Design and Analysis of a Real-Time Storage Server for Multimedia Applications using Disk Arrays

Jinsung Cho and Heonshik Shin

Department of Computer Engineering
Seoul National University
Seoul 151-742, Korea

Abstract

This paper designs a real-time storage server (RTSS) for remote multimedia playback applications. The design goal is to maximize the number of clients which can be serviced simultaneously. RTSS must, however, maintain the continuity of each medium. To satisfy both the design goal and the constraint, we propose a new scheduling scheme called round scheduling to effectively retrieve multimedia disk blocks. The round scheduling is shown to minimize disk idle time. We also present SCAN-like disk arm scheduling that optimizes seek time to access multimedia blocks. Disk architecture for our server adopts a disk array, where data blocks are striped across all the disks.

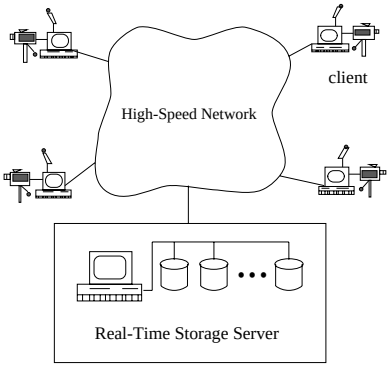

Figure 1: Configuration of remote multimedia playback system

1 Introduction

Recent advances in computer technology and demand of video, audio, graphics, and text integration services have provided driving force behind the emergence of various multimedia applications including teleconferencing, computer supported coorperative work, multimedia electronic mail, and computer aided learning[1, 2]. Most of today's multimedia playback applications are standalone and not distributed. This is due to the limitations of networks and storage servers. Thus, remote multimedia playback applications have given a technical challenge with regards to real-time transmission, storage, and retrieval of multimedia informaiton.

In this paper, we present the design and analysis of a real-time storage server(RTSS) for remote multimedia playback system. RTSS services the real-time playback of multimedia files over high-speed network to remote clients as shown in Fig. 1, as well as nonreal-time insert, delete, and update operations. During this real-time playback, the server must maintain the continuity of each medium and the synchronization between media[3], and service as many clients as possible. To accomplish this, a real-time storage server must satisfy the following conditions: (1) It must have a high capacity storage system with a high I/O bandwidth. (2) We must implement a real-time disk scheduling that ensures the highest performance of storage system.

The first condition requires the use of disk array technology for the design of RTSS with respect to both high capacity and bandwidth. Thus our presentation will center around disk arrays and their effective management. For the second condition, we propose a new scheduling scheme, *round scheduling*, which schedules disk blocks to retrieve in every *round*. For this purpose we also presents an adaptation of the classical SCAN for the most suitable disk arm scheduling for RTSS.

The rest of this paper is organized as follows: Section 2 states the assumptions for the application model. Section 3 describes the basic framework and designs the key elements of the RTSS. Section 4 dis-

cusses the disk architecture and Section 5 concludes this paper.

2 Application Model

Storage layout of RTSS employs the following data model: A *media unit* is the basic unit of each medium like, for example, frame for video, sample for audio, character for text. A *multimedia block* is defined as the basic unit of disk storage and retrieval, and consists of media units. A set of multimedia blocks forms a *multimedia file*, which is the basic unit of a client request. The *directory* is a data structure that stores the attributes of multimedia files. The application model of RTSS is based as the following assumptions.

Assumption 1 *The client has the processing capabilities necessary for RTSS.*

The client must include functions like decompression for compressed multimedia files, resynchronization between media, buffering for network jitter and so on[6, 7]. Also, this assumption implies that they have all the necessary resources for the scheduling and prefetch algorithm of the server to run.

Assumption 2 *The outgoing data packets from RTSS must arrive at a client in the same order as transmitted with no errors and variations in transmission time .*

RTSS provides services to clients through high speed network. The arbitrary delay of transmission, however, will make real-time playback impossible. So, the server must provide a mechanism that transmits a packet with a fixed transmission delay. This can be done by reserving the transmission bandwidth and other resources during the authentication process of the client. Real-time communications for multimedia applications have been extensively studied[13, 14].

By the preceding assumptions, this paper guarantees the continuity of multimedia files with regards to the server's point of view only. That is, for the design of RTSS we take into account the continuity of multimedia blocks being transmitted from server to client.

Assumption 3 *RTSS allows only system managers with special privileges to add, delete, update and perform other manipulations to the stored data.*

Data shared by several clients cannot be altered by themselves. Privately owned data is of no great importance to our discussion. Clients can only access the shared data. The management of shared data is done by the system manager who has special privileges. This assumption implies that system manager's operations are not time-critical; so they can be executed as the background tasks to the real-time operations of the clients, or off-line.

Assumption 4 *Multiple media are stored in a block.*

There are two approaches for storing multiple media on a disk[2]: *homogeneous block* which contains only one medium, and *heterogeneous block* which contains multiple media. We choose heterogeneous blocks because creating heterogeneous blocks is a nonreal-time job, while the retrieval of them is time-critical. That is, in homogeneous block approach each media block must be accessed, where the number of disk seeks is the number of media, while in heterogeneous blocks only one seek is required. As seek time is larger than CPU processing time, it has an adverse effect on the performance of the system[8]. Hence, we are justified in choosing the heterogeneous block for the storage server where real-time retrieval is of major concern. This decision is also accompanied by the extra benefit of synchronization between media.

Assumption 5 *The playback time of the multimedia block within a multimedia file is constant.*

Assumption 5 allows a hard real-time scheduling, and thus, deterministic service can be accomplished[1]. Variable-rate compression techniques such as MPEG[15] make the deterministic service impossible and cause the anomaly called *jitter*.

3 Design of Real-Time Storage Server

As mentioned in Section 1, a real-time storage server must have high-performance storage system. To satisfy this constraint, we adopt disk array architecture, modeled on the level 5 RAID which distributes the data and check information across all

[1] In [1], this assumption is called 'normalized stream' and in [5], 'heterogeneous stream'.

the disks[8]. RAID 5 shows a good performance for read operation[9] thus is considered the most suitable model for the read-intensive storage server. Its justification is discussed in detail in Section 4. The on-line recovery of damaged data in RAID, however, is excluded. We assume that there are back-up copies in other storage facilities.

The logical structure of RTSS and relationship between its components are as follows: the *interface manager* receives and analyzes client requests[2] and then sends the service request to the multimedia file manager. The *multimedia file manager* then provides a high-level service, *multimedia file service*, to the client by performing the feasibility test for a new request and, if it is successful, demanding the disk manager to prefetch multimedia blocks. Another role of the multimedia file manager is to transmit multimedia blocks periodically to the clients through communication channels. The *disk manager* performs the round scheduling which reduces disk idle time, and a disk arm scheduling which reduces seek time, thereby providing a low-level service, *disk service*. Separate disk managers exist for every disk under disk array architecture. In the next subsections, the two key elements of RTSS, the multimedia file service and the disk service, are described in detail.

3.1 Multimedia File Service

In this section, we derive the feasiblity test condition which guarantees that RTSS can service a new request without causing discontinuity in the retrieval of any of the existing client requests. An efficienct disk manager, as will be shown in Section 3.2, improves this condition. We also describe prefetching and data transmission under real-time constraints.

3.1.1 Feasibility Test

We first observe the process involved in serving a single client. A disk request is issued for each playback time, T_{play}, of a multimedia block. Multimedia blocks of a multimedia file are striped across all the disks[3], as shown in Fig. 2. Disk requests generated every T_{play} time are distributed to all the disks in a disk array and they arrive at each individual disk every $m \times T_{play}$ time, where m is the number of disks under disk array architecture. Hence, retrieval of a multimedia block must be completed within $m \times T_{play}$.

[2] They are general client requests such as play, stop, pause, resume, rewind, fastforward, and system manager's such as open, close, delete, record, insertblock, deleteblock.

[3] This is called 'block-striping'.

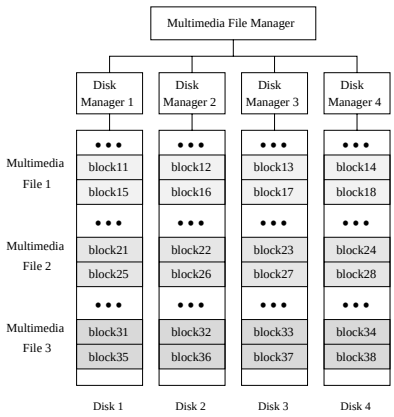

Figure 2: Disk service when $m = 4$

The following condition must be satisfied for every disk.

$$T_{seek\_max} + \frac{s}{R} \leq m \times T_{play} \qquad (1)$$

where T_{seek_max} is the maximum seek time, R is the data transfer rate of disk, and s is the size of multimedia block.

We now consider n client requests, $r_1, r_2, \cdots, r_n$. The multimedia file manager services client requests in the round-robin fashion. In Fig. 2, for example, block11, block21, block31, block12, block22, block32 are serviced in order. In each disk, n retrievals for r_1, $r_2, \cdots, r_n$ constitutes a *round*. That is, in a disk, B_j^1, $B_j^2, \cdots, B_j^n$ are retrieved in the jth round, where B_j^i is the jth multimedia block of r_i in the disk[4]. Each multimedia block, B_j^i, must be retrieved within its deadline, $m \times T_{play}^i$. The deadlines of B_j^i, $1 \leq i \leq n$, will be met if the period of a round is given by the shortest playback time of the n multimedia blocks, $m \times T_{play}^{min}$, where $T_{play}^{min} = \min_{1 \leq i \leq n}(T_{play}^i)$. In order to service n client requests, Eq. (1) is extended as the following feasibility test condition[5][1].

$$n \times T_{seek\_max} + n \times \frac{s}{R} \leq m \times T_{play}^{min} \qquad (2)$$

[4] In Fig. 2, the first round in disk1 includes the retrieval of block11, block21, block31 and the second round includes block15, block25, block35.

[5] The time for the retrieval of a disk block consists of seek time and rotational latency and data transfer time. Because rotational latency can be added to T_{seek_max}, it is excluded in our analysis.
Similary, the time for accessing a directory can be assumed as constant and is also excluded.

3.1.2 Prefetch and Transmission of Multimedia blocks

RTSS services client requests by transmitting multimedia blocks at a constant rate, and for this, multimedia blocks must be prefetched and buffered before transmission. The multimedia file manager periodically sends a prefetch command to each disk manager in consideration of synchronization between disk managers. Each disk manager completes prefetching before deadline or within a period and the multimedia file manager transmits multimedia blocks prefetched by the disk managers. Outgoing multimedia blocks will be delivered to a client in time, and the continuity constraint is satisfied by Assumptions 1 and 2.

3.2 Disk Service

In real-time database systems, a real-time disk scheduling is required to handle transactions with deadlines[10, 11, 12]. The charateristics of disk requests in the proposed RTSS, however, differ in many aspects from those of the traditional real-time database systems. First, in RTSS, read requests are the most common. Furthermore, since write operations are regarded as not time-critical, real-time disk scheduling deals with read operations only. Second, transactions of real-time database systems are unpredictable, while in RTSS transactions are somewhat predictable as the periodic disk requests for a multimedia file are issued. The real-time disk scheduling in RTSS can be optimized by this determinism. Third, hard real-time constraints must be satisfied. Taking the above characteristics into consideration, we present the round scheduling and a disk arm scheduling.

3.2.1 Round Scheduling

T^i_{play}'s of client request r_i, $1 \le i \le n$, are different from each other. If $T^1_{play} = T^2_{play} = \cdots = T^n_{play} = T_{play}$ then at every $m \times T_{play}$ time, a block for each client request is retrieved and consumed, and thus not accummulated. If $T^i_{play} > T^{min}_{play}$, however, data accummulation will occur for r_i[6]. To avoid this situation, the blocks to be retrieved in each round must be scheduled. We call this procedure *round scheduling*. Fig. 3 shows a simple scenario where $m \times T^1_{play} = 1$, $m \times T^2_{play} = 2$, $m \times T^3_{play} = 3$, $m \times T^4_{play} = 1.5$. As shown in this example, B^1_1, B^2_1, B^3_1, B^4_1 are retrieved in the first round and B^1_2, B^4_2 in the second round.

[6] In [1], the solution to this problem is only to stop reading. This will result in underutilization of the disk bandwidth.

round:	(1)	(2)	(3)	(4)	(5)	(6)	(7)	$\cdots$
r_1 :	B^1_1	B^1_2	B^1_3	B^1_4	B^1_5	B^1_6	B^1_7	$\cdots$
r_2 :	B^2_1		B^2_2		B^2_3		B^2_4	$\cdots$
r_3 :	B^3_1			B^3_2			B^3_3	$\cdots$
r_4 :	B^4_1	B^4_2		B^4_3	B^4_4		B^4_5	$\cdots$
time:	0	1	2	3	4	5	6	$\cdots$

r_1 , r_2 , r_3 , r_4 : client request
B^i_j : j^{th} multimedia block of r_i in a disk

Figure 3: A scenario of the simple round scheduling in a disk

round:	(1)	(2)	(3)	(4)	(5)	(6)	(7)	$\cdots$
r_1 :	B^1_1	B^1_2	B^1_3	B^1_4	B^1_5	B^1_6	B^1_7	$\cdots$
r_2 :	B^2_1		B^2_2		B^2_3		B^2_4	$\cdots$
r_3 :	B^3_1			B^3_2			B^3_3	$\cdots$
r_4 :	B^4_1	B^4_2		B^4_3	B^4_4		B^4_5	$\cdots$
r_5 :		B^5_1	B^5_2		B^5_3	B^5_4		$\cdots$
r_6 :		B^6_1		B^6_2		B^6_3		$\cdots$
time:	0	1	2	3	4	5	6	$\cdots$

r_1 , r_2 , r_3 , r_4 , r_5 , r_6 : client request
B^i_j : j^{th} multimedia block of r_i in a disk

Figure 4: A scenario of the efficient round scheduling in a disk

As only two blocks are retrieved in the second round, there exists disk idle time. Hence, client requests that have failed the feasibility test can be serviced in this idle time. For instance, if we assume that four clients can be simultaneously serviced or that four blocks can be retrieved in a round, and if r_5 with $m \times T^5_{play} = 1.5$ and r_6 with $m \times T^6_{play} = 2$ arrive, then the feasibility test for these two requests will fail. r_5 and r_6, however, can be serviced in the disk idle time. Fig. 4 shows a scheduling for this scenario. Further analysis of these cases results in the following theorem. Let $Q = \{i \mid r_i \text{ is a client request}\}$ and $k_i = T^{min}_{play}/T^i_{play}$.

Theorem 1 *It is possible to service a new request r_a even if the feasibility test fails, if the following relationships hold*:

$$\sum_{i \in Q} k_i \ \le \ 1 \text{ and } a \in Q \qquad (3)$$

Proof k_i for r_i is the average number of blocks retrieved in a round and $k_i \le 1$[7]. If there exists Q such that $\sum_{i \in Q} k_i \le 1$ then the blocks for

[7] For r_i such that $T^i_{play} = T^{min}_{play}$, $k_i = 1$.

$\{r_i | i \in Q\}$ can be retrieved, on the average, within the time of one block retrieval in a round. That is, for r_i, $i \in Q$, $\lfloor k_i/k_{min} \rfloor$ blocks are retrieved at every $\lfloor 1/k_{min} \rfloor$ round, where $k_{min} = \min_{i \in Q}(k_i)$. Thus r_a can be serviced. □

Let us apply Theorem 1 to r_5 and r_6 of Fig. 4. Since $k_2 = 1/2$, $k_3 = 1/3$, $k_5 = 2/3$, $k_6 = 1/2$, there exist Q_1 and Q_2 for r_5 and r_6, respectively, such that $Q_1 = \{3, 5\}$, $Q_2 = \{2, 6\}$. Hence r_2 and r_6 are serviced at every alternate rounds. A block for r_3 and two blocks for r_5 are retrieved in three rounds, and these three rounds are repeated. The algorithm for an efficient round scheduler is shown in Fig. 5.

3.2.2 Disk Arm Scheduling

The major goal in designing RTSS is to maximize the number of clients which can be serviced simultaneously. For this, we make use of an elevator-type scheduling scheme[4], which is an adaptation of the SCAN algorithm. A disk arm scheduling algorithm which optimizes buffer requirements is proposed in [5]. In the storage server that must service as many client as possible, however, the optimization of seek time is of major issue. In the elevator-type scheduling scheme, the ordering of retrievals is arranged according to their position on the disk in such a way that the disk arm picks up data as it scans from one end of the disk to the other end. In this arrangement the seek time of a round is fixed at T_{seek_max}[8]. Since the block retrieval within deadline is guaranteed in the upper layer(multimedia file manager), this disk arm scheduling satisfies the hard real-time constraints. Another advantage is the independence of the disk layout. The operation of the disk arm scheduler is described in Fig. 6, which can be asserted to be optimal as follows:

Theorem 2 *The disk arm scheduler of Fig. 6 optimizes seek time. That is, the greatest lower bound of the time for n retrievals in a round is* $T_{seek\_max} + n \times s/R$.

Proof The time for n retrievals consists of the seek time

$$\sum_{i=1}^{n} T_{seek}^{i} \tag{4}$$

and the data transmission time

$$n \times \frac{s}{R} \tag{5}$$

[8] For voice coil actuators, the seek time is given by a non-linear function[12]. In that case, T_{seek_max} is defined to be (no. of cylinders × seek time between adjacent cylinders).

```
procedure Round_Scheduling
(n, {T^1_play, T^2_play, ..., T^n_play}, l, {Q_1, Q_2, ..., Q_l})

begin
    /* initialize k_i , S_i */
    T^min_play = min_{1≤n}(T^i_play);
    for i := 1 to n do
        k_i := T^min_play / T^i_play;
        S_i := 0;
    endfor

    /* initialize start_round_i */
    for j := 1 to l do
        tmp := 1;
        for each i in Q_j do
            start_round_i := tmp;
            tmp := tmp + 1;
        endfor
    endfor
    for i := 1 to n do
        if ( start_round_i = 0 ) then
            start_round_i := 1;
    endfor

    /* main loop */
    round := 1;
    while ( all requests have been serviced ) do
        for i := 1 to n do
            if ( round = start_round_i ) then
            /* start of service */
                S_i := S_i + 1;
                schedule i;
            endif
            if ( round > start_round_i ) then
                if ( S_i < k_i ) then
                    S_i := S_i + 1;
                    schedule i;      /* prefetch */
    /* Retrieve a block for r_i in round^th round */
                endif
                /* a block consumed */
                S_i := S_i - k_i;
            endif
        endfor
        Send scheduled list
            to Disk Arm Scheduler;
        round := round + 1;
    endwhile
end
```

Figure 5: The algorithm of the round scheduling

```
procedure Disk_Arm_Scheduler()
begin
    arm_status := OUTMOST;
    Move disk arm outermost;
    while ( True ) do
        Get {B_1, B_2, ..., B_n}
            from Round Scheduler;
        {B'_1, B'_2, ..., B'_n} :=
            sort({B_1, B_2, ..., B_n}, arm_status);
        for each B'_i in {B'_1, B'_2, ..., B'_n}
            Read block B'_i;
        endfor
        /* Move disk arm innermost or outermost
            for the next round */
        if ( arm_status = OUTMOST )
            Move disk arm innermost;
            arm_status = INMOST;
        else      /* arm_status = INMOST */
            Move disk arm outermost;
            arm_status = OUTMOST;
        endif
    endwhile
end
```

Figure 6: The algorithm of the disk arm scheduling

Since the data transmission time in Eq. (5) is constant we must minimize the seek time in Eq. (4) for optimization. Our disk scheduling algorithm arranges requests according to the seek direction. Thus, the time for the n retrievals in a round is $T_{seek\_max} + n \times s/R$. The greatest lower bound of Eq. (4) is T_{seek_max} in the case that two requests are issued for a block in the innermost cylinder and a block in the outermost cylinder. Hence, this algorithm optimizes seek time. □

Integration of the round scheduling and the disk arm scheduling of disk managers allows us to extend Eq. (2) for feasibility test as follows:

$$T_{seek\_max} + n \times \frac{s}{R} \leq m \times T_{play}^{min} \text{ or } \exists Q \text{ such that } \sum_{i \in Q} k_i \leq 1 \quad (6)$$

4 Discussions on the Disk Architecture

In this section, we show that disk array is the most appropriate architecture for RTSS through the performance comparison of SLED(Single Large Expensive Disk) and disk array. The metric for the performance analysis is the maximum number of clients that can be serviced simultaneously and this value is derived from the feasibility test condition. The use of our proposed algorithms for disk scheduling makes the performance comparison complex, while the use of FCFS will simplify and clarify this process and will yield the same results as in the case of the proposed round scheduling and SCAN-like disk arm scheduling.

In Section 3.1.1 the feasibility test condition for n client requests has been derived under disk array architecture using the block-striping method.

$$n \times T_{seek\_max}^{small} + n \times \frac{s}{R^{small}} \leq m \times T_{play}^{min} \quad (7)$$

Hence, the maximum of n, n_{max}, is

$$n_{max}^{block\_striping} = \left\lfloor \frac{m \times T_{play}^{min}}{T_{seek\_max}^{small} + s/R^{small}} \right\rfloor \quad (8)$$

In SLED, if only one block is retrieved after a seek, this will cause a high overhead. We assume that k blocks are retrieved. But, unlike other applications, the size of a disk block for multimedia playback applications is known to be 60 KB or more. In consideration of buffer size, k is smaller than m, the number of disks in disk array. The condition for servicing n client requests and n_{max} in SLED are as follows:

$$n \times T_{seek\_max}^{large} + n \times k \times \frac{s}{R^{large}} \leq k \times T_{play}^{min} \quad (9)$$

$$n_{max}^{SLED} = \left\lfloor \frac{k \times T_{play}^{min}}{T_{seek\_max}^{large} + k \times s/R^{large}} \right\rfloor \quad (10)$$

We now consider the byte-striping method where a byte is used as a striping unit, that is, level 3 RAID. This architecture shows good performance for large data transfer of supercomputer I/O[8]. Byte-striping can be modeled on SLED which have a data transmission rate m times larger than the original one. n_{max} can be calculated to be

$$n_{max}^{byte\_striping} = \left\lfloor \frac{k \times T_{play}^{min}}{T_{seek\_max}^{small} + k \times s/(m \cdot R^{small})} \right\rfloor \quad (11)$$

From Eqs. (8), (10), and (11), we can compare the performance of disk architecture. For disk parameters, IBM 3380 disk model AK4 is used for SLED and Conner Peripherals CP 3100 for disk array. Hence $R^{large} : R^{small} = 3 : 1$, and $T_{seek\_max}^{large} : T_{seek\_max}^{small} = 2 : 1$[8]. We now show that disk arrays best fit the need of RTSS.

Theorem 3 *The most appropriate disk architecture for RTSS is disk array with three or more*

disks using block-striping method. That is, $n_{max}^{SLED} \leq n_{max}^{block\_striping}$, $n_{max}^{byte\_striping} \leq n_{max}^{block\_striping}$ ($k \leq m$).

Proof From Eqs. (10) and (11), we obtain

$$\begin{array}{lll} n_{max}^{SLED} < n_{max}^{byte\_striping} & \text{if} & m \geq 3 \\ n_{max}^{SLED} > n_{max}^{byte\_striping} & \text{if} & m = 2 \end{array} \quad (12)$$

and from Eqs. (8) and (11), we obtain

$$\begin{array}{lll} n_{max}^{byte\_striping} \leq n_{max}^{block\_striping} & \text{if} & k \leq m \\ n_{max}^{byte\_striping} > n_{max}^{block\_striping} & \text{if} & k > m \end{array} \quad (13)$$

If $m \geq 3$ and $k \leq m$, we derive the following relation from Eqs. (12) and (13).

$$n_{max}^{SLED} < n_{max}^{byte\_striping} \leq n_{max}^{block\_striping} \quad (14)$$

□

5 Conclusions

In this paper, we present the design of a real-time storage server and give the results of its analysis. In order to achieve high capacity and wide transmission bandwidth, we use the disk array. Under this architecture, feasibility test condition has been derived to satisfy the continuity constraint. In addition, we propose the round scheduling algorithm that minimizes disk idle time and the disk arm scheduling algorithm that optimizes seek time. We are now implementing a prototype RTSS testbed to verify our approach. It is designed to run on UNIX and access four SCSI disks via device driver for block-striping.

References

[1] P. Lougher and D. Shepherd, "The design and implementation of a continuous media storage server," *Proc. 3rd International Workshop on Network and OS Support for Digital Audio and Video*, San Diego, California, pp. 63–74, Nov. 1992.

[2] P. V. Rangan and H. M. Vin, "Desining file systems for digital video and audio," *Proc. 13th ACM Symposium on Operationg Systems Principles (SOSP'91), Operating Systems Review*, Vol.25, No.5, pp. 81–94, Oct. 1991.

[3] P. V. Rangan, H. M. Vin, and S. Ramanathan, "Designing an on-demand multimedia service," *IEEE Communications Magazine*, Vol.30, No.7, pp. 56–65, Jul. 1992.

[4] D. D. Kandlur, M.-S. Chen, and Z.-Y. Shae, "Design of a multimedia storage server," *IBM Research Report*, Jun. 1991.

[5] M.-S. Chen, D. D. Kandlur, and P. S. Yu, "Optimization of the grouped sweeping scheduling (GSS) with heterogeneous multimedia streams," *Proc. First International Conference on Multimedia*, Anaheim, California, pp. 235–242, Aug. 1993.

[6] D. P. Anderson and G. Homsy, "A continuous media I/O server and its synchronization mechanism," *IEEE Comp., Spec. Issue on Multimedia Info. Sys.*, pp. 51–57, Oct. 1991.

[7] P. V. Rangan and S. Ramanathan, "Rate-based feedback techniques for continuity and synchronization in multimedia retrieval over high-speed networks," *UC San Diego Technical Report No. CS92–230*, Mar, 1992.

[8] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," *ACM SIGMOD*, pp. 109–116, Jun. 1988.

[9] P. M. Chen, G. A. Gibson, R. H. Katz, and D. A. Patterson, "An evaluation of redundant arrays of disks using an Amdahl 5890," *ACM SIGMETRICS*, pp. 74–85, 1990.

[10] R. K. Abbott and H. Garcia-Molina, "Scheduling I/O requests with deadlines: A performance evaluation," *Proc. Real-Time Systems Symposium*, pp. 113–124, 1990.

[11] S. Chen, J. A. Stankovic, J. F. Kurose, and D. Towsley, "Performance evaluation of two new disk scheduling algorithms for real-time systems," *Journal of Real-Time Systems*, Vol. 3, No. 3, pp. 307–336, Sep. 1991.

[12] K. Hwang and H. Shin, "Real-time disk scheduling based on urgent group and shortest seek time first," *Proc. 5th Euromicro Workshop on Real-Time Systems*, pp. 124–130, 1993.

[13] J. C. Pasquale, G. C. Polyzos, E. W. Anderson, K. R. Fall, J. S. Kay, V. P. Kompella, S. R. McMullan, and D. Ranganathan, "Network and operating system support for multimedia applications," *UC San Diego Technical Report No.CS91–186*, Mar. 1991.

[14] B. Wolfinger and M. Morau, "A continuous media data transport service and protocol for real-time communication in high speed networks," *Proc. 2nd International Workshop on Network and OS Support for Digital Audio and Video*, Heidelberg, Germany, Nov. 1991.

[15] D. L. Gall, "MPEG: A video compression standard for multimedia applications," *Communications of ACM*, Vol. 34, No. 4, pp. 35–45, Apr. 1991.