mentioned in the previous section, $1/T_0$ is approximately $120k$ bytes, or alternatively 60 blocks, per second. The maximum queue length theorem is stated below while its proof appears in the appendix.

Theorem 1 *Let T_0 denote the I/O service time, and $\Delta = \{\tau_1, \tau_2, \cdots, \tau_m\}$ denote a collection of periodical input streams, where $\tau_i = (T_i, N_i, \delta_i)$, for $i = 1, 2, \cdots, m$. If $\sum_{i=1}^{m} \frac{N_i}{T_i} \leq \frac{1}{T_0}$, then the maximum queue length is $\sum_{i=1}^{m} N_i$.*

Note that if $\sum_{i=1}^{m} \frac{N_i}{T_i} > \frac{1}{T_0}$, then the queue grows indefinitely until either the end of the entire recording process or running of system buffer and a certain I/O requests are forced to wait. Also notice that the upper bound $\sum_{i=1}^{m} N_i$ is tight (consider the time instance when all the I/O requests arrive simultaneously). In other words, this is the minimum buffer requirement achievable by any I/O servers for the particular class of periodical task sets as described above. It is also a reasonable model of the operation of the system interfaces of a kernel-based multiprogramming system, e.g., Unix operating systems. Since a I/O request for a data block finds itself queued at a position no greater than $\sum_{i=1}^{m} N_i$ in the FCFS queue, the FCFS queuing policy guarantees that this particular I/O request is processed in at most $\sum_{i=1}^{m} N_i T_0$ time. Which again is a minimum due to the above discussions. Since the user processes use asynchronous write requests to send I/O requests and the amount of resources are always available, the *write()* system call is never blocked and system response time is thus minimized. We have the following corollary.

Corollary 1 *The FCFS I/O server as described above has the following properties:*

1. *buffer requirement is minimized;*

2. utilization of I/O bandwidth is maximized;

3. I/O latency is $\sum_{i=1}^{m} N_i T_0$ is minimized;

4. system response time is minimized.

Note that our file system architecture also provides a maximum I/O throughput at the current default block size of Unix SVR4.0 version 4.0 as was described earlier.

We are now ready to present our admission control algorithm. The algorithm involves only three inequality predicates.

$$\sum_{i=1}^{m} N_i \leq N_B \tag{1}$$

$$\sum_{i=1}^{m} \frac{N_i}{T_i} \leq \frac{1}{T_0} \tag{2}$$

$$\sum_{i=1}^{m} N_i \left\lceil \frac{D_i}{T_i} \right\rceil \leq N_D \tag{3}$$

Note that the first inequality is also denoted as the buffer constraint, the second the utilization constraint, and the third the disk space constraint. In order for a set of m tasks as described above to be admitted simultaneously, they have to satisfy all the three constraints. In a on-line environment in which the tasks arrive one by one, the admission control algorithm takes constant time to decide whether to accept a new request for continuous media recording or not.

In our system, three new programming interfaces are provided for continuous media recording, i.e., *r_open*, *r_write* and *r_close*. When a process requests to record a continuous media stream into the file system, *r_open* is called to perform the admission control algorithm and pre-allocate system resources for later use once admitted. A descriptor is returned by *r_open* upon successful return from *r_open*, otherwise −1 is returned. The second service call *r_write* also verifies that the user process writes data at the negotiated rate and amount of data in addition to the normal operations of a conventional *write* system call. The last service call *r_close* deallocates the kernel buffers previously allocated to the user process. Our previous analysis guarantees real-time behavior of continuous media recording operated under these programming interfaces.

3 Experimental Results

In commercially available Unix SVR4.0 version 4.0 operating system, the response time of *write* system call is highly non-deterministic. These non-determinism are due to I/O transfer of internal data structures of the file system, i.e., transfer of *superblocks* and *inodes*, the access of indirect blocks for indirect addressing, and the retrieval of the addresses of free blocks. As mentioned previously, our file system defers the transfer of *superblocks* and *inodes* until the end of entire recording process. On the other hand, since the continuous data blocks are written to a write-once region in the hard disk, two pointers in this disk region is sufficient for management of free blocks.

Each Unix file is associated with a *inode* structure for mapping the address of a logical block to its physical address on the disk. An index array in the *inode* structure is used by the file system to map a logical block number into a physical block number, where a logical block number is a local index in a particular file and a physical block number refers to the address of a disk storage area to hold a logical data block. The index array contains 10 direct pointers to data blocks. For a file system with 2Kbyte block size, it allows the first 20kbytes of file data to be accesses without extra overhead, assuming that the *inode* block is cached in kernel buffer previously. The 11th pointer points to an address block containing several, say 256, physical block number of the 11th to the 267th logical blocks. The 12th pointer is a double-indirect pointer and the 13th is a triple-indirect pointer. As a result, it takes two I/O transfers to access the 11th logical block, etc. Note that once the indirect block is cached, access of the, e.g., 12th logical block takes only one I/O transfer.

In its original design, Unix file system fetches indirect blocks using synchronous I/O operations, i.e., a process blocks until the end of the transfer of an indirect block. This design is necessary for a *read* operation, but we don't see any particular reason why it's necessary for a asynchronous *write* operation. This waiting

no. of streams	no. of streams miss deadline	measured first deadline violation	predicted first deadline violation	mean (msec)	deviation (msec)
1	0	-	-	3.0	0.8
2	0	-	-	3.1	1.2
3	0	-	-	2.9	4.2
4	4	43	45		
5	5	23	22		

Table 5: frame rate : 10 frames/sec, frame size : 4 Kbytes

time is usually hard to predict and is harmful to real-time performance of a real-time process. So, we eliminate this nuisance by modifying the synchronous operation of a indirect block to an asynchronous I/O operation. Effect of this modification on the response time of an asynchronous write operation is dramatic.

In this following, we present two sets of experiments to evaluate our recording system. In these experiments, the buffer size is configured as 87 for a certain technical reasons which we are not going to address in this paper. Each experiment had been run for approximately 20 minutes due to disk space constraint. Note also that the I/O throughput is 120K bytes per second. The first set of experiments is designed to verify the effect of satisfying the admission tests and the results of violating the bandwidth constraint. While the second set of experiment is used to verify the effect of satisfying the admission tests and the results of violating the buffer constraint.

To see the effect of over committing I/O bandwidth, we may consider the scenario of the fourth experiment in set 1 (see table 3). In this experiment, there are 4 recording streams each sending 4K-byte frames (2 data blocks) at the rate of 10 frame per second. Thus, at each 0.1-second time interval, buffer requirement is 8 block frames and the number of block frames returned by the I/O server is 6. Number of block frames in system buffer decreases by 2 for each period of 0.1 seconds. The system originally has 87 free block frames and will last for 43 periods, the processes are then blocked to wait until buffer becomes available again. This is consistent with the experimental data.

In the second experiment (see table 4), the buffer requirement (104 block frames) is greater than the buffer

stream id	measured deadline violations	predicted deadline violations
s1	0,6,12,18,24,*27,30,36,42, 48,54,60,66,72,78,84,*89,90, ...	0,6,12,18,24,30,36,42 48,54,60,72,78,84,90 ...
s2	-	-
s3	-	-

Table 6: Streams at different frame size and different frame rate. s1: frame size: 44 kbyte, frame rate: 1 frame/sec; s2: frame size: 80 kbyte, frame rate: 1 frame/2sec; s3: frame size: 84 kbyte, frame rate: 1 frame/3sec.

size (87 block frames). Obviously, the buffer is not sufficient when the I/O requests arrive simultaneously, i.e., at common multiples of the periods of the continuous streams. Experimental result are consistent.

4 Conclusion

In this paper, we present a Unix-based real-time file system design optimized for supporting video conference applications. We also analyze the maximum queue length of a FCFS server for continuous streams of data. This bound is shown to be tight. We also conclude that our design uses minimum amount of buffer to achieve optimum performance, i.e., maximum I/O bandwidth utilization, maximum I/O throughput, minimum I/O latency and minimum response time. Preliminary experiments on PC 486's running Unix SVR4.0 version 4.0 illustrate promising results.

We also observe that the mathematical analysis we present in this paper can be extended to model a continuous playback server, and other communication network design problems. Related research works are still ongoing.

References

[1] D. Anderson, Y. Osawa, and R. Govindan. Real-time disk storage and retrieval of digital audio and video. Technical report, U.C. Berkeley, 1991. UCB/ERL Tech. Rep. M91/646.

[2] David P. Anderson and Goerge Homsy. A continuous media I/O server and its synchronization mechanism. *IEEE Computer*, October 1991.

[3] A. Lester Buck and Robert A. Coyne. An experimental implementation of draft POSIX asynchronous I/O. *USENIX*, Winter 1991.

[4] N.G. Davies and J.R. Nicol. A technological perspective on multimedia computing. *Computer Communications*, 14(5), 1991.

[5] Jim Gemmell and Stavros Christodoulakis. Principles of delay-sensitive multimedia data storage and retrieval. *ACM Transactions on Information Systems*, January 1992.

[6] Shahram Ghandeharizadeh and Luis Ramos. Object placement in parallel hypermedia system. In *Proc. 1991 VLDB Conf.*, 1991.

[7] Shahram Ghandeharizadeh and Luis Ramos. An overview of techniques to support continuous retrieval of multimedia objects. *ACM, SIGOPS*, 1993.

[8] Shahram Ghandeharizadeh and Luis Ramos. Continuous retrieval of multimedia data using parallelism. *IEEE Transactions on Knowledge and Data Engineering*, 5(4), August 1993.

[9] P. T. Zellweger Harrick M. Vin, D. C. Swinehart, and P. Venkat Rangan. Multimedia conferencing in the Etherphone environment. *IEEE Computer, Special Issue on Multimedia Information System*, October 1991.

[10] A. Hopper. Pandora - an experimental system for multimedia application. *ACM Operating System Review*, 24(2), April 1990.

[11] D. D. Kandlur, M. S. Chen, and Z. Y. Shae. Design of a multimedia storage server. In *IBM Research Report*, June 1991.

[12] P. Lougher and D. Shepherd. The design of a storage server for continuous media. *The Computer Journal*, 36(1), 1993.

[13] P. Venkat Rangan, W. A. Burkhard, R. W. Bowdidge, Harrick M. Vin, J. W. Lindwall, K. Chan, I. A. Aaberg, L. M. Yamamoto, and I. G. Harris. A testbed for managing digital video and audio storage asynchronous I/O. *USENIX*, Summer 1991.

[14] P. Venkat Rangan and Harrick M. Vin. Designing file systems for digital video and audio. In *Proc. 13th Symp. Operation Syst. Principles*, 1991.

[15] P. Venkat Rangan and Harrick M. Vin. Efficient storage techniques for digital continuous multimedia. *IEEE Transactions on Knowledge and Data Engineering*, 5(4), August 1993.

[16] P. Venkat Rangan and Harrick M. Vin. Designing an on-demand multimedia service. *IEEE Communications Magazine*, July 1992.

[17] P. Venkat Rangan and Harrick M. Vin. Admission control algorithm for multimedia on-demand server. In *Proc. Third Symp. on Operating System Supports for Audio and Video*, June 1993.

[18] W. D. Sincoskie. System architecture for a large video on demand service. *Computer Networks and ISDN Systems*, (22), 1991.

[19] Harrick M. Vin and P. Venkat Rangan. Designing a multi-user HDTV storage server. *IEEE Journal on Slected Areas in Communication, Special Issue on High Definition Television and Digital Video Communication*, 11(1), January 1993.

[20] James Yee and Pravin Varaiya. An analytical model for real-time multimedia disk scheduling. In *Proc. Third Symp. on Operating System Supports for Audio and Video*, June 1993.

[21] C. Yu, W. Sun, and D. Bitton. Efficient placement of audio on optical disks for real-time applications. *Communication ACM*, July 1989.

[22] Philip S. Yu, Mon-Song Chen, , and Dilip D. Kandlur. Design and analysis of a grouped sweeping scheme for multimedia storage management. In *Proc. Third Symp. on Operating System Supports for Audio and Video*, pages 44–55, June 1993.

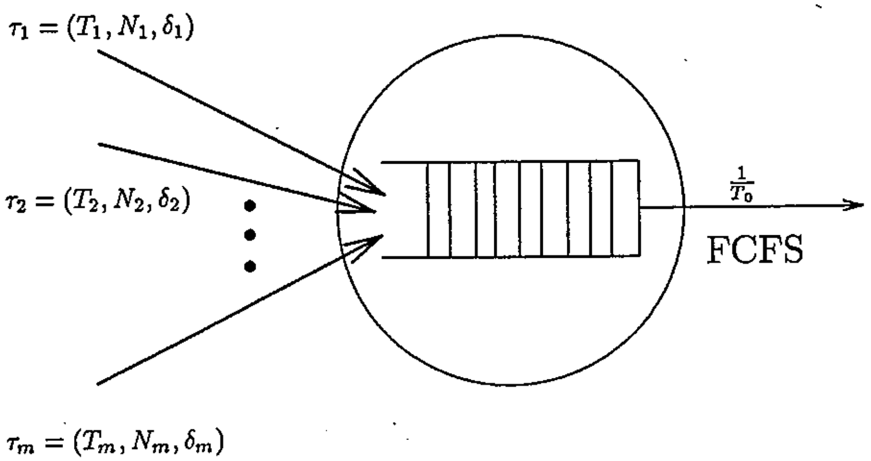

$\tau_1 = (T_1, N_1, \delta_1)$
$\tau_2 = (T_2, N_2, \delta_2)$
$\tau_m = (T_m, N_m, \delta_m)$
$\frac{1}{T_0}$
FCFS

Figure 1: Periodical Message Processing System Architecture

5 Appendix

In this section, we study the buffer consumption problem for a periodical message processing system. This system (see Fig. 1) has a set of m periodical input streams $\Delta = \{\tau_1, \tau_2, \cdots, \tau_m\}$. Every T_i time, each input stream $\tau_i = (T_i, N_i, \delta_i)$ periodically feeds N_i message into the system buffer starting from its release time δ_i. This system has unlimited size of buffer, and it takes T_0 time interval to process and send a message out of system via its output port on first come first serve (FCFS) basis. We shall show that the maximum number of buffer required is $\sum_{i=1}^{m} N_i$ if system *utilization factor* $\sum_{i=1}^{m} \frac{N_i T_0}{T_i}$ is less than or equal to 1.

Before preceding, we define some convenient vocabularies. Let *accumulated input function* $g_1^{\Delta}(t)$ (respectively, *accumulated processed function* $g_2^{\Delta}(t)$) be the number of messages fed to the system (respectively, serviced by the system) from time 0 to time t. The essential idea of our proof is to show that $g_2^{\Delta}(t) + \sum_{i=1}^{m} N_i \geq g_1^{\Delta}(t)$, for $t \geq 0$.

Consider the case in which utilization factor is equal to 1 (i.e. $\sum_{i=1}^{m} \frac{N_i}{T_i} = \frac{1}{T_0}$), and all the release time $\delta_i' s$ are 0 (i.e. $\delta_i = 0$, for $i = 1, 2, \cdots, m$). In this case, system is fully utilized (i.e. has no idle time). Lemma 1 shows that this system processes and outputs a message every T_0 time.

Lemma 1 *Let T_0 denote message processing time, and $\Delta = \{\tau_1, \tau_2, \cdots, \tau_m\}$ denote a collection of periodical input streams, where $\tau_i = (T_i, N_i, \delta_i)$, for $i = 1, 2, \cdots, m$. If $\sum_{i=1}^{m} \frac{N_i}{T_i} = \frac{1}{T_0}$, and if release time $\delta_i = 0$, for $i = 1, 2, \cdots, m$, then $g_2^{\Delta}(t) = \left\lfloor \frac{t}{T_0} \right\rfloor$, for $t \geq 0$.*

Proof: If the system has no idle time, then $g_2^{\Delta}(t) = \left\lfloor \frac{t}{T_0} \right\rfloor$, for $t \geq 0$. Assuming that there exists some idle periods, and let t_0 denote the starting time of the earliest idle period. Then, buffer must be empty at time t_0 (i.e. $g_1^{\Delta}(t_0) = g_2^{\Delta}(t_0)$). Since $g_1^{\Delta}(t_0) = \sum_{i=1}^{m} N_i \left\lceil \frac{t_0}{T_i} \right\rceil$ and $g_2^{\Delta}(t_0) = \left\lfloor \frac{t_0}{T_0} \right\rfloor$, then

$$\sum_{i=1}^{m} N_i \left\lceil \frac{t_0}{T_i} \right\rceil = \left\lfloor \frac{t_0}{T_0} \right\rfloor$$

holds. By mathematical laws and problem assumptions, we have

$$\sum_{i=1}^{m} N_i \left\lceil \frac{t_0}{T_i} \right\rceil \geq \sum_{i=1}^{m} N_i \frac{t_0}{T_i} = \frac{t_0}{T_0} \geq \left\lfloor \frac{t_0}{T_0} \right\rfloor$$

Thus, we have

$$\sum_{i=1}^{m} N_i \left\lceil \frac{t_0}{T_i} \right\rceil = \sum_{i=1}^{m} N_i \frac{t_0}{T_i} = \frac{t_0}{T_0} = \left\lfloor \frac{t_0}{T_0} \right\rfloor$$

This equation implies that t_0 is a common integral multiple of $T_0, T_1, \cdots$ and T_m. However, all input streams feed $\sum_{i=1}^{m} N_i$ messages simultaneously to the system at time t_0. Thus, there is no idle period starting from t_0. A contradiction. ◇

We now continue our argument for the case in which utilization factor is 1 and all the release time $\delta_i^\prime s$ are 0. Consider the following formulas.

$$\begin{aligned}
g_1^{\Delta}(t) &= \sum_{i=1}^{m} N_i \left\lceil \frac{t}{T_i} \right\rceil \\
&\leq \sum_{i=1}^{m} N_i (\left\lfloor \frac{t}{T_i} \right\rfloor + 1) \\
&= \sum_{i=1}^{m} N_i \left\lfloor \frac{t}{T_i} \right\rfloor + \sum_{i=1}^{m} N_i
\end{aligned}$$

Observe the fact that the inequality $\lfloor xy \rfloor \geq x \lfloor y \rfloor$ holds for non-negative real number x and y. Since $\sum_{i=1}^{m} \frac{N_i}{T_i} = \frac{1}{T_0}$, we have the following formulas.

$$\begin{aligned}\sum_{i=1}^{m} N_i \left\lfloor \frac{t}{T_i} \right\rfloor + \sum_{i=1}^{m} N_i &\leq \left\lfloor \frac{t}{T_0} \right\rfloor + \sum_{i=1}^{m} N_i \\ &= g_2^{\Delta}(t) + \sum_{i=1}^{m} N_i \qquad \textit{by Lemma 1} \qquad (4)\end{aligned}$$

From Equation (4), we have Theorem 1.

Theorem 1 *Let T_0 denote message processing time, and $\Delta = \{\tau_1, \tau_2, \cdots, \tau_m\}$ denote a collection of periodical input streams, where $\tau_i = (T_i, N_i, \delta_i)$, for $i = 1, 2, \cdots, m$. If $\sum_{i=1}^{m} \frac{N_i}{T_i} = \frac{1}{T_0}$, and if release time $\delta_i = 0$, for $i = 1, 2, \cdots, m$, then $g_2^{\Delta}(t) + \sum_{i=1}^{m} N_i \geq g_1^{\Delta}(t)$, for $t \geq 0$.*

Fig. 2 depicts $g_1^{\Delta}(t)$ and $g_2^{\Delta}(t)$ for the case in which $T_0 = 1$ and input stream set $\Delta = \{\tau_1 = (N_1 = 2, T_1 = 4, \delta_1 = 0), \tau_2 = (N_2 = 4, T_2 = 12, \delta_2 = 0), \tau_3 = (N_3 = 1, T_3 = 6, \delta_3 = 0)\}$. It shows that the maximal number of buffer required is 7 which is exactly $\sum_{i=1}^{3} N_i$.

We now consider the general case in which utilization factor is less than or equal to 1, and release time $\delta_i^{\prime}s$ are greater than or equal to 0. Unlike the previous case, system now may become idle for a certain period of time (see Fig. 3). System is idle if and only if there is no message stored in the buffer. Consider a busy interval I_k starting from s_{I_k} ending at e_{I_k}. At the time just prior to s_{I_k} or immediately after e_{I_k}, the buffer is empty. Let $\Delta_{I_k} \subseteq \Delta$ be the set of input streams which feed messages to the system during time interval $[s_{I_k}, e_{I_k}]$. We define $g_1^{\Delta_{I_k}}(t)$ (respectively, $g_2^{\Delta_{I_k}}(t)$) to be the number of messages fed to system (respectively, serviced by the system) from time s_{I_k} to time t, where $s_{I_k} \geq t \geq e_{I_k}$. Using the same argument used in proving Theorem 1, we have the following formulas, for all t, $s_{I_k} \geq t \geq e_{I_k}$.

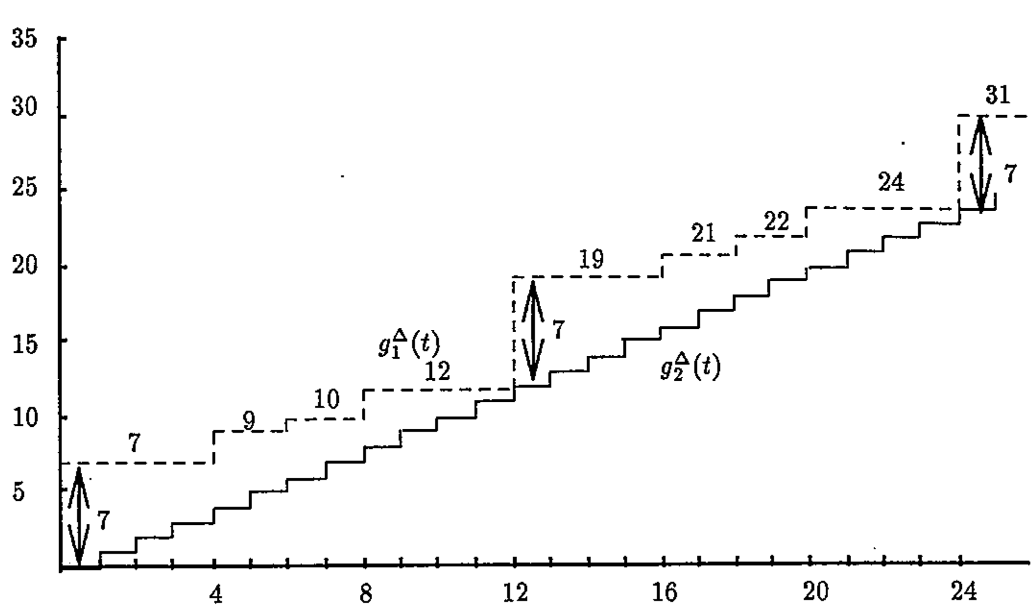


Figure 2: Buffer Requirement

$$\begin{aligned}
g_1^{\Delta_{I_k}}(t) &\le \sum_{\tau_i \in \Delta_{I_k}} N_i \left\lceil \frac{t - s_{I_k}}{T_i} \right\rceil && \text{as if release time } \delta_i' s \text{ were } s_{I_k} \\
&\le \sum_{\tau_i \in \Delta_{I_k}} N_i (\left\lfloor \frac{t - s_{I_k}}{T_i} \right\rfloor + 1) \\
&= \sum_{\tau_i \in \Delta_{I_k}} N_i \left\lfloor \frac{t - s_{I_k}}{T_i} \right\rfloor + \sum_{\tau_i \in \Delta_{I_k}} N_i \\
&\le \left\lfloor \frac{t - s_{I_k}}{T_0} \right\rfloor + \sum_{\tau_i \in \Delta_{I_k}} N_i && \text{since } \textstyle\sum_{\tau_i \in \Delta_{I_k}} \frac{N_i}{T_i} \le \frac{1}{T_0} \\
&= g_2^{\Delta_{I_k}}(t) + \sum_{\tau_i \in \Delta_{I_k}} N_i && \text{since } I_k \text{ is a busy period}
\end{aligned}$$

We therefore conclude Theorem 2. Some busy periods of the case in which $T_0 = 1$ and input stream set $\Delta = \{\tau_1 = (N_1 = 2, T_1 = 4, \delta_1 = 0), \tau_2 = (N_2 = 4, T_2 = 12, \delta_2 = 11), \tau_3 = (N_3 = 1, T_3 = 6, \delta_3 = 5)\}$ is depicted in Fig. 4.

Theorem 2 *Let T_0 denote message processing time, and $\Delta = \{\tau_1, \tau_2, \cdots, \tau_m\}$ denote a collection of periodical input streams, where $\tau_i = (T_i, N_i, \delta_i)$, for $i = 1, 2, \cdots, m$. If $\sum_{i=1}^m \frac{N_i}{T_i} \le \frac{1}{T_0}$, then equation $g_2^\Delta(t) + \sum_{i=1}^m N_i \ge g_1^\Delta(t)$ holds, for $t \ge 0$.*

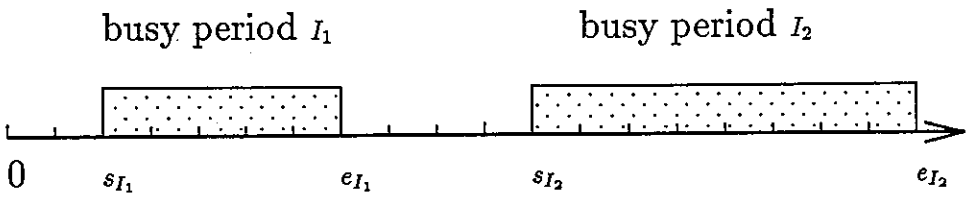
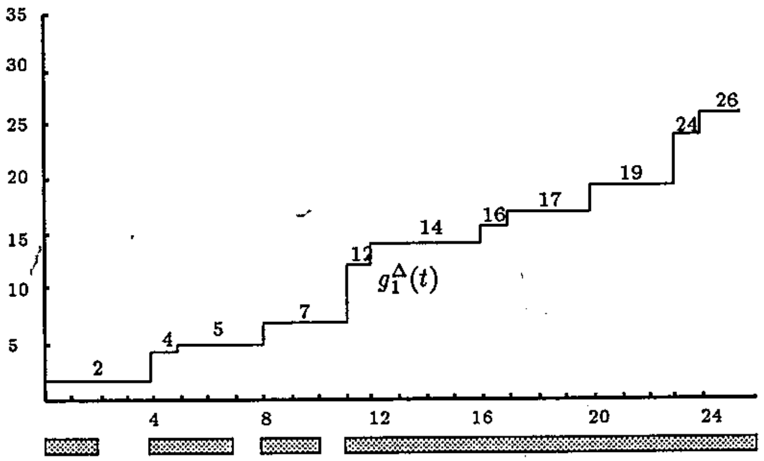


Figure 3: Busy and Idle Periods



shaded area: busy period

Figure 4: Busy Periods