Multimedia Systems (1997) 5:310–323

Multimedia Systems

Scheduling and data layout policies for a near-line multimedia storage architecture

Siu-Wah Lau, John C.S. Lui

Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong

Abstract. Recent advances in computer technologies have made it feasible to provide multimedia services, such as news distribution and entertainment, via high-bandwidth networks. The storage and retrieval of large multimedia objects (e.g., video) becomes a major design issue of the multimedia information system. While most other works on multimedia storage servers assume an on-line disk storage system, we consider a two-tier storage architecture with a robotic tape library as the vast near-line storage and an on-line disk system as the front-line storage. Magnetic tapes are cheaper, more robust, and have a larger capacity; hence, they are more cost effective for large scale storage systems (e.g., video-on-demand (VOD) systems may store tens of thousands of videos). We study in detail the design issues of the tape subsystem and propose some novel tape-scheduling algorithms which give faster response and require less disk buffer space. We also study the disk-striping policy and the data layout on the tape cartridge in order to fully utilize the throughput of the robotic tape system and to minimize the on-line disk storage space.

Key words: Multimedia storage – Scheduling – Data layout

1 Introduction

In the past few years, we have witnessed tremendous advances in computer technologies, such as storage architectures (e.g. fault-tolerant disk arrays and parallel I/O architectures), high-speed networking systems (e.g., ATM switching technology), compression and coding algorithms. These advances have made it feasible to provide multimedia services, such as multimedia mail, news distribution, advertisement, and entertainment [10], via high bandwidth networks. Consequently, research in multimedia storage system has received a lot of attention in recent years. Most of the recent research works have emphasized the investigation of the design of multimedia storage server systems with magnetic disks as the primary storage. In [2, 7], issues such as real-time playback of multiple audio channels have been studied. In [17], the author presented a technique for storing video and audio streams individually on magnetic disk. The same author proposed in [16] techniques for merging storage patterns of multiple video or audio streams to optimize the disk space utilization and to maximize the number of simultaneous streams. In [9], a performance study was carried out on a robotic storage system. In [4, 5], a novel storage structure known as the staggered striping technique was proposed as an efficient way for the delivery of multiple video or audio objects with different bandwidth demands to multiple display stations. In [8], a hierarchical storage server was proposed to support a continuous display of audio and video objects for a personal computer. In [11], the authors proposed a cost model for data placement on storage devices. Finally, a prototype of a continuous media disk storage server was described in [12].

It is a challenging task to implement a cost-effective continuous multimedia storage system that can store many large multimedia objects (e.g., video), and at the same time, can allow the retrieval of these objects at their playback bandwidths. For example, a 100-min HDTV video requires at least 2 MB/s display bandwidth and 12 GB of storage [3]. A moderate size video library with 1000 videos would then require 12 TB storage. It would not be cost-effective to implement and manage such a huge amount of data all on the magnetic disk subsystem. A cost-effective alternative is to store these multimedia objects permanently in a robotic tape library and use a pool of magnetic disks, such as disk arrays [15], for *buffering* and *distribution*. In other words, the multimedia objects reside permanently on tapes, and are loaded onto the disks for delivery when requested by the disk server. To reduce the tape access delays, the most actively accessed videos would also be stored in the disks on a long-term basis. The disk array functions as a *cache* for the objects residing in the tape library, as well as a buffer for handling the bandwidth mismatch of the tape drive and the multimedia objects.

Given the above architecture, this paper aims at the design of a high-performance storage server with the following requirements:

Correspondence to: J.C.S. Lui
e-mail: cslui@cse.cuhk.edu.hk

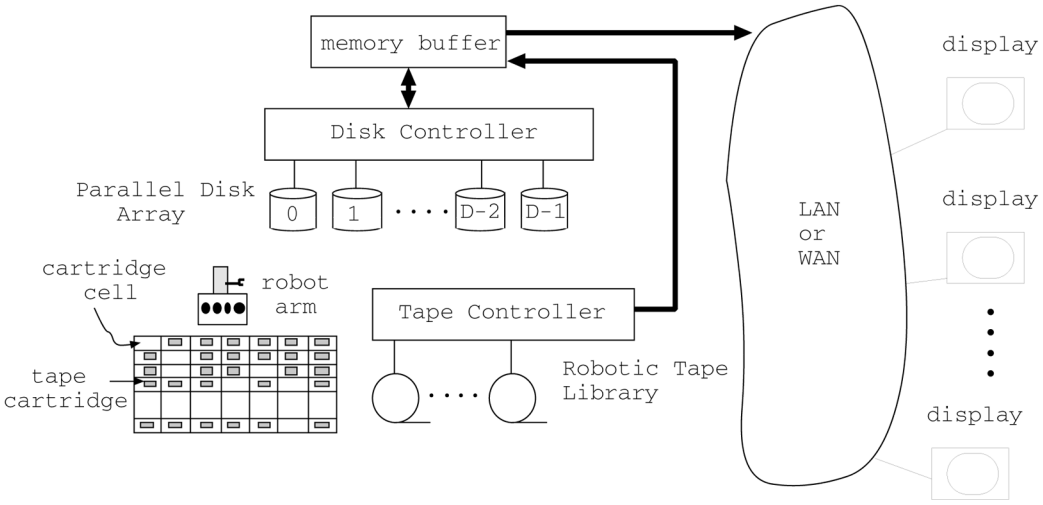


Fig. 1. Cost-effective multimedia storage server

– Minimal disk buffer space between the robotic tape library and the parallel disk array. Disk space is required for handling the bandwidth mismatch of large multimedia objects, such as video or HDTV, and the tape subsystem.
– Minimal response time for the request to the multimedia storage system. The response time of a request to a large multimedia object can be greatly reduced by organizing the display unit, the network device, the parallel disk and the robotic tape library as a pipeline such that data flows at the continuous rate of the display bandwidth of the multimedia object along the pipeline. Since multimedia objects reside in the tape subsystem, to minimize the system response time, we have to minimize the tape subsystem response time. Throughout this paper, the tape subsystem response time is defined as the arrival time of the first byte of data of a request to the disk array minus the arrival time of the request to the multimedia storage system.
– Maximal bandwidth utilization of the tape drives. The current tape library architectures usually have few tape drives. Hence, the bandwidth utilization of tape drives is a major factor of the average response time and throughput of the storage server. A better utilization of the bandwidth of tape drives means a higher throughput of the tape subsystem.

The contribution of this paper is twofold. First, we propose a novel scheduling approach for the tape subsystem, and we show that the approach can reduce the system response time, increase the system throughput and lower the disk buffer requirement. Secondly, we study the disk block organization of the disk subsystem and show how it can be incorporated with the tape subsystem to support concurrent upload and playback of large multimedia objects.

The organization of the paper is as follows. We describe the architecture of our multimedia storage system and present the tape subsystem scheduling algorithms in Sects. 2 and 3, respectively. Then, we discuss the disk buffer requirement for supporting various tape subsystem scheduling algorithms in Sect. 4. In Sect. 5, we describe the disk block organization and the data layout on the tape cartridge for supporting concurrent upload and playback of large multimedia objects. In Sect. 6, we discuss the performance study, and lastly the conclusion is given in Sect. 7.

2 Multimedia storage system architecture

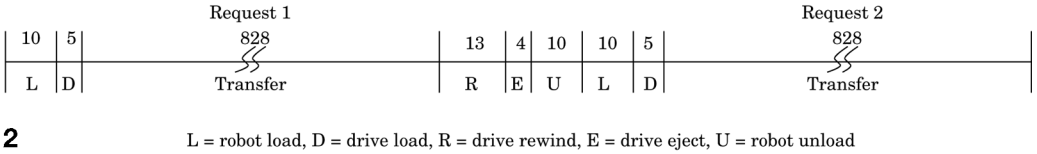
Our multimedia storage system consists of a robotic tape library and a parallel disk array. The robotic tape library has a robotic arm, multiple tape drives, tape cartridges on which multimedia objects reside, and tape cartridge storage cells for placing tape cartridges. Figure 1 illustrates the architectural view of the multimedia storage server. The robotic arm, under computer control, can load and unload tape cartridges. To load a tape cartridge into a tape drive, the system performs the following steps.

1. Wait for a tape drive to become available.
2. If a tape drive is available but occupied by another tape (e.g. this is the tape that was uploaded for a previous request), eject the tape in the drive and unload this tape to its storage cell in the library. We call these operations the *drive eject* operation and the *robot unload* operation, respectively.
3. Fetch the newly requested tape from its storage cell and load it into the ready tape drive. We call these operations the *robot load* operation and the *drive load* operation, respectively.

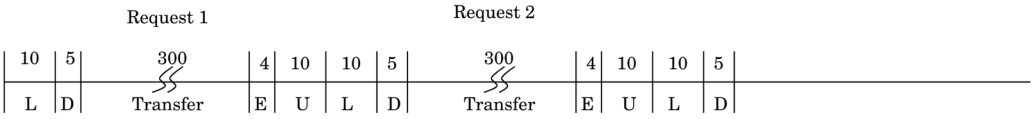
When a multimedia object is requested, the multimedia object is first read from the tape and stored in the disk drives via the memory buffer and the CPU. Then the multimedia object is played back by retrieving the data blocks of the multimedia object from the disk drives, at a continuous rate of the object bandwidth, into the main memory, while the storage server sends the data blocks in the main memory to the playback unit via the network interface. Frequently accessed multimedia objects can be *cached* in the disk drives to reduce tape access and improve system response time as well as throughput.

We define the notations for the robotic tape library in Table 1. These notations are useful for the performance study in later sections.

It is important to point out that the parameter values of a robotic tape library can vary greatly from system to



2 L = robot load, D = drive load, R = drive rewind, E = drive eject, U = robot unload



3 L = robot load, D = drive load, R = drive rewind, E = drive eject, U = robot unload

Fig. 2. the conventional tape scheduling algorithm

Fig. 3. The time-slice tape scheduling algorithm

Table 1. Notations used for the robotic tape library

N_r	number of robotic arms
N_t	number of tape drives
T_l	drive load time
T_e	drive eject time
T_r	tape rewind time
T_u	robot load or unload time
B_t	tape drive transfer rate
$B_d(O)$	display bandwidth of object O
$S(O)$	size of object O

Table 2. Typical parameter values of two commercial storage libraries

Parameter	Exabyte120	Ampex DST800
N_r	1	1
N_t	1 to 4	1 to 4
average T_l	35.4	5
average T_e	16.5 s	4 s
T_r (1/2 tape)	75 s	12-13 s
average T_s	45 s	15 s
T_u	22 s	$<$ 10 s
B_t	0.47 MB/s	14.5 MB/s
Number of tapes	116	256
Tape capacity	5 GB	25 GB

system. For instance, Table 2 shows the typical numbers for two commercial storage libraries.

3 Tape subsystem and scheduling algorithms

In this section, we describe several tape drive scheduling algorithms for our multimedia storage system. A typical robotic tape library has one robot arm and a small number of tape drives. A request to the tape library demands reading (uploading) a multimedia object from a tape cartridge. The straight-forward algorithm or the conventional algorithm to schedule a tape drive is to serve requests one by one, i.e., the tape drive reads the whole multimedia object of the current request to the disk array before reading the multimedia object of the next request in the queue. Since the number of tape drives is small and the reading time of a multimedia object is quite long[1], a new request will often have to wait for an available tape drive. The conventional algorithm performs reasonably well when the tape drive has a bandwidth lower than the display bandwidth of the multimedia objects being requested. However, the conventional algorithm would not result in the good request response time when the tape drive bandwidth is the total display bandwidth of two or more objects. To illustrate this, suppose the tape library is an Ampex DST800[2] with one tape drive. Consider the situation in which two requests for 100 min of different HDTV video objects, each with a display bandwidth of 2 MB/s. These two requests arrive at the same time when the tape drive is idle. The video object size is equal to the display duration times the display bandwidth, which is equal to 100 $\times$ 60 $\times$ 2 MB = 12000 MB. With the conventional algorithm, the transfer of the first request starts after a robot load operation and a drive load operation. The response time of the first request is: $T_u + T_l$ = 15 s . However, the second request will have to wait for the complete transfer of the first multimedia object request, rewinding that tape (T_r), ejecting that tape from the drive (T_e), and unloading that tape from the tape drive to its cell by the robot (T_u). Then the robot can load the newly requested tape (T_u) and load it into the tape drive (T_l). The response time of the second request is:

$$T_u + T_l + 12000/14.5 + T_r + T_e + 2 * T_u + T_l = 885 \text{ s} .$$

Hence, the average response time of the two requests is 450 s. This scenario is illustrated in Fig. 2.

The major problem about the conventional algorithm is that multiple requests can arrive within a short period of time and the average request response time is significantly increased due to the large service time of individual requests. Since the tape drive of the Ampex system is several times the display bandwidth of the multimedia objects, the tape drive can serve the two requests in a time slice manner such that each request receives about half the bandwidth of the tape drive.

Suppose the tape drive serves the two requests in a time-slice manner with a transfer period of 300 s as illustrated in Fig. 3. The two objects are being uploaded into the disk array at an average rate of 6.5 MB/s [3]. From Fig. 3, the response

[1] It takes 1200 s to upload a 1-h HDTV video object by a tape drive with 6 MB/s bandwidth

[2] The parameter values are in Table 2

[3] The overhead of tape switch is approximately 10% of the transfer time. Hence, the effective bandwidth of the tape drive is 13.05 MB/s or 6.5 MB/s for each object

time of the first and second request are $T_u + T_l$ = 15 s and $T_u + T_l + 300 + T_e + T_u + T_u + T_l$ = 344 s respectively. Hence, the average response time is (15 + 344) / 2 s = 179.5 s or an improvement of 60%. We argue that the time slice scheduling algorithm can be implemented with small overheads. In some tape systems, for instance, the D2 tapes used in the Ampex robot system, have the concept of *zones* [1]. Zones are the places on the tape where the tape can *drift* to when we stop reading from the tape. The function of the zone is that the tape drive can start reading from the zone rather than rewinding to the beginning of the tape when the tape drive reads the tape again.

The time slice algorithm has the following advantages.

- The average response time is greatly improved in light load conditions.
- In the case that the request of a multimedia object can be canceled after uploading some or all parts of the object into the disks (e.g., customers may want to cancel the movie due to emergency or the poor entertainment value of the movie), the waste of tape drive bandwidth for uploading unused parts of multimedia objects is reduced.
- The time slice algorithm requires less disk buffer space than the conventional algorithm. The discussion of disk buffer space requirement is given in Sect. 4.

However, the time slice algorithm requires more tape switches and therefore has a higher tape switch overhead and a higher chance of robot arm contention. Our goal is to study several versions of the time slice scheduling algorithm which can minimize the average response time of requests to the multimedia storage system and, also, find the point of switch from the time slice algorithm to the conventional tape- scheduling algorithm. In the rest of this section, we will describe each scheduling algorithm in detail.

3.1 Conventional algorithms

The conventional algorithm is any non-preemptive scheduling algorithm, such as the first-come-first-served (FCFS) algorithm. As each request arrives, the request joins the request queue. A request in the request queue is said to be *ready* if the tape cartridge of the request is not being used to serve another request. The simplest scheduling algorithm is the FCFS algorithm. The FCFS algorithm selects the oldest ready request in the queue for reading when a tape drive is available. A disadvantage of the FCFS algorithm is that the response time of a short request can be greatly increased by any preceding long requests [18].

Another possible conventional algorithm is the shortest-job-first (SJF) algorithm. The SJF algorithm improves the average response time by serving the ready request with the shortest service time, where the service time of a request is the time required to complete the tape switch, the data transfer, and the tape rewind operation of the request. However, a risk of using the SJF algorithm is the possibility of starvation for longer requests as long as there is steady supply of shorter requests.

The implementations of the FCFS and SJF algorithms are similar. We have to separate the implementation into two cases: (1) where there is only a single tape drive available for the tape subsystem and, (2) where there are multiple tape drives in the tape subsystem.

Single tape drive. The implementation of the conventional algorithms is straight-forward:

```
procedure conventional();
begin
    while true do
    begin
        if (there is no ready request) then
            wait for a ready request;
        get a ready request from the request queue;
        serve the request;
    end;
end;
```

Multiple tape drives. The implementation of the conventional algorithms consists of several procedures. The procedure `robot` is instantiated once and procedure `tape` is instantiated N_t times, where each instance of procedure `tape` corresponds to a physical tape drive and each instance has an unique ID.

```
procedure conventional()
begin
    run robot() as a process;
    for i := 0 to NUM_TAPE -1 do
        run tape(i) as a process;
end;

procedure robot();
begin
    while true do
    begin
        /* accept new request */
        if (a request is ready and
            a tape drive is available) then
        begin
            get a request from the request queue;
            send the request to an idle tape drive;
        end
        else
            if (an available drive is occupied) then
                perform the drive unload operation
                    and the robot unload operation;
            else
                wait for a ready request
                    or an occupied available drive;
    end;
end;

procedure tape(integer id);
begin
    while true do
    begin
        wait for a request from robot arm;
        serve the request;
    end;
end;
```

3.2 Time slice algorithms

The time slice algorithms classify requests into two types: (1) non-active requests and (2) active requests. Newly arrived requests are first classified as non-active requests and put into

the request queue. A non-active request is said to be ready when the tape cartridge is not being used for serving another request. Active requests are those requests being served by the tape drive in a time slice manner.

Since the time slice algorithms are viable only if the tape switch overhead is small, we restrict that the tape rewind operation to be performed when a request has been completely served and the tape search operation is performed only at the beginning of the service of a request. This implies that two requests of the same tape cannot be served concurrently. Note that the chance of having two requests of the same tape in the system is very small, because (1) the access distribution of objects is highly skewed, since video rental statistics suggest some highly skewed access distributions, such as the 80/20 rule, in which 80% of accesses go to the most popular 20% of the data [6] and, (2) frequently accessed objects are kept in the disk drives.

The tape switch time is equal to the total time to complete a tape drive eject operation, a robot unload operation, a robot load operation, a tape drive load operation and a tape search operation. In the remainder of the paper, we let H to be the maximum tape switch time. The time slice algorithms break a request into many tasks, each with a unique task number. Each task of the same request is served separately in the order of increasing task number. Each request is assigned a time slice, s, which is the maximum service time of a task of the request. The service time of a task includes the time required for the tape switch and the data transfer of the task. For the last task of a request, the service time also includes the time required for a tape rewind operation. There are many possible ways to serve several requests in a time slice manner. We concentrate on two representative time slice algorithms: the *round-robin* (RR) algorithm and the *least slack* (LS) algorithm.

3.2.1 Round-robin algorithm

In this section, we formally describe the RR algorithm.

Let $R_1, \ldots, R_n$ be the active requests and $R_{n+1}, ..., R_m$ be the ready non-active requests, where $m \geq n$. Let O_i be the video object requested by R_i for $i = 1, ..., m$. Let $s_1, \ldots, s_n$ be the time slices assigned to $R_1, \ldots, R_n$, respectively.

With the RR algorithm, the active requests are served in an RR manner. In each round of service, one task of each active request will be served. The active requests are served in the same order in each round of service. In order to satisfy the bandwidth requirement of active request R_i, the average transfer bandwidth allocated for R_i must be geater than or equal to the bandwidth of R_i. Formally speaking, the bandwidth requirement of R_i is satisfied if

$$\frac{(s_i - H)B_t}{\sum_{k=1}^{n} s_k} \geq B_d(O_i).$$

The RR algorithm maintains the following condition:

$$\frac{(s_i - H)B_t}{\sum_{k=1}^{n} s_k} \geq B_d(O_i) \quad \text{for } 1 \leq i \leq n$$

The condition guarantees that the bandwidth requirements of the active requests are satisfied. The efficiency of the algorithm is defined as:

$$E = 1 - \frac{nH}{\sum_{i=1}^{n} s_i} \tag{1}$$

When the system is lightly loaded, the tape drive can serve at least one more request in addition to the currently active requests, the average response time is reduced for a smaller time slice, because a new arrival is less likely to have to wait for a long period. However, a smaller time slice means that a smaller number of active requests can be served simultaneously, thereby increasing the chance that a newly arrived request has to wait for the completion of an active request. Therefore, different time slices or different efficiencies of the time slice algorithm are required to optimize the average response time at different load conditions. To simplify our discussion, we assume each request has the same time slice in the rest of the paper, unless we state otherwise.

The specification of the RR algorithm is:

Simple RR Algorithm. *The algorithm assigns each active request a time slice period of $s > H$ s which has to satisfy the following conditions.*

Condition 1. The tape drive serves requests $R_1, ..., R_n$ in a round-robin manner with a time slice period of s s.

Condition 2. In each time slice period, the available time for data transfer is $s - H$ if the task being served is not the last task of an active request, otherwise, the available time for data transfer is $s - H -$ the rewind time of the tape.

Condition 3. Request R_{n+1} which is the oldest ready non-active request becomes active if

$$\frac{(s - H) * B_t}{(n + 1)s} \geq \max_{i=1,\ldots,n+1} \{B_d(O_i)\}$$

The straight-forward implementation of the simple RR algorithm is to consider whether more active requests can be served concurrently at the end of a service round, i.e., the algorithm evaluates Condition 3 at the end of each service round. We call this implementation the RR-1 algorithm. Again, we separate the implementation into two cases: (1) where there is only a single tape drive in the tape subsystem, and (2) where there are multiple tape drives in the tape subsystem.

Single Tape Drive

```
procedure RR-1();
begin
    while true do
    begin
        if (there is no active request
            and ready non-active request) then
                wait for a ready non-active request;
        if (the last active request is served
            and Condition 3 of the
            Simple Round-robin Algorithm is satisfied) then
                accept a ready non-active request;
        get a task from the active task queue;
        serve the task;
    end;
end;
```

With the RR-1 algorithm, a newly arrived request has to wait for one half of the duration of a service round when the tape subsystem can serve at least one more request in addition to the currently active requests. Since the duration of a service round grows linearly with the number of active requests, the average waiting time of a request is high when there are several active requests. To improve the above situation, we can check whether one more request can be served by the tape subsystem after every completion of an active task. We call this improved implementation of the RR algorithm the RR-2 algorithm.

Multiple tape drives. For the case of multiple tape drives, we have to consider the robot arm contention, because the tape drives need to wait for the robot arm to load or unload. In the worst case, each tape switch requires a robot load operation and a robot unload operation. Therefore, the worst case robot waiting time is $2 \times T_u \times (N_t - 1)$. Hence, Condition 3 can be revised to become:

$$\frac{(s_i - H - 2(N_t - 1)T_u)B_t}{\sum_{k=1}^{n} s_k} \geq B_d(O_i) \quad \text{for } 1 \leq i \leq n \ ,$$

3.2.2 The least-slack (LS) algorithm

Let us study another version of time slice algorithm which can improve on the response time of the multimedia request. In order to maintain the playback continuity of an object, task i of the request of the object must start to transfer data before finishing the playback of the data of the previous task $i-1$. We define the latest start time of transfer (LSTT) of a task of an active request as the latest time that the task has to start to transfer data in order to maintain the playback continuity of the requested object. Formally, the LSTT of task J_i is defined as:

$$LSTT(J_i) = \begin{cases} \text{request arrival time + request response time} \\ \quad \text{if } J_i \text{ is the first task} \\ \\ LSTT(J_{i-1}) + \text{playback time of the data of task } J_{i-1} \\ \quad \text{otherwise} \end{cases}$$

The slack time of a task is defined as $max(LSTT\ of\ task - current\ time, 0)$. Let $transfer(J_i)$ be the time required to complete the data transfer of J_i and the tape rewind operation of J_i (if J_i is the last task of a request). The deadline of a task J_i is defined as

$$deadline(J_i) = LSTT(J_i) + transfer(J_i) \ . \qquad (2)$$

A ready non-active request R can become active when the tasks of R can be served immediately such that each task of an active request can be served at or before its LSTT.

LS Algorithm *The algorithm serves requests with the following conditions:*

Condition 1. Each active task can be served in one time slice of s s.

Condition 2. Active tasks are served in ascending order of slack time.

Condition 3. In each time slice, the available time for data transfer is $s - H$ if the task being served is not the last task of an active request, otherwise, the available time for data transfer is $s - H -$ the rewind time of the tape.

Condition 4. The data transfer of each active task can start at or before the LSTT of the active task.

Condition 5. A ready non-active request can become active if Condition 4 is not violated after the request has become active.

We choose the LS algorithm for tape scheduling because it is optimal for a single tape system [14][4] in the sense that, if scheduling can be achieved by any algorithm, it can be achieved by the optimal algorithm.

For the case that the tape subsystem has only one robot arm and one tape drive, Condition 4 of the LS Algorithm can be rewritten as follows.

Lemma 1. *Given a robotic tape library with a single tape drive, let $J_1, \ldots, J_n$ be the active tasks listed in ascending order of slack time. If no active task is in service, then Condition 4 of the LS algorithm is equivalent to the condition that each active task can be completed at or before its deadline. In other words, Condition 4 of the LS algorithm is equivalent to the following condition:*

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} (transfer(J_i) + switch(J_i)) \leq deadline(J_k) - \textit{current time}$$

where $switch(J_i)$ is the tape switch time of J_i.

Proof. Assume there is no active task in service. By Eq. 2, an active task can start data transfer at or before its LSTT if and only if it can be completed at or before its deadline. A task J_k can be completed at or before its deadline if and only if the time between the current time and the deadline of J_k is enough to complete J_k and its preceding tasks. Therefore, Condition 4 of the LS algorithm is equivalent to

$$\forall k, 1 \leq k \leq n, \quad \sum_{i=1}^{k} (transfer(J_i) + switch(J_i)) \leq deadline(J_k) - \textit{current time}$$

Again, we separate the implementation into two cases, (1) where there is only a single tape drive in the tape subsystem and, (2) where there are multiple tape drives in the tape subsystem. The implementation of the LS algorithm for the single tape case is as follows:

Single Tape Drive

```
procedure LS();
begin
    while true do
    begin
        if (there is no request) then
            wait for a new request;
        if (there is a ready request is non-empty
```

[4] The paper discussed scheduling in single and multiple processors. The case of a single tape drive robot library is equivalent to the case of a single processor described in the paper

```
            and acceptnew()) then
            begin
                get the oldest ready request;
                put the tasks of the request
                  into the active task queue;
            end;
            get the active task with the least slack time;
            serve the task;
        end
end

function acceptnew() : boolean;
begin
        float work;
        pointer x;
        if (the active task queue is empty) then
            return(true);
        work := 0.0;
        save the active task queue;
        put the tasks of the oldest ready request into the active task queue;
        while task queue is not empty do
        begin
            x := next active task;
            work := work + x->deadline - x->LSTT + tape switch time;
            if (work > x->deadline - current time) then
            begin
                restore the active task queue;
                return(false)
            end;
        end;
        restore the active task queue;
        return(true);
end
```

Multiple tape drives. This implementation consists of two procedures: `robot` and `tape drive`. Procedure `robot` performs the following steps repeatedly: accept a ready request if the request can be accepted to become active immediately; if there are active tasks and an idle tape, then send the active task with the least slack time to an idle tape, else wait for an idle tape or an active task. Procedure `tape` repeatedly waits for an active task and performs the sequence of a drive eject operation, a drive load operation, a data transfer, and a tape rewind operation (for the last task of a request). Procedure `robot` is instantiated once and procedure `tape` is instantiated N_t times. Each instance of procedure `tape` has a unique ID.

4 Disk buffer space requirement

In this section, we study the disk buffer requirement for the various scheduling algorithms we have described. First, we show that the conventional algorithm (the FCFS or the SJF algorithm) requires a huge amount of buffer space to achieve the maximum throughput. The following theorem states the buffer space requirement for the conventional algorithm.

Theorem 1. *If each object of a request is of the same size S and same display bandwidth $B_d(O)$, then the conventional algorithm requires $O(\frac{B_t^* S}{2B_d(O)})$ disk buffer space in order to achieve its maximum throughput, where the sustained tape throughput is B_t^* and is equal to $\frac{SB_t}{S+B_t(H+T_r)}$.*

Proof. The tape subsystem achieves its maximum throughput when (1) there is an infinite number of ready requests and (2) each request does not have a search time, i.e., the requested object resides at the beginning of the tape cartridge and the tape drive can start to read the object right after the drive load operation has been done. The sustained bandwidth of tape subsystem is:

$$B_t^* = B_t \frac{S/B_t}{S/B_t + H + T_r} = \frac{SB_t}{S + B_t(H + T_r)} .$$

At time $t = 0$, the tape subsystem is idle and starts to serve requests one by one. In time interval (0, $\frac{S}{B_t^*}$), data are consumed at the rate of $B_d(O)$ and uploaded at the rate of B_t^*. Hence, at time $t = \frac{S}{B_t^*}$, $\frac{(B_t^* - B_d(O))S}{B_t^*}$ buffer space is required to hold the accumulated data. In time interval $[\frac{S}{B_t^*}, \frac{2S}{B_t^*})$, data are consumed at the rate of $2B_d(O)$ and uploaded at the rate of B_t^*. Therefore, at time $t = \frac{2S}{B_t^*}$, $\frac{(B_t^* - B_d(O))S}{B_t^*} + \frac{(B_t^* - 2B_d(O))S}{B_t^*}$ buffer space is required to hold the accumulated data. This argument continues until the total object display throughput matches with the tape sustained throughput. To obtain the upper bound buffer requirement, assume we have a tape system whose sustained throughput $B_t^u = k_1 B_d(O)$, where k_1 satisfies the following criterion

$$k_1 = \lceil B_t^* / B_d(O) \rceil ,$$

then the upper bound buffer requirement is:

$$\sum_{n=1}^{k_1} \left(\frac{B_t^u - nB_d(O)}{B_t^u} \right) S = \sum_{n=1}^{k_1} \left(\frac{k_1 B_d(O) - nB_d(O)}{k_1 B_d(O)} \right) S$$
$$= k_1 S - \frac{k_1(k_1 + 1)}{2k_1} S = \frac{(k_1 - 1)S}{2}$$

To obtain the lower bound buffer requirement, assume we have a tape system whose sustained throughput $B_t^l = k_2 B_d(O)$ satisfies the following criterion

$$k_2 = \lfloor B_t^* / B_d(O) \rfloor ,$$

then the lower bound buffer requirement is:

$$\sum_{n=1}^{k_2} \left(\frac{B_t^l - nB_d(O)}{B_t^l} \right) S = \sum_{n=1}^{k_2} \left(\frac{k_2 B_d(O) - nB_d(O)}{k_2 B_d(O)} \right) S$$
$$= k_2 S - \frac{k_2(k_2 + 1)}{2k_2} S = \frac{(k_2 - 1)S}{2}$$

Therefore, the buffer space requirement is $O(\frac{B_t^* S}{2B_d(O)})$.

For example, if B_t = 15 MB/s, $B_d(O)$ = 2 MB/s, H = 30 s, T_r = 13 s, S = 10800 MB[5], the disk buffer size = 38.22 GB.

Corollary 1. *If there are N_t tape drives in the tape library system. The buffer disk buffer requirement is $O(\frac{N_t B_t^* S}{2B_d(O)})$.*

In the following theorem, we state the disk buffer requirement for the RR time slice algorithm.

Theorem 2. *If R_1, ..., R_n are the active requests that satisfy the condition*

[5] equivalent to 1.5 h of display time

$$\frac{(s_i - H)B_t}{\sum_{k=1}^{n} s_k} \geq B_d(O_i) \quad \textit{for } 1 \leq i \leq n \ ,$$

then the RR algorithm achieves the bandwidth requirements of the requested objects, $O_1, ..., O_n$ iff the disk buffer size is $\sum_{i=1}^{n} 2(s_i - H)B_t$.

Proof. For R_i ($1 \leq i \leq n$), at least two disk buffers of size $(s_i - H)B_t$ are required for concurrent uploading and display of object O_i. Hence, the necessary condition is proved.

Suppose, for each request O_i, there are two disk buffers b_{i1} and b_{i2}, each with size $(s_i - H)B_t$. While one buffer is used for uploading the multimedia object from the tape library, the other buffer is used for displaying object O_i. At steady state, the maximum period between an available buffer and the time of uploading from tape is $\sum_{i=1}^{n} s_i$. When b_{i1} has just been available, the system starts to output data from the other buffer b_{i2} for display. By the condition of the theorem, b_{i2} will not be emptied before the tape drive starts to upload data to b_{i1}. Hence, the bandwidth of O_i is satisfied.

With the same arguments, we have the following corollary for the disk buffer requirement for the LS algorithm.

Corollary 2. *If $R_1, ..., R_n$ are the active requests that satisfy the condition*

$$\frac{(s_i - H)B_t}{\sum_{k=1}^{n} s_k} \geq B_d(O_i) \quad \textit{for } 1 \leq i \leq n \ ,$$

then the LS algorithm achieves the bandwidth requirements of the requested objects, $O_1, \ldots, O_n$ iff the disk buffer size is $\sum_{i=1}^{n} 2(s_i - H)B_t$.

By Theorem 2 and Corollary 2, the LS and RR algorithms require less buffer than the conventional algorithm for the same throughput because the transfer time of each time slice, $s_i - H$, can be chosen to be much smaller than the total upload period of the object, $\frac{S}{B_t^*}$.

5 The disk subsystem

Since the tape drive bandwidth or the object bandwidth can be higher than the bandwidth of a single disk drive, we have to use striping techniques to achieve the required bandwidth of the tape drive or the object. In [4], a novel architecture known as the staggered striping technique was proposed for high-bandwidth objects, such as HTDV video objects. It has been shown that staggered striping has a better throughput than the simple striping and virtual data replication techniques for various system loads [4]. In this section, we show how to organize the disk blocks in staggered striping together with the robotic tape subsystem so that (1) the bandwidths of the disks and the tape drives are *matched*, and (2) concurrent upload and display of multimedia objects is supported.

5.1 Staggered striping

We first give a brief review of the staggered striping architecture. With this technique, an object O is divided into subobjects, U_i, which are further divided into M_O fragments. A fragment is the unit of data transferred to and from a single disk drive. The disk drives are clustered into logical groups. The disk drives in the same logical group are accessed concurrently to retrieve a subobject (U_i) at a rate equivalent to $B_d(O)$. The stride, k, is the distance[6] between the first fragment of U_i and the first fragment of U_{i+1}. The relationships of the above parameters are shown below.

- $M_O = \lceil \frac{B_d(O)}{B_{disk}} \rceil$, where B_{disk} is the bandwidth of a single disk drive.
- The size of a subobject = $M_O \times$ the size of a fragment.
- A unit of time = the time required for reading a fragment from a single disk drive.

Note that a subobject can be loaded from the disk drives into the main memory in one time unit. To reduce the seek and rotational overheads, the fragment size is chosen to be a multiple of the size of a cylinder. A typical 1.2 GB disk drive consists of 1635 cylinders of size 756000 bytes each and has a peak transfer rate of 24 Mbit/second, a minimum disk seek time of 4 ms, a maximum disk seek time of 35 ms, and a maximum latency of 16.83 ms. For a fragment size of 2 cylinders, the maximum seek and latency delay times of the first cylinder and the second cylinder are 16.83+35 = 51.83 ms and 4+16.83 = 20.83 ms respectively. The transfer time of two cylinders is 481 ms. The total service time (including disk seek, latency delay, and disk transfer time) of a fragment is 553.66 ms. Hence, the seek and rotational overheads is about 13% of the disk bandwidth [7]. To simplify our discussion, we assume the fragment size is two cylinders and one unit of time is 0.55 s. To illustrate the idea of staggered striping, we consider the following example.

Example 1. Figure 4 shows the retrieval pattern of a 5.0 MB/s object in five 2.5-MB/s disk drives. The stride is 1 and M_O is 2. When the object is read for display, subobject U_0 is read from disk drives 0 and 1, and so on.

5.2 Layout of storage on the tape

In the following discussion, we assume that (1) staggered striping is used for the storage and retrieval of objects in the disk drives and, (2) the memory buffer between the tape drives and the disk drives is much smaller in size than a fragment.

Let the effective bandwidth for the time slice algorithm be B_t^*, which is equal to $\frac{s-H}{s} B_t$. We show that the storage layout of an object on the tape must match the storage layout on the disk drives so as to achieve maximum throughput of the tape drive. When the object is displayed, each fragment requires a bandwidth of $\frac{B_d(O)}{M_O}$. Therefore, the tape drive produces N_O fragments, where $N_O = \lfloor \frac{B_t^* M_O}{B_d(O)} \rfloor$ in a unit of time. The blocks of N_O fragments are stored in an RR manner such that the N_O fragments are produced as N_O continuous streams of data at the same time. Consider the case described in Example 1. Suppose B_t^* = 7.5 MB/s, then

[6] which is measured in number of disks

[7] A further increase in number of cylinders does not result in much reduction of the overhead. Hence, a fragment of 2 cylinders is a reasonable assumption

		disk				
		0	1	2	3	4
time	0	U0.0	U0.1			
	1		U1.0	U1.1		
	2			U2.0	U2.1	
	3				U3.0	U3.1
	4	U4.1				U4.0
	5	U5.0	U5.1			
	6		U6.0	U6.1		
	7			U7.0	U7.1	
	8				U8.0	U8.1
	9	U9.1				U9.0

4

		disk				
		0	1	2	3	4
time	0	U0.0	U0.1	U1.1		
	1		U1.0	U2.0	U2.1	
	2			U6.1	U3.0	U3.1
	3	U4.1			U7.1	U4.0
	4	U5.0	U5.1			U8.1
	5	U9.1	U6.0	U7.0		
	6		U10.1	U11.1	U8.0	
	7			U12.0	U12.1	U9.0
	8	U10.0			U13.0	U13.1
	9	U14.1	U11.0			U14.0

5

	disk				
	0	1	2	3	4
0	U0.0	U0.1	U1.1	U2.1	U3.1
1	U4.1	U1.0	U2.0	U3.0	U4.0
2	U5.0	U5.1	U6.1	U3.0	U8.1
3	U9.1	U6.0	U7.0	U7.1	U9.0
4	U10.0	U10.1	U11.1	U8.0	U13.1
5	U14.1	U11.0	U12.0	U12.1	U14.0

6

Fig. 4. Retrieval pattern of an object

Fig. 5. Upload pattern of an object

Fig. 6. An example of storage pattern

$N_O = 3$. If the subobjects are stored in the following order: $\{U_{0.0}, U_{0.1}, U_{1.0}\}$, $\{U_{1.1}, U_{2.0}, U_{2.1}\}$, ... [8]. In the first time unit, $U_{0.0}$, $U_{0.1}$, $U_{1.0}$ are read from the tape drive. At the same time, $U_{0.0}$ and $U_{0.1}$ are stored in disk drive 0 and disk drive 1. Fragment $U_{1.0}$ has to be discarded and re-read in the next time unit, because disk drive 1 can only store either $U_{0.1}$ or $U_{1.0}$. Since the output rate of the tape drive must match the input rate of the disk drives, the effective bandwidth of the tape drive is 5 MB/s and the tape drive bandwidth cannot be fully utilized.

On the other hand, if the storage layout of the object is as follows: $\{U_{0.0}, U_{0.1}, U_{1.1}\}$, $\{U_{1.0}, U_{2.0}, U_{2.1}\}$, $\{U_{6.1}, U_{3.0}, U_{3.1}\}$, ... In each time unit, the output fragments from the tape drive can be stored in three consecutive disk drives. Hence, the bandwidth of the tape drive is fully utilized. Figure 5 shows the timing diagram for the upload of the object from the tape drive. From time 2, subobject U_0 can be read from disk drives 0 and 1. Hence, the object can be displayed at time 2, while the remaining subobjects are being uploaded into the disk drives from the tape drive. Both the bandwidth of the disk drives and the tape drive are fully utilized.

Now we should derive the conditions of matching the way that the fragments are retrieved from the disk and the way that the fragments are uploaded from the tape. Let D and k be the number of disk drives of the disk array and the stride, respectively. In the rest of the section, we assume that the bandwidth of tape drive is at least $(M_O+1) \times \frac{B_d(O)}{M_O}$, i.e., $N_O > M_O$.

Definition 1. *Given an object O which has been uploaded from a tape drive into the disk array, the retrieval pattern $\mathscr{R}_\mathscr{O}$ of O is an $L \times D$ matrix, where L is the number of time units required for the retrieval of O from the disk drives and $\mathscr{R}_\mathscr{O}(i,j)$ is equal to "$U_{a.b}$" if fragment $U_{a.b}$ of O is read at time i from disk drive j. $\mathscr{R}_\mathscr{O}(i,j)$ contains a blank entry if no fragment is read from disk drive j at time i.*

Definition 2. *Given an object O, the upload pattern $\mathscr{P}_\mathscr{O}$ of O is an $L \times D$ matrix, where L is the number of time units required for uploading O and $\mathscr{P}_\mathscr{O}(i,j)$ from a tape drive into the disk array is equal to "$U_{a.b}$" if fragment $U_{a.b}$ is read at time i and stored in disk drive j. $\mathscr{R}_\mathscr{O}(i,j)$ contains a blank entry if no fragment is stored in disk drive j at time i.*

Definition 3. *The storage pattern $\mathscr{L}_\mathscr{P}$ of a retrieval or upload pattern $\mathscr{P}$ is an $L \times D$ matrix where L is an integer and $\mathscr{L}_\mathscr{P}(i,j)$ the i-th non-blank entry of column j of $\mathscr{P}$, i.e., $\mathscr{L}_\mathscr{P}$ is obtained by replacing all the blank entries of P by lower non-blanking entries of the same column with the preservation of the row-order of the entries, i.e., $\forall \mathscr{L}_\mathscr{P}(a,b)$ and $\mathscr{L}_\mathscr{P}(c,b)$, $a \neq c$, $P(i,b) = \mathscr{L}_\mathscr{P}(a,b)$ and $P(j,b) = L_\mathscr{P}(c,b)$, $a > c$ iff $i > j$.*

Examples of retrieval and upload patterns are shown in Figs. 4 and 5, respectively. The retrieval and upload patterns of Figs. 4 and 5 have the same storage pattern which is shown in Fig. 6.

Lemma 2. *With staggered striping, when an object O is uploaded from a tape drive into the disk array, the tape drive bandwidth can be fully utilized if*

- *the tape drive reads N_O fragments of O into N_O different disk drives in each unit of time; and*
- *the storage patterns of the retrieval pattern and upload pattern of O are the same, i.e., $\mathscr{L}_{\mathscr{P}_\mathscr{O}} = \mathscr{L}_{\mathscr{R}_\mathscr{O}}$.*

[8] {X Y Z} is a representation which shows that the blocks of X, Y, and Z are stored in an RR manner

Proof. Assume that the retrieval pattern and the upload pattern of O have the same storage pattern and the tape drive reads N_O fragments into N_O different disk drives. Since the retrieval pattern and the upload pattern has the same storage pattern, each uploaded fragment (from the tape drive) can be retrieved from its storage disk for display. Since the tape drive reads N_O fragments in each unit of time and all uploaded fragments (from the tape drive) can be retrieved from the storage disks, the bandwidth of the tape drive is fully utilized.

Definition 4. *An object is said to be uniformly distributed over a set of disk drives if each disk drive contains the same number of fragments of the object.*

Theorem 3. *With staggered striping, when an object O is uploaded from a tape drive to the disk array, the tape drive bandwidth can be fully utilized if*

1. *k and D do not have a common factor greater than 1, i.e., the greatest common divisor (GCD) of k and D is 1, and*
2. *the data transfer period, $s - H$, is a multiple of $\frac{LCM(D,M_O,N_O)}{N_O}$ time units, where $LCM(x, y, z)$ is the least common multiple of integers x, y, z.*

Proof. Suppose the GCD of k and D is 1 and $s - H$ is a multiple of $\frac{LCM(D,M_O,N_O)}{N_O}$ time units.

Consider the case that the object starts to be uploaded at time 0. At time i, N_O fragments have been stored and uniformly distributed into disk drives $(i \times k)\ mod\ D$, $(i \times k+1)\ mod\ D$, ..., $(i \times k+N_O-1)\ mod\ D$. Since the GCD of k and D is 1, $\{0, ..., D-1\}^2 : f(i) = (i \times k)\ mod\ D$ is a one-one mapping. If we extend the domain of f to the set of natural numbers $\mathcal{N}$, then $\forall i, j \in \mathcal{N}, f(i+j \times D) = f(i)$. This implies that $LCM(D, M_O, N_O)$ fragments can be uniformly distributed over the disk drives. Hence, at time $\frac{LCM(D,M_O,N_O)}{N_O} - 1$, $LCM(D, M_O, N_O)$ have been stored and uniformly distributed over the disk drives of the disk buffer.

Consider the case that the object is played back at time 0. At time i, M_O fragments are retrieved from disk drives $(i \times k)\ mod\ D$, $(i \times k+1)\ mod\ D$, ..., $(i \times k+M_O-1)\ mod\ D$. At time $\frac{LCM(D,M_O,N_O)}{M_O} - 1$, $LCM(D, M_O, N_O)$ have been retrieved and $\frac{LCM(D,M_O,N_O)}{D}$ fragments have been retrieved from each disk drive. Let O' be the object consisting of the $LCM(D, M_O, N_O)$ fragments. The following procedure finds the upload pattern $\mathscr{P}_{\mathscr{O}'}$ which has the same storage pattern of the retrieval pattern $\mathscr{R}_{\mathscr{O}'}$:

```
procedure upload(var upattern : upload pattern; rpattern : retrieval pattern);
var
    spattern : storage pattern;
    i, j, c: integer;
    count[D] : integer;
begin
    initialize all the entries in count to 0;
    initialize all the entries in upattern to blank;
    spattern := storage pattern of rpattern;
    for i := 0 to LCM(D,M_O,N_O)/N_O - 1 do
        for j := 0 to N_O - 1 do
        begin
            c := (i*k+j) mod D;
            upattern[i,c] := spattern[count[c],c];
            count[c] := count[c]+1;
        end
end
```

With upload pattern $\mathscr{P}_{\mathscr{O}'}$, the tape reads N_O different fragments into N_O different disk drives in each time unit and the storage pattern of the retrieval pattern and the upload pattern of O' are the same. By Lemma 2, O' can be retrieved with the maximum throughput of the tape drive. Hence, an object of a multiple of the size of O', i.e., $LCM(D, M_O, N_O)$, can be uploaded with the maximum throughput of the tape drive. Thus, if the data transfer period is a multiple of $\frac{LCM(D,M_O,N_O)}{N_O}$, the tape drive bandwidth can be fully utilized.

For the case of Example 1, the data transfer period is a multiple of $\frac{LCM(5,2,3)}{3}$ = 10 time units or 5.5 s. For the case that H = 30 s, a reasonable time slice period is from 200 to 300 s [9]. A video-on-demand system with a capacity of 1000 100-min HDTV videos of 2 MB/s bandwidth requires a storage space of 1000 × 12 MB = 12 TBytes. If 10% of the videos reside on disks, 1.2 TBytes disk space is required. The number of 1.2-GB disk drives of the disk array is 1200, and the data transfer period is a multiple of $\frac{LCM(1200,2,3)}{3}$ = 400 time units = 400 × 0.55 s = 220 s or the time slice is 250 s. Hence, the disk array of 1200 disk drives can be used as a disk buffer as well as a disk cache.

To maximize the tape drive throughput, the maximum output rate of the disk buffer must be at least the maximum utilized bandwidth of the tape drive. The maximum utilized bandwidth of the tape drive is given by $\frac{N_O B_d(O)}{M_O}$. To have an output rate of at least the maximum utilized bandwidth of the tape drive, the disk buffer must support concurrent retrieval of at least $\lceil \frac{N_O}{M_O} \rceil$ subobjects. For each tape drive, the minimum number of required disk drives for buffering is $N_O + \lceil \frac{N_O}{M_O} \rceil \times M_O$.

Video uploading from the tape drive is first stored in the disk array. The playback of the video object can start when the cluster of disk drives for uploading does not overlap with the cluster of disk drives of the first subobject. Hence, the minimum delay, d, of the disk buffer is defined as the smallest integer n such that $\forall 0 \leq i < M_O, (nk + i)\ mod\ \geq M_O$. The stride k should be carefully chosen to minimize the disk buffer delay and improve the overall response time of the storage server.

6 Performance evaluation

We evaluate the the performance of the scheduling algorithms for two values of the tape drive bandwidth, 6 MB/s and 15 MB/s, by computer simulation. We assume that (1) each tape contains only one object, and hence the search time of each request is 0 s and (2) a request never waits for a tape. Since frequently accessed objects are kept in disk drives, the probability that a request has to wait for a tape which is being used to serve another request is very low

[9] For this time slice, the tape switch overhead is about 10–15% of the tape drive bandwidth

Table 3. Simulation parameters

Parameter	Case 1	Case 2
T_l	5 s	5 s
T_e	5 s	5 s
T_r	12 s	12 s
T_s	0 s	0 s
T_u	10 s	10 s
B_t	6 MB/s	15 MB/s
$B_d(O)$	2 MB/s	2 MB/s

[10]. Hence, the second assumption causes negligible errors in the simulation results. We assume that the disk contention between disk reads (generated by the playback of objects) and disk writes (generated by the upload of objects) is resolved by delaying disk writes [13] as follows. A fragment uploaded from the tape is first stored in the memory buffer and written into its storage disk in an idle period of the disk. This technique smoothes out the bursty data traffic from the disk subsystem, and hence improves request response time. In practice, the additional memory buffer space required by this technique is small, because the aggregate transfer rate of the tape subsystem is much lower than that of the disk subsystem [13]. The storage size of each object is uniformly distributed between 7200 and 14 400 MB. Table 3 shows the major simulation parameters. The results are presented with 95% confidence intervals, where the length of each confidence interval is bounded by 1%.

6.1 Single tape drive

We first study the performance of the algorithms in a system with one robot arm and one tape drive. Here, the request arrival process is Poisson.

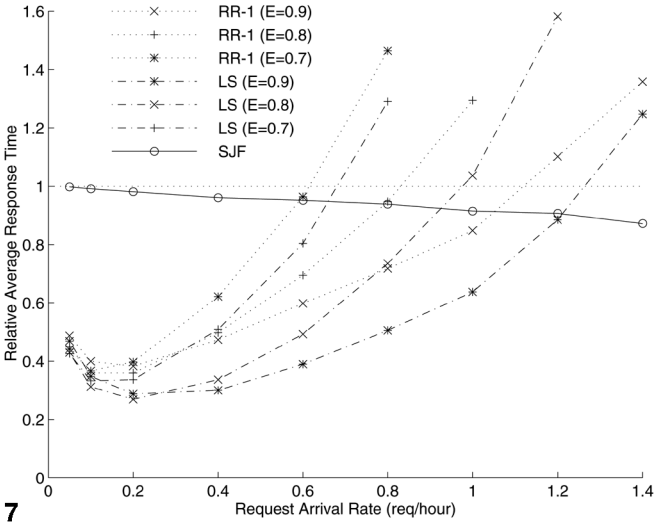
Case 1. Tape drive bandwidth = 6 MB/s.

The maximum throughput of the tape subsystem is 1.95 requests/hour. Table 4 presents the average response time of the FCFS, SJF, RR, and LS algorithms. Blank entries in the table show that the tape subsystem has reached the maximum utilization and the system cannot sustain the input requests. The efficiency of RR and LS algorithms is defined as the percentage of time spent in data transfer. An efficiency of 90% means that 10% of time is spent in tape switches. We define the relative response time to be the ratio of the scheduling algorithm response time divided by the FCFS algorithm response time. The relative response times of the SJF, RR, and LS algorithms are shown in Fig. 7.

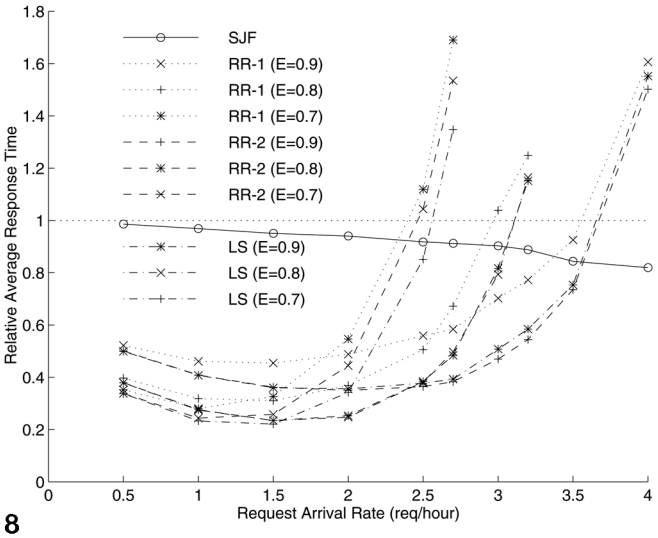
Case 2. Tape drive bandwidth = 15 MB/s.

The maximum throughput of the tape subsystem is 4.72 requests/h. Here, we consider a tape subsystem with a higher performance tape drive. The average response time of the FCFS, SJF, RR, and LS algorithms are shown in Table 5. Again, those blank entries in the table represent a case where the tape subsystem has reached the maximum utilization and the system cannot sustain the input requests. The relative response time of RR and LS algorithms is shown in Fig. 8.

[10] The probability is in the order of 0.001 for the parameters of the simulation



7



8

Fig. 7. The relative response time of SJF, RR, and LS scheduling algorithms

Fig. 8. Relative response time of SJF, RR-1, RR-2, and LS algorithms

In both cases, the LS algorithm has the best performance in a wide range of request arrival rates. The simulation result shows that the time slice algorithm (especially the LS algorithm) performs better than the FCFS algorithm and the SJF algorithm under a wide range of request arrival rates. The SJF algorithm performs better than the FCFS algorithm for all request arrival rates.

6.2 Multiple tape drives

Previous experiments have shown that LS and RR algorithms outperform the FCFS and SJF algorithms in a wide range of load conditions. We study the effect of robot arm contention of the LS algorithm in this experiment.

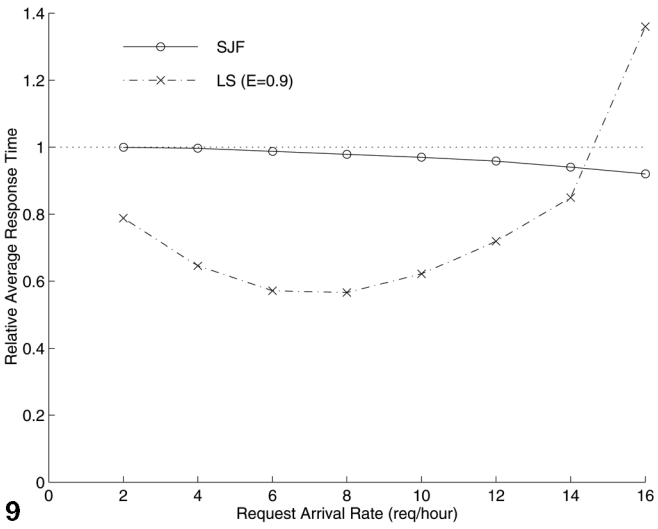
The system contains four tape drives which have a bandwidth of 15.0 MB/s. The maximum throughput of the tape subsystem is 18.90 requests/h. The results are shown in Table 6. A plot of the relative response time vs arrival rate is shown in Fig. 9.

Table 4. Response time vs request arrival rate

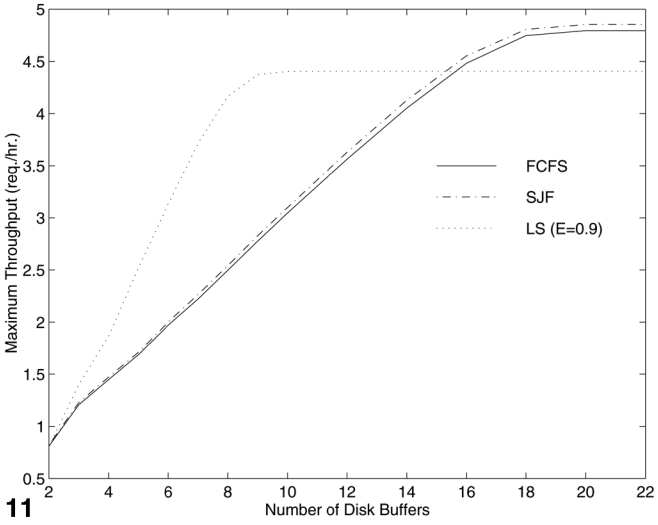
Req. arr. rate (req./h)	FCFS (s)	SJF (s)	RR-1 (E=0.9) (s)	RR-1 (E=0.8) (s)	RR-1 (E=0.7) (s)	LS (E=0.9) (s)	LS (E=0.8) (s)	LS (E=0.7) (s)
0.05	40.23	40.14	19.62	18.06	17.75	18.88	17.28	17.14
0.10	67.43	66.85	26.94	24.29	24.71	23.50	21.04	22.45
0.20	126.02	123.64	48.25	45.36	50.07	36.25	33.90	42.41
0.40	264.74	254.32	125.44	131.82	164.58	79.43	89.10	134.73
0.60	441.67	420.36	264.15	306.79	425.61	172.18	217.71	355.05
0.80	680.54	638.91	488.36	644.61	996.54	344.34	500.21	878.12
1.00	1022.84	936.15	867.36	1323.80	2426.81	652.23	1059.95	2069.68
1.20	1512.06	1371.47	1666.01	2797.85	7412.91	1338.12	2391.70	6950.47
1.40	2397.93	2089.88	3257.67	7922.88		2991.30	6867.68	
1.60	4376.38	3537.21	8725.85			7767.43		

Table 5. Response time vs request arrival rate

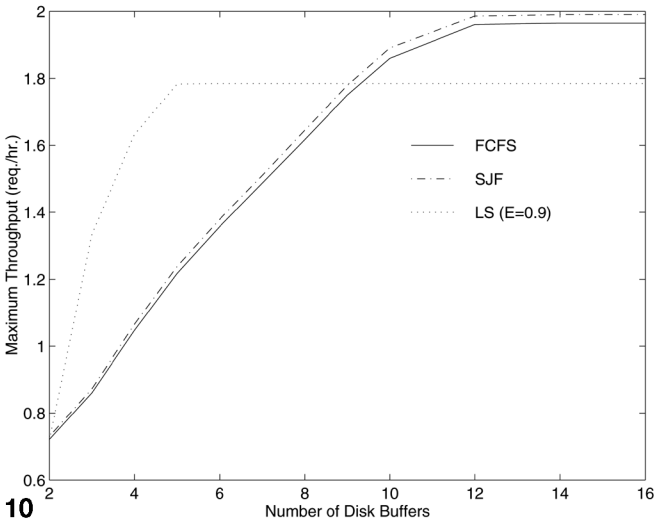
Req. arr. rate (req./h)	FCFS (s)	SJF (s)	RR-1 (E=0.9) (s)	RR-1 (E=0.8) (s)	RR-1 (E=0.7) (s)	RR-2 (E=0.9) (s)	RR-2 (E=0.8) (s)	RR-2 (E=0.7) (s)	LS (E=0.9) (s)	LS (E=0.8) (s)	LS (E=0.7) (s)
0.5	61.55	60.68	32.12	24.43	21.67	30.84	23.38	20.83	30.73	23.31	20.77
1.0	119.66	115.90	55.14	38.15	33.56	48.91	33.07	29.16	48.77	32.84	27.89
1.5	196.54	186.81	89.35	60.98	64.17	71.21	46.10	50.80	70.73	46.10	43.39
2.0	298.70	280.84	145.82	110.13	163.25	104.66	75.62	132.80	106.30	73.92	102.24
2.5	447.42	410.77	250.22	226.21	500.59	162.93	171.68	466.91	169.01	170.92	380.94
3.0	669.852	604.58	470.40	695.71	2294.71	314.68	547.31	2290.34	340.19	531.41	1961.64
3.5	1111.92	937.85	1029.00	2554.95		817.91	2664.21		837.57	2595.05	
4.0	2030.96	1664.11	3262.76			3049.79			3152.80		



9



11



10

Fig. 9. Relative response time of the SJF and LS algorithms

Fig. 10. Maximum throughput of the FCFS, SJF, and LS algorithms

Fig. 11. Maximum throughput of the FCFS, SJF, and LS algorithms

Table 6. Multiple tape drives case: response time vs request arrival rate

Request arrival rate (request/h)	FCFS (s)	SJF (s)	LS (E=0.9) (s)
2.0	19.19	19.18	15.13
4.0	25.11	25.02	16.22
6.0	36.05	35.60	20.60
8.0	55.44	54.25	31.38
10.0	88.17	85.49	54.82
12.0	143.39	137.40	103.21
14.0	241.68	227.33	205.39
16.0	444.21	409.57	605.13

In this simulation experiment, we found that, for the large range of request arrival rates, the utilization of the robot arm is very small. For example, the robot arm utilization is only 0.215 when the request arrival rate is 14.0 requests/h. Hence, the effect of robot arm contention is not a major factor in determining the average response time.

6.3 Throughput under finite disk buffer

In this section, we study the maximum throughput of the FCFS, SJF, and LS algorithms with finite disk buffer space. The maximum throughput of the scheduling algorithm is found by a close-queueing network in which there are 200 clients and each client initiates a new request immediately after its previous request has been served. Hence, there are always 200 requests in the system. The maximum throughput of the LS, FCFS, and SJF algorithms are evaluated for Cases 1 and 2. In each case, the size of each disk buffer is chosen to be large enough to store the data uploaded from a tape drive in one time slice. The efficiency of the LS algorithm is chosen to be 0.9, and therefore, the time slice is 300 s. The disk buffer sizes of Case 1 and 2 are 1.582 GB and 3.955 GB, respectively. The results for Case 1 and Case 2 are shown in Figs. 10 and 11, respectively. From the figures, we observe that the LS algorithm has much higher throughput (in some cases, we have 50% improvement) than the FCFS and SJF algorithms in a wide range of number of disk buffers. The throughput of each algorithm grows with the number of disk buffers, but the LS algorithm reaches its maximum possible throughput with about half of the buffer requirement that the FCFS algorithm needs to achieve its maximum possible throughput. The SJF algorithm performs slightly better than the FCFS algorithm. The FCFS (or SJF) algorithm performs better than the LS algorithm for about 10% when the disk buffer space is large enough.

6.4 Discussion of results

The results show that the LS and RR algorithms outperform the conventional algorithms (FCFS and SJF) in a wide range of request arrival rates. In all the cases, the LS algorithm with 90% efficiency outperforms the FCFS algorithm and the SJF algorithm when the request arrival rate is below 60% of the maximum throughput of the tape subsystem. The conventional algorithms have a better response time when the request arrival rate is quite high (above 70% of the maximum throughput of the conventional algorithms). For the LS or RR algorithm, the algorithm performs better with a lower efficiency factor at low request arrival rate and better with a higher efficiency factor at high request arrival rate. The results also show that the relative response time of the LS and RR algorithms reach a minimum at certain request arrival rates. This is because the response time is the sum of the waiting time W and the tape switch time H. At low request arrival rate, H is the major component of the response time. As the request arrival rate increases from zero, the waiting time of the conventional algorithms grows faster than that of the LS and RR algorithms, because the LS and RR algorithms can serve several requests at the same time, and hence reduce the possibility of waiting for available tape drive. Therefore, the relative response time of the LS and RR algorithms decreases with the increase of request arrival rate when the request arrival is low. When the request arrival rate is high enough, the waiting time of LS and RR algorithms becomes higher than that of the conventional algorithms, because the conventional algorithms have a better utilization of the tape drive bandwidth which covers the high load conditions.

7 Concluding remarks

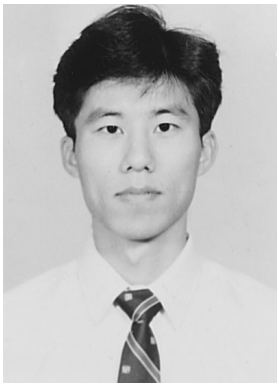
In this paper, we have proposed a cost-effective near-line storage system for a large-scale multimedia storage server using a robotic tape library. We have studied a class of novel time slice scheduling algorithms for the tape subsystem and have shown that under light-to-moderate workload, this class of tape-scheduling algorithm has better response time and requires less disk buffer space than the conventional algorithm. Also, we have complemented our work to the proposed Staggered Striping architecture [4], and showed that, using our proposed scheduling algorithms, how we can organize the data layout on disks and tape cartridges for concurrent upload and display of large multimedia objects.

From the performance results, the selection of the time slice value is often more important than the choice of the time slice algorithm used. If the request arrival process is known in advance (i.e., the average request arrival rate and the inter-arrival time distribution are known), the time slice value can be adjusted by using precomputed results (obtained by either analytical methods or simulations). In practical situations, the request arrival process is usually not known in advance. One simple method that can be used is to adjust the time slice value according to the length of the queue of waiting requests, i.e., a larger time slice value is required if the length of the queue is longer. The function from the queue length to the time slice value can be predetermined by empirical studies. In general, the optimal time slice value depends on the request arrival process, the number of requests waiting for service, and the states of the currently active requests. Further work is required to find the best way to determine the optimal time slice value.

Acknowledgements. This research was supported by the UGC Earmarked Grant.

References

1. The Ampex DST800 Robotic Tape Library Technical Marketing Document (1994)
2. Anderson DP, Osawa Y (1992) A file system for continuous media. ACM Trans Comput Syst 10(4): 311–337
3. Beakley GW (1991) Channel coding for digital HDTV terrestrial broadcasting. IEEE Trans Broadcasting 37(4): 137–140
4. Berson S, Ghandeharizadeh S, Muntz RR, Ju X (1994) Staggered striping in multimedia information systems. In: Proceedings of ACM SIGMOD Conference, pp 79–90
5. Berson S, Golubchik L, Muntz RR (1995) A fault-tolerant design of a multimedia server. In: Proceedings of ACM SIGMOD Conference, pp 364–375
6. Chervenak AL (1994) Tertiary Storage: An evaluation of new applications. Ph.D. dissertation, Computer Science Department, University of California at Berkeley
7. Gemmell DJ, Christodoulakis S (1994): Principles of delay-sensitive multimedia data storage and retrieval. ACM Trans Inf Syst 10(1): 51–90
8. Ghandeharizadeh S, Shahabi C (1994) On multimedia repositories, personal computers, and hierarchical storage systems. In: Proceedings of 2nd ACM Multimedia Conference, pp 407–416
9. Golubchik L, Muntz RR, Watson RW (1994) Analysis of striping techniques in robotic storage libraries. UCLA Technical Report CSD-940014
10. Hodge W et al (1993) Video on demand: architecture, systems, and applications. SMPTE J, pp 791–803
11. Kienzle M et al (1995) Using tertiary storage in video-on-demand servers. In: Proceedings of COMPCON ’95, pp 225–233
12. Lougher P, Shepherd D (1993) The design of a storage server for continuous media. Computer J 36(1): 32–42
13. Lau SW, Lui JCS (1996) Scheduling and replacement policies for a hierarchical multimedia storage server. In: Proceedings of the International Symposium on Multimedia Systems, pp 68–75
14. Mok AK, Dertouzous MLL (1978) Multiprocessor scheduling in a hard real-time environment. In: Proceedings of the 7th Texas Conference on Computing Systems
15. Patterson D, Gibson G, Katz R (1988) A case for redundant arrays of inexpensive disks (RAID). In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp 109–116
16. Rangan PV, Vin HM (1993) Efficient storage techniques for digital continuous multimedia. Trans Knowl Data Eng, pp 564–573
17. Rangan PV, Vin HM (1992) Designing an on-demand multimedia service. IEEE Commun Mag 30(7): 56–65
18. Stallings W (1995) Operating Systems. Prentice-Hall, Englewood Cliffs, N.J.
19. Vin HM, Rangan PV (1993) Designing a multi-user HDTV storage server. IEEE J Select Areas Commun 11: 153–164



SIU-WAH LAU received his B.Sc. and M.Phil. in Computer Science from The University of Hong Kong in 1988 and 1991 respectively. He received his M.Sc. in Computer Science from UCLA in 1993. Currently, he is a PhD candidate in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include distributed multimedia systems and computer networks.

Dr. JOHN CHI-SHING LUI received his Ph.D in Computer Science from UCLA in 1991. He then joined a team in the IBM Almaden/San Jose and participated in a research and development of a parallel I/O architecture project. He also participated in the parallel database project in IBM Yorktown Research Laboratory. In the summer of 1993, he joined the Department of Computer Science and Engineering in the Chinese University of Hong Kong. His current research interests are distributed multimedia systems, distributed mobile computing systems, parallel and distributed database systems, communication networks and performance evaluation theory.