Algorithms for Designing Multimedia Servers *

Harrick M. Vin, **Alok Goyal**, and **Pawan Goyal**

Department of Computer Sciences, University of Texas at Austin
Taylor Hall 2.124, Austin, Texas 78712-1188
E-mail: {vin,alok,pawang}@cs.utexas.edu, Telephone: (512) 471-9732, Fax: (512) 471-8885

ABSTRACT

In this paper, we propose a novel adaptive admission control algorithm, in which a client is admitted for service by a multimedia server only if the extrapolation from the past measurements of the storage server performance characteristics indicate that the service requirements of all the clients can be met satisfactorily. Each client may request the retrieval of a variable bit rate (VBR) encoded media stream and may require a different quality of service. We also present a disk scheduling algorithm that minimizes both seek time and rotational latency incurred while accessing a sequence of media blocks from disk. We examine the effects of a finite read-ahead on the quality of service provided to each admitted client. The effectiveness of the admission control and disk scheduling algorithms is demonstrated through extensive simulations.

Keywords: Multimedia servers, admission control algorithm, disk scheduling algorithm, quality of service

1 Introduction

1.1 Motivation

Recent advances in computing and communication technologies have made it economically viable to provide on-line access to a variety of information sources (such as reference books, journals, newspapers, images, video clips, scientific data, etc.) over high speed networks [5, 6, 16]. In its simplest configuration, the architecture of such systems will comprise of multimedia information servers (which we will henceforth refer to as *multimedia servers*) connected to client sites via high speed networks (e.g., ATM) [12]. Multimedia servers will digitally store information such as proceedings of tele-conferences, educational documentaries, entertainment movies, advertisements, etc., on a large array of extremely high-capacity storage devices (e.g., optical or magnetic disks). On receiving a playback request from a client, a multimedia server will service the client by retrieving the chosen media segments from disk, and then transmitting them to the client's site.

The fundamental problem in developing such multimedia servers is that video, audio, and other similar forms of data differ from numeric data and text in their characteristics, and hence, require totally different techniques for their organization and management. The most critical of these characteristics is the *continuity requirement*. Since digital audio and video streams consist of a sequence of media quanta such as audio samples or video frames, which convey meaning only when presented continuously in time (unlike text in which spatial continuity is sufficient), a multimedia server must ensure that recording and retrieval of media streams to and from disks proceed at their real-time rates. Whereas designing a dedicated, single-client multimedia server does not offer very many design choices and is relatively straightforward, the design of a multimedia sever that is capable of servicing multiple clients simultaneously poses interesting research challenges. The techniques that enable a multimedia server to service a large number of clients simultaneously is the subject matter of this paper.

1.2 Relation to Previous Work

Digitization of audio yields a sequence of samples, and that of video yields a sequence of frames. We refer to a continuously recorded sequence of audio samples or video frames as a *strand*. A multimedia server can organize the storage of media strands on disk in terms of fixed size *media blocks* [13]. Due to the periodic nature of media playback, a multimedia server can service multiple clients simultaneously by proceeding in *rounds*, retrieving media blocks for each client during each round. The number of media blocks of a strand retrieved during a round is dependent on its playback rate requirement, as well as the buffer space availability at the client [19]. Ensuring continuous playback for each strand requires that the *service time* (i.e., the total time spent in retrieving media blocks during a round) does not exceed the minimum of the playback durations of the blocks retrieved for each strand during a round. Hence, before admitting a new

*This research was supported in part by the National Science Foundation (Research Initiation Award CCR-9409666), NASA, Mitsubishi Electric Research Laboratories (MERL), Sun Microsystems Inc., and the University of Texas at Austin.
A preliminary version of this paper appeared in the Proceedings of the First IEEE International Conference on Multimedia Computing and Systems (ICMCS'94), Boston, Pages 234-243, May 1994.

client, a multimedia server must employ admission control algorithms to decide whether a new client can be admitted without violating the continuous playback requirements of the clients already being serviced.

The formulation of the admission control algorithm is dependent on the quality of service requirements of the clients. If the entire clientele of a multimedia server desires deterministic service guarantees (i.e., their continuous playback requirements should never be violated for the entire service duration), the corresponding admission control algorithm will be characterized by worst-case assumptions regarding service time [1, 7, 8, 14, 18, 19, 21]. Notice, however, that due to the human perceptual tolerances as well as the inherent redundancy in continuous media streams, most clients of a multimedia server are tolerant to brief distortions in playback continuity as well as occasional loss of media information. Therefore, providing deterministic service guarantees to all the clients is superfluous. Furthermore, the worst-case assumptions that characterize deterministic admission control algorithms needlessly constrain the number of clients that can be serviced simultaneously, and lead to severe underutilization of server resources. Hence, in order to improve the utilization of server resources, a multimedia server must employ an admission control algorithm which exploits the variation in the access times of media blocks from disk, and provides varying quality of service to the clients.

1.3 Research Contributions of This Paper

In this paper, we propose an *adaptive admission control algorithm*, in which a client is admitted for service only if the extrapolation from the past measurements of the storage server performance characteristics indicate that the service requirements of all the clients can be met satisfactorily. A multimedia server that employs such an algorithm is referred to as providing *predictive* service guarantees to clients [3, 10]. Such an algorithm offers fairly reliable service but no absolute guarantees. We demonstrate that a read-ahead of a finite number of frames of a strand enhances the reliability of the algorithm as well as improves the quality of service provided to each client.

Most of the conventional disk scheduling algorithms (such as, SCAN, Shortest Seek Time First (SSTF), etc.) have addressed the problem of optimizing the total seek time incurred while accessing a sequence of blocks from disk. The fundamental limitation of these algorithms, however, is that they optimize only the seek time, and completely ignore the rotational latency. To address this limitation, Seltzer et al. [15] have recently proposed a Shortest Access Time First (SATF) algorithm, in which data blocks are accessed from disk in a order that simultaneously minimizes both seek time and rotational latency. In this paper, we propose a near optimal algorithm for deriving such a retrieval sequence. We compare the performance of our algorithm with the SATF algorithm.

Finally, since the adaptive admission control algorithm admits clients based on observed performance characteristics of the server, rather than theoretical worst-case bounds, simultaneous servicing of multiple clients may lead to occasional violation of the continuity requirements of some of the clients. In order to enable a multimedia server to meet the requirements of clients as closely as possible, we propose a technique for minimizing as well as distributing such violations among multiple clients.

We have carried out extensive simulations to measure the effectiveness of the adaptive admission control and disk scheduling algorithms. We have also measured the effects of read-ahead and policies for enforcing predictive service requirements of clients on the performance of multimedia servers. We present and analyze our simulation results.

The rest of the paper is organized as follows: Techniques for servicing multiple clients simultaneously are presented in Section 2. In Section 3, we present the adaptive admission control algorithm. The disk scheduling algorithm and the technique for minimizing and distributing violations in playback continuity are outlined in Sections 4 and 5, respectively. Our simulation results are described in Section 6, and finally, Section 7 summarizes our results.

2 Servicing Multiple Clients

Consider a multimedia server that is servicing n clients, each retrieving a video strand (say $S_1, S_2, ..., S_n$, respectively). Let n_d and n_p denote the number of clients that require deterministic and predictive service guarantees, respectively (i.e., $n = n_d + n_p$). Without loss of any generality, let us assume that clients retrieving strands $S_1, S_2, ..., S_{n_d}$ require deterministic service guarantees, and those retrieving $S_{n_d+1}, ..., S_n$ are tolerant to brief violations in playback continuity. Let the total number of frames contained in strand S_i be denoted by $\mathcal{F}_i$. Furthermore, let the service requirement of client i be characterized by α_i, which denotes the *minimum* fraction of the frames of strand S_i that must be retrieved from the server in time for their continuous playback. Clearly, since the clients retrieving strands $S_1, S_2, ..., S_{n_d}$ require deterministic service guarantees, we get: $\forall i : i \in [1, n_d] : \alpha_i = 1$. On the other hand, for all the tolerant clients, $0 < \alpha_i < 1$.

Let us assume that the server is servicing all the n clients simultaneously by proceeding in terms of periodic rounds, retrieving a fixed number of frames of each strand during each round. Let $\mathcal{R}_{pl}^1, \mathcal{R}_{pl}^2, ..., \mathcal{R}_{pl}^n$ denote the playback rates (expressed in frames/sec) of strands $S_1, S_2, ..., S_n$, respectively. Then, the number of frames of strands $S_1, S_2, ..., S_n$ to be retrieved during each round (denoted by $f_1, f_2, ..., f_n$, respectively) can be derived such that:

$$\frac{f_1}{\mathcal{R}_{pl}^1} = \frac{f_2}{\mathcal{R}_{pl}^2} = \cdots = \frac{f_n}{\mathcal{R}_{pl}^n}$$

We refer to $\frac{f_i}{\mathcal{R}_{pl}^i}$ as the duration of a round, and is denoted by $\mathcal{R}$.

The time spent in retrieving these frames from disk (referred to as service time τ) is dependent on the number of media blocks being accessed during a round as well as their relative placement on disk. Since each media strand may be encoded using a variable bit rate compression technique (e.g., JPEG, MPEG, etc.), the number of media blocks that contain f_i frames of strand S_i may vary from one round to another. This variation, when coupled with different relative placements of blocks, yields different service times across rounds. We refer to rounds in which $\tau < \mathcal{R}$ and $\tau > \mathcal{R}$ as *underflow* and *overflow* rounds, respectively. For rounds in which $\tau < \mathcal{R}$, the server may continue to retrieve additional blocks for clients as long as the service time does not exceed $\mathcal{R}$. On the other hand, for rounds in which $\tau > \mathcal{R}$, maintaining playback continuity for intolerant clients (i.e., ones requiring deterministic service) will necessitate the server to delay the retrieval of some of the frames of tolerant clients until the next round.

Notice that, delaying the retrieval of a frame from disk by the duration of a round may or may not result in a playback discontinuity depending on the read-ahead $\mathcal{A}_i$ (defined as the number of frames accessed from disk prior to playback initiation) for strand S_i. Whereas delaying the retrieval of a frame during overflow rounds depletes the read-ahead, accessing additional frames during underflow rounds augments it. Hence, conservatively, if $\mathcal{A}_i$ exceeds the number of frames of strands S_i whose retrieval is delayed, client i will not experience any playback discontinuity. Thus, the desired service requirements of all the tolerant clients can be guaranteed to be met if the server ensures timely retrieval of at least $\widehat{\alpha}_i * \mathcal{F}_i$ frames, where:

$$\widehat{\alpha}_i = \frac{\alpha_i * \mathcal{F}_i - \mathcal{A}_i}{\mathcal{F}_i} \tag{1}$$

Clearly, the value of $\widehat{\alpha}_i$ computed in Equation (1) is conservative. In what follows, we present an adaptive admission control algorithm, in which a client is admitted for service only if the measured characteristics of the current load on the server indicate that, for each admitted client i, at least $\widehat{\alpha}_i * \mathcal{F}_i$ frames of strand S_i will be retrieved on time over the entire service duration.

3 Formulating the Admission Control Problem

An *adaptive admission control algorithm* admits clients based on observed performance characteristics of the server, rather than theoretical worst-case bounds. Although such an algorithm may, in practice, provide fairly reliable service to tolerant clients, its inability to provide absolute guarantees may render it inadequate for intolerant clients. Hence, in the following sections, we separately formulate the admission control criteria for tolerant and intolerant clients, and then present an integrated admission control algorithm for multimedia servers.

3.1 Providing Predictive Service Guarantees

Since the retrieval of a few frames may be delayed during overflow rounds, ensuring that the service requirements of the tolerant clients will not be violated requires the server to limit the occurrence of overflow rounds. To do so, the server must first estimate the number of media blocks to be accessed as well as the corresponding service time for each round:

- **Estimating the number of blocks accessed during a round**: To estimate the number of blocks of strand S_i that may have to be retrieved during a round, the server can utilize the values of the mean (μ_i) and the standard deviation (δ_i) (computed at the time of its storage) of the number of media blocks that contain f_i frames. Specifically, the number of blocks, $\widehat{k}_i$, of strand S_i to be retrieved during a round can be estimated as:

  $$\widehat{k}_i = \mu_i + \epsilon_1 * \delta_i$$

  where ϵ_1 is an empirically derived constant.

- **Service time estimation**: If $\mathcal{K}$ denotes the aggregate number of media blocks accessed during a round, the average amount of time spent in retrieving each media block (which includes seek time, rotational latency, and transfer time) during the round can be computed as:

  $$\eta = \frac{\tau}{\mathcal{K}}$$

  The adaptive servicing policy is based on the assumption that the average amount of time spent for the retrieval of each media block (i.e., the value of η) does not change significantly even after a new client is admitted by the server. In fact, to enable the multimedia server to accurately estimate the amount of time spent in retrieving media blocks during a future round, a history of the values of η observed during the most recent W rounds (referred to as the *averaging window*) can be maintained. Thus, if η_{avg} and σ, respectively, denote the average and the standard deviation of η over W rounds, then the time required to retrieve a block in future rounds ($\widehat{\eta}$) can be estimated as:

  $$\widehat{\eta} = \eta_{avg} + \epsilon_2 * \sigma \tag{2}$$

  where ϵ_2 is an empirically derived constant.

Notice that positive values of ϵ_1 and ϵ_2 enable the estimation process to take into account the second moments, and hence, can be used to make the estimates conservative and limit the occurrence of overflow rounds. Using the values of $\widehat{k}_i$ and $\widehat{\eta}$, thus derived, the admission control criteria can be stated as follows:

Since $\widehat{\alpha}_i$ denotes the percentage of frames of strand S_i that must be retrieved on time so as to meet the requirements of client i, the average number of blocks of strand S_i that must be retrieved by the multimedia server during each round can be approximated by $\widehat{k}_i * \widehat{\alpha}_i$. Consequently, given the empirically estimated average access time of a media block from disk, the requirements of tolerant clients will not be violated if:

$$\widehat{\eta} * \left(\sum_{i=1}^{n} \widehat{k}_i * \widehat{\alpha}_i \right) \leq \mathcal{R} \tag{3}$$

We refer to this as the *adaptive admission control criteria*.

3.2 Providing Deterministic Service Guarantees

For the set of clients that desire deterministic service guarantees (i.e., their continuous playback requirements should never be violated for the entire service duration), the admission control criteria will be characterized by the worst-case assumptions regarding the service time. To illustrate, consider a multimedia server that employs a SCAN disk scheduling algorithm [4, 17]. Let $\forall i \in [1, n] : k_i^{max}$ denote the maximum number of media blocks that may contain f_i frames of strand S_i. Hence, the server may be required to retrieve $k^{max} = (k_1^{max} + k_2^{max} + ... + k_n^{max})$ media blocks from disk during a round. Since, in the worst-case, each media block may be placed on a different track, the disk head may have to be repositioned onto a new track at most k^{max} times. Furthermore, while accessing these blocks, the disk head may have to move from the inner-most track to the outer-most track, or vice versa. Hence, assuming that the disk contains T tracks and the seek time incurred while moving the disk head from track t_1 to t_2 is given by $l_{seek}(t_1, t_2) = a + b * |t_1 - t_2|$ where a and b are constants, the upper bound on the total seek time incurred during each round can be computed as: $(a * k^{max} + b * T)$. Moreover, once the disk head is positioned on the track containing a media block, the total rotational latency and transfer time incurred while accessing the block is bounded by the maximum rotational latency (denoted by l_{rot}^{max}). Hence, the total service time for each round can be computed as:

$$\tau = b * T + (a + l_{rot}^{max}) * k^{max} \tag{4}$$

Consequently, to ensure that the continuous playback requirements of all the clients are *strictly* met for the entire service duration, the server must verify that:

$$b * T + (a + l_{rot}^{max}) * \sum_{i=1}^{n} k_i^{max} \leq \mathcal{R} \tag{5}$$

We refer to this as the *deterministic admission control criteria*.

3.3 Admission Control Algorithm

Consider a multimedia server that is servicing $n = n_d + n_p$ clients, where n_d and n_p denote the number of clients that require deterministic and predictive service guarantees, respectively. Now, consider a scenario that the server receives a new client request for the retrieval of strand S_{n+1}. Let f_{n+1} denote the number of frames of strand S_{n+1} that need to be retrieved during each round. Furthermore, let μ_{n+1}, δ_{n+1}, and k_{n+1}^{max}, respectively, denote the mean, the standard deviation, and the maximum number of media blocks that may contain f_{n+1} frames of strand S_{n+1}.

If the new client desires deterministic service guarantees, then before admitting the client, the multimedia server must ensure that neither the deterministic nor the predictive services being provided to the existing clients will be violated after the new client is admitted:

1. To verify that the deterministic service guarantees provided to n_d clients as well as the requirements imposed by the $(n+1)^{th}$ client will not be violated, the multimedia server must ensure that:

$$b * T + (a + l_{rot}^{max}) * \left(k_{n+1}^{max} + \sum_{i=1}^{n_d} k_i^{max} \right) \leq \mathcal{R} \tag{6}$$

2. To verify that the predictive service being provided to the $n_p = n - n_d$ clients will not be violated after the admission of the $(n+1)^{th}$ client, the server must ensure that the service requirements of all the $(n+1)$ clients can be met satisfactorily. Specifically, if $\widehat{k}_{n+1} = \mu_{n+1} + \epsilon_1 * \delta_{n+1}$ denotes the estimated number of media blocks of strand S_{n+1} to be retrieved during a round, then the requirements of the tolerant clients will not be violated if:

$$\widehat{\eta} * \left(\widehat{k}_{n+1} + \sum_{i=1}^{n_d} \widehat{k}_i + \sum_{i=n_d+1}^{n} \widehat{k}_i * \widehat{\alpha}_i \right) \leq \mathcal{R} \tag{7}$$

Hence, a multimedia server will admit a new client requiring deterministic service guarantees if and only if Equations (6) and (7) are satisfied.

On the other hand, if the new client requires predictive service, let $\widehat{\alpha}_{n+1}$ (derived using Equation (1)) denote the minimum fraction of the frames of strand S_{n+1} that must be retrieved on time. Since admitting a tolerant client cannot violate the requirements of intolerant clients (because in the case of an overflow, media blocks of only tolerant clients are delayed), the multimedia server can admit the new client if its admission will not violate the predictive service being provided to any client. Specifically, the server must verify that the following condition is satisfied:

$$\widehat{\eta} * \left(\sum_{i=1}^{n_d} \widehat{k}_i + \sum_{i=n_d+1}^{n+1} \widehat{k}_i * \widehat{\alpha}_i \right) \leq \mathcal{R} \tag{8}$$

Notice that the adaptive admission control criteria (utilized in Equations (7) and (8)) is a heuristic that enables a multimedia server to increase the number of clients that can be serviced simultaneously. The effectiveness of such a criteria, however, is dependent on:

- The values of $\widehat{\eta}$: smaller the value of $\widehat{\eta}$, greater is the number of clients that can be serviced simultaneously by the server. Hence, the multimedia server must employ disk scheduling algorithms that minimize the total time spent in retrieving media blocks from disk (Section 4).

- The ability of the algorithm to meet the requirements of clients as closely as possible. This requires that the multimedia server employ policies for judiciously selecting the additional frames to be retrieved during underflow rounds, as well as the frames to be delayed during overflow rounds (Section 5).

4 Techniques for efficient retrieval of media blocks

Consider a multimedia server that is expected to retrieve N blocks during each round. Let the position of each media block on disk be denoted as $z_i = \langle x_i, y_i \rangle$, where x_i and y_i denote the track number and the block number on that track, respectively. The goal of the disk scheduling algorithm is to find the optimal access sequence for blocks $z_1, z_2, ..., z_N$, assuming that the head is initially positioned at location $z_0 = \langle x_0, y_0 \rangle$. Formally, if $\psi(z_i, z_j)$ denotes the cost function characterizing the overhead in positioning the disk head at location z_j starting from location z_i, then a sequence for retrieving N media blocks can be considered optimal if it minimizes the sum:

$$\Psi = \psi(z_0, \omega_1) + \psi(\omega_1, \omega_2) + \cdots + \psi(\omega_{N-1}, \omega_N)$$

where $\forall i \in [1, N] : \omega_i \in \{z_1, z_2, ..., z_N\}$, and $\forall j \in [1, N], j \neq i : \omega_i \neq \omega_j$. The sequence $\omega_1, \omega_2, ..., \omega_N$ denotes the *retrieval sequence*.

Most of the conventional disk scheduling algorithms (such as, SCAN, Shortest Seek Time First (SSTF), etc.) have addressed the problem of optimizing the total seek time incurred while accessing a sequence of blocks from disk [4, 9, 17, 20]. That is, the cost function is defined as $\psi(z_i, z_j) = l_{seek}(z_i, z_j) = a + b * |x_i - x_j|$. The fundamental limitation of these disk scheduling algorithms is that they optimize only the seek time, and completely ignore the rotational latency. In most of the state-of-the-art disks, however, the values of maximum seek and rotational latencies are comparable (typically, the maximum rotational latency is about half of the maximum seek time). Consequently, disk scheduling algorithms which optimize only the seek time (e.g., SCAN, SSTF, etc.) may incur significant rotational latencies during the retrieval of a sequence of blocks from disk.

To address this limitation, Seltzer et al. [15] have recently proposed a Shortest Access Time First (SATF) disk scheduling algorithm that derives a sequence for accessing media blocks from disk by simultaneously minimizing both seek and rotational latency incurred during retrieval. Specifically, using this algorithm, a retrieval sequence is derived as follows: Given the location of the disk head as well as the locations of all the blocks to be retrieved, first access the block that will incur minimum seek time and rotational latency, and then repeat the procedure. Since this heuristic derives a retrieval sequence by chaining together "locally" optimal choices, the resulting retrieval sequence may not yield globally optimal service times.

In this paper, we propose a near optimal algorithm for deriving retrieval sequences. In what follows, we first formulate the problem of finding an optimal sequence for retrieving media blocks from disk as a graph theory problem, and then present a simple heuristic which is extremely easy to implement, and yet achieves significant performance improvements over the conventional SCAN and the SSTF disk scheduling policies.

4.1 An efficient disk scheduling algorithm

Consider a fully connected directed graph $G = (V, E)$, where $V = \{z_0, z_1, z_2, ..., z_N\}$. Let each edge $(z_i, z_j), i \neq j$ in G be labeled with weight $\psi(z_i, z_j)$. Notice that since the rotational latency incurred while moving the disk head from node z_i to z_j, in general, is not equal to that incurred while moving the disk head from z_j to z_i, the graph G contains both the edges (z_i, z_j) and (z_j, z_i), each with a different weight.

Having constructed this graph, the problem of minimizing the total cost of retrieving media blocks during a round starting from node z_0 can be reduced to the *traveling salesman problem*, a classical graph theory problem known to be NP-complete [20]. Hence, instead of deriving an optimal sequence for accessing media blocks, we present a *greedy* algorithm which, in practice, derives near-optimal retrieval sequences.

Given the graph $G = (V, E)$, the goal of the greedy algorithm is to construct a minimum cost sub-graph $G' = (V, E')$, such that edges in E' constitute a simple path of length N starting from z_0. In order to do so, the greedy heuristic evaluates and selects edges in the increasing order of their weight. An edge $(z_i, z_j), j \neq 0$ is included in E' if it does not create a cycle amongst already selected edges, and if out-degree(z_i) $= 0$ and in-degree(z_j) $= 0$ (i.e., ensure that at most one edge is incident upon and emanating from each node). Notice that since the edges are chosen in the increasing order of weight, the heuristic may initially create a disconnected set of *fragments* (i.e., a chain of edges), which are eventually connected to form a path that connects all the nodes in the graph. Consequently, this heuristic is also some times referred to as the *multiple fragment* heuristic [2]. The algorithm terminates when $|E'| = N$. The complete algorithm for deriving graph G' is described in Figure 1.

As is evident from Figure 1, the greedy algorithm first sorts all the edges in the increasing order of weight, and then constructs G' by sequentially considering edges from the sorted array. In order to ensure that at most one edge is incident upon and emanating from each node in G', the algorithm maintains the values of in-degree and out-degree for each

```
sort all edges in increasing order of cost;
edgesPicked = 0;
G' = NIL;
While (edgesPicked < N) do
{
        select the next edge (z_i, z_j), j ≠ 0;
        if (in-degree(z_j) = 0 ∧ out-degree(z_i) = 0
           ∧ tail(z_i) ≠ z_j) then
        {
                G' = G' + (z_i, z_j);
                edgesPicked = edgesPicked +1;
                in-degree(z_j) = 1;
                out-degree(z_i) = 1;
                MaintainTail();
        }
}
```

Figure 1 : Greedy disk scheduling algorithm

node. Furthermore, cycles are detected by maintaining the *tail* array (through the MaintainTail() procedure) to identify the first node in each fragment. A straightforward analysis of this algorithm will reveal that the overall complexity of the algorithm is $O(N^2 logN)$. It can also be shown that, if the range of values for edge weights is countably finite, the complexity of the above algorithm can be reduced to $O(N^2)$ by employing bucket sort for ordering the edges in the increasing order of weight. In comparison, the complexity of the SATF algorithm is $O(N^2)$.

Observe that the greedy algorithm is an off-line algorithm. Hence, it requires that the set of blocks to be retrieved from disk be known a priori. The sequential nature of audio and video playback enables a multimedia server to predict the set of blocks that need to be accessed during successive rounds. Consequently, during each round, a multimedia server can retrieve media blocks from disk while concurrently computing the retrieval sequence for the next round, and so on. Hence, employing this algorithm is not only feasible, but also efficient.

5 Enforcing Predictive Service Guarantees

Since the sequential nature of media playback enables a multimedia server to precompute the retrieval sequence for round i while accessing media blocks during round $(i-1)$, an overflow or an underflow in a round can be detected prior to its initiation. For underflow rounds, the server may retrieve additional blocks for clients as long as the service time does not exceed $\mathcal{R}$. For overflow rounds, maintaining the playback continuity for intolerant clients will necessitate the server to delay the retrieval of some of the frames of tolerant clients to subsequent rounds. Consequently, to meet the service requirements of clients as closely as possible, the server must: (1) maximize the number of additional frames retrieved during each underflow round, (2) minimize the number of frames whose retrieval is delayed during each overflow round, and (3) equitably distribute the additional frames retrieved during underflow rounds as well as the frames delayed during overflow rounds among all the clients.

5.1 Scheduling Frame Retrievals During Underflow Rounds

During an underflow round, a multimedia server can schedule the retrieval of additional media blocks for each client as long as the service time does not exceeds $\mathcal{R}$. Clearly, the first block to be accessed for each client during the next round is an eligible candidate for retrieval during an underflow round. However, the selection of a subset of these blocks may be governed by the increase in service time resulting from their retrieval, and the service requirements of the clients.

Formally, let b_i denote the block of strand S_i eligible for retrieval during an underflow round j. Clearly, if $c(b_i)$ denotes the increase in the service time yielded by scheduling the retrieval of block b_i in round j, the server must schedule the retrieval of those media blocks during round j which maximize the value of $\frac{1}{c(b_i)}$. Similarly, the server must distribute the number of additional blocks being accessed during the underflow round among all the tolerant clients proportional to their service requirements. Hence, a server may define an objective function $\mathcal{I}_{i,j}$ as:

$$\mathcal{I}_{i,j} = \alpha_i * \left(\frac{1}{c(b_i)}\right) \tag{9}$$

and then schedule the retrieval of additional blocks in the decreasing order of $\mathcal{I}_{i,j}$ as long as the resulting service time does not exceed $\mathcal{R}$. Since α_i denotes a client parameter specified at the time of their admission, evaluating the objective function, for each eligible block b_i, will require the server to compute the value of $c(b_i)$.

Since the addition of block b_i to the set of blocks to be retrieved in round j may alter the retrieval sequence, determination of $c(b_i)$, ideally, requires the server to derive a new retrieval sequence, and then compute the difference between the new and the old service times. If, however, the server is employing the greedy disk scheduling algorithm (presented in Section 4), the computational complexity of deriving the retrieval sequence may render the above approach for computing $c(b_i)$ for each block b_i infeasible. To avert the recomputation of the retrieval sequence, the server may approximate the overhead of retrieving block b_i during round j by scheduling its retrieval between two blocks in the original sequence, and then computing the new service time. Formally, if $\omega_0, \omega_1, ..., \omega_N$ denotes the original retrieval sequence, then the server can first compute the overhead (i.e., the increase in service time) yielded by inserting block b_i

between each pair of (ω_{j-1}, ω_j), and then compute $c(b_i)$ as:

$$c(b_i) = \min_{j \in [1,N]} \left(\psi(\omega_{j-1}, b_i) + \psi(b_i, \omega_j)\right) \qquad (10)$$

where $\psi(z_i, z_j)$ denotes the overhead of positioning the disk head at location z_j starting from location z_i.

5.2 Scheduling Frame Retrievals During Overflow Rounds

During an overflow round, maintaining the playback continuity for each of the intolerant clients, will necessitate the server to delay the retrieval of some of the frames of tolerant clients until a subsequent round. Notice, however, that delaying the retrieval of a frame of strand S_i may or may not yield a discontinuity in its playback. This is because, occasional delay in retrieving a frame from disk can be absorbed by the frames buffered at the client. In fact, a discontinuity in playback will be observed only if the retrieval of a frame of strand S_i is delayed when all the frames in the client buffer have already been consumed. Formally, if the retrieval of $\bar{f}_i, \bar{f}_i \leq f_i$ frames of strand S_i have been delayed to a subsequent round as a result of an overflow in round j, and if $\mathcal{A}_{i,j-1}$ denotes the number of frames buffered at client i at the end of round $(j-1)$, then a playback discontinuity will be observed during round j if $\bar{f}_i > \mathcal{A}_{i,j-1}$. In fact, the number of frames of client i whose playback has beed delayed during the first j rounds of service (denoted by $d_{i,j}$) can be computed as:

$$d_{i,j} = d_{i,j-1} + \max\left(0, \bar{f}_i - \mathcal{A}_{i,j-1}\right) \qquad (11)$$

where $d_{i,0} = 0$ and $\mathcal{A}_{i,0} = \mathcal{A}_i$.

Thus, given that α_i denotes the minimum fraction of the frames of strand S_i that must be retrieved from the server in time, if $\mathcal{A}_i$ frames of strand S_i are read-ahead prior to initiating its playback, then the maximum number of frames whose playback can be delayed without violating the service requirements of client i is bounded by:

$$\mathcal{D}_i = \lfloor (1 - \alpha_i) * \mathcal{F}_i + \mathcal{A}_i \rfloor \qquad (12)$$

In the simplest case, to reduce the service time during an overflow round j, the server may first delay the retrieval of frames of the client with the highest value of $\mathcal{D}_i$, until no more frames of that client can be delayed during round j. If the resulting round continues to yield an overflow, then the retrieval of frames of the client with the second largest value of $\mathcal{D}_i$ can be delayed, and so on. Notice that, such a simplistic policy may cluster the violations in playback continuity for clients over relatively short periods of time, and hence, may make them highly perceptible.

In order to achieve an equitable distribution of playback continuity violations throughout the playback duration of a strand, we define *delay affordability* of client i (measured in terms of number of frames) at the end of round j as:

$$\chi_{i,j} = \mathcal{D}_{i,j} - d_{i,j} \qquad (13)$$

where $\mathcal{D}_{i,j} = \lfloor j * (1 - \alpha_i) * f_i \rfloor + \mathcal{A}_{i,j}$. The server can then select a set of frames whose retrieval can be delayed to subsequent rounds based on the values of $\chi_{i,j}$. Clearly, since $\mathcal{D}_{i,j}$ gradually increases from one round to the next, employing such a policy will enable the server to disperse the frames whose retrieval is delayed throughout the service duration.

Once the delay affordability of each client has been computed, the server must determine a subset of media blocks of the clients with $\chi_{i,j} > 0$ whose retrieval could be delayed to a subsequent round. The algorithm for determining such a subset consists of the following two steps:

- In the first step, based on the delay affordability of each client, media blocks to be retrieved during an overflow round are labeled as either *can-be-delayed* or *can-not-be-delayed*. For each strand S_i, due to the sequential nature of its playback, only the last block (defined with respect to their order of consumption at client i) being retrieved in the round is eligible to be delayed. Given such a block b_i for each strand S_i, it is labeled as *can-be-delayed* only if the number of frames contained in the block is smaller than the delay affordability of client i. All the other blocks being retrieved during the round are labeled as *can-not-be-delayed.*

- Once all the blocks have been labeled, to minimize the number of blocks (and hence, the number of frames) whose retrieval is delayed, the server must select blocks in the decreasing order of the reduction in service time per delayed frame yielded by their exclusion from the retrieval sequence. Specifically, if $\phi(b_i)$ and $f(b_i)$ denote the reduction in service time yielded by excluding media block b_i from the retrieval sequence, and the number of frames contained in block b_i, respectively, then a gain function $g(b_i)$ can be defined as:

  $$g(b_i) = \frac{\phi(b_i)}{f(b_i)}$$

  The server can then delay the retrieval of *can-be-delayed* media blocks in the decreasing order of $g(b_i)$.

Notice, however, that for each block whose retrieval can be delayed, computing the value of $g(b_i)$ may require the server to recompute the retrieval sequence as well as the corresponding service time, after excluding that block from the path selection process (see Section 4). Furthermore, after delaying the retrieval of a block that maximizes the value of $g(b_i)$ to a subsequent round, if the service time continues to exceed $\mathcal{R}$, then the server is required to repeat this entire procedure until the resulting service time is smaller than $\mathcal{R}$. Whereas such an algorithm may be appropriate for conventional disk scheduling algorithms (e.g., SCAN, SSTF, etc.) in which recomputation of retrieval sequence is relatively inexpensive, the high computational complexity of determining the retrieval sequence using the greedy disk scheduling algorithm makes this approach virtually infeasible.

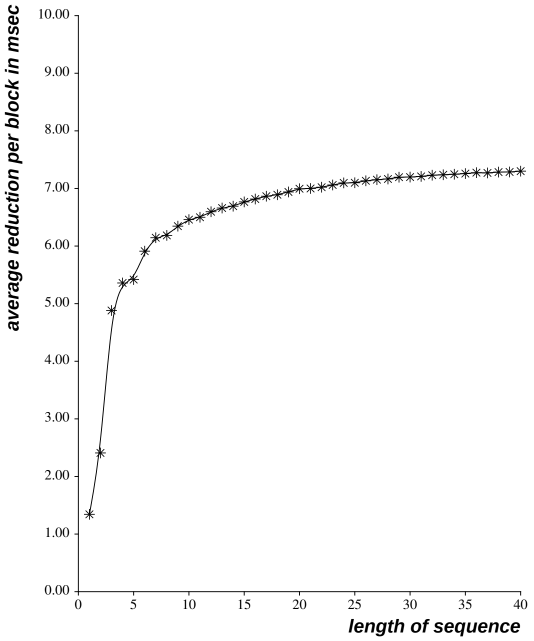

Figure 2 : Variation in the average reduction in service time yielded for each delayed media block with the subsequence length

To avert such recomputations, a multimedia server may just approximate the reduction in service time yielded by delaying the retrieval of a media block located at ω_i as $\psi(\omega_{i-1}, \omega_i) + \psi(\omega_i, \omega_{i+1}) - \psi(\omega_{i-1}, \omega_{i+1})$, where ω_{i-1} and ω_{i+1} denote the predecessor and successor nodes of ω_i in the original retrieval sequence. Conceptually, this is equivalent to replacing edges (ω_{i-1}, ω_i) and (ω_i, ω_{i+1}) from graph G' by edge $(\omega_{i-1}, \omega_{i+1})$ (see Section 4.1). However, our analysis has demonstrated that since the greedy disk scheduling algorithm optimizes both seek time and rotational latency, employing the above heuristic to approximate the reduction in service time yielded by excluding an isolated media block from the retrieval sequence is not very beneficial. In fact, the average reduction in service time yielded for each delayed media block is higher if a sequence of n media blocks, rather than n isolated media blocks, are excluded from the retrieval sequence (see Figure 2). Consequently, rather than delaying isolated media blocks which maximize $g(b_i)$, a server that employs the greedy disk scheduling algorithm may determine maximal length subsequences of *can-be-delayed* media blocks, and then delay the retrieval of subsequences in the decreasing order of $G(s)$, where:

$$G(s) = \frac{\Phi(s)}{\sum_{b \in s} f(b)}$$

where $\Phi(s)$ denotes the reduction in service time yielded by excluding subsequence s from the retrieval sequence.

As a final caveat, if, even after delaying the retrieval of all the *can-be-delayed* blocks (defined using the delay affordability criteria defined in Equation (13)), the service time τ continues to exceed $\mathcal{R}$, the server may repeat the above process by using $\mathcal{D}_i$ instead of $\mathcal{D}_{i,j}$ in Equation (13). A proper choice of values for ϵ_1 and ϵ_2 in the admission control criteria can limit the number of admitted clients, and hence, can virtually ensure that the server will always be successful in determining a subset of *can-be-delayed* blocks whose exclusion will yield $\tau \leq \mathcal{R}$.

6 Experimental Evaluation

In this section, we demonstrate the viability of the adaptive admission control algorithm through trace-driven simulations. The simulations were carried out in an environment consisting of a synchronous disk array with 128 disks, each with a storage capacity of 0.5 GBytes. The storage capacity of the disk is assumed to be partitioned into 1024 tracks of uniform capacity (namely, 0.5 MBytes/cylinder). For the sake of simplicity, the disk is assumed to consist of only one platter[1], and that the capacity of each track is assumed to be partitioned into disk blocks of size 4 KBytes each. For computing rotational latency incurred while accessing a block, we have assumed the disk head to move only radially. The characteristics of each disk are summarized in Table 1.

For the purposes of the simulations, each video strand is assumed to be encoded using a Variable Bit Rate (VBR) compression technique. Furthermore, each media block is assumed to be striped across the entire array, yielding a media block size of 512 KBytes. Due to variable rate compression, the number of frames contained in a media block may vary from one media block to another. Successive blocks of a strand are assumed to be stored on disk using the random placement model [11]. The greedy disk scheduling algorithm (presented in Section 4) was employed for all the simulations. The playback rate of each video strand is assumed to be 30 frames/sec. Furthermore, 30 frames of each video strand are assumed to be retrieved during each round, yielding $\mathcal{R} = 1$ sec. The trace data for frame size variation yielded by VBR encoding techniques was obtained from Bellcore, University of California at Berkely, and Columbia University.

6.1 Evaluation of the greedy disk scheduling algorithm

Recall that the greedy algorithm presented in Section 4 derives a sequence of accessing media blocks from disk so as to simultaneously minimize both seek and rotational latency. In contrast, conventional disk scheduling algorithms (such as, SCAN) optimize only the seek time. Figure 3 shows

[1] In fact, we have assumed a multi-platter disk in which all the platters of the disk spin at a constant rate, each platter is associated with a read/write head that is attached to a common actuator, and that data can be read and written by all the disk heads simultaneously. Conceptually, such a multi-platter disk configuration is logically equivalent to a disk with only one platter such that the capacity of each track on the single-platter disk is equal to the capacity of a cylinder in the multi-platter disk.

Disk capacity	0.5 GBytes
Number of cylinders per disk	1024
Disk block size	4 KBytes
Rate of disk rotation	3600 RPM
$l_{seek}(t_1, t_2)$	$4 + 0.02 * \|t_1 - t_2\|$ ms
Maximum seek time	24.48 ms
Maximum rotational latency	16.66 ms

Table 1 : Disk parameters assumed in the simulation

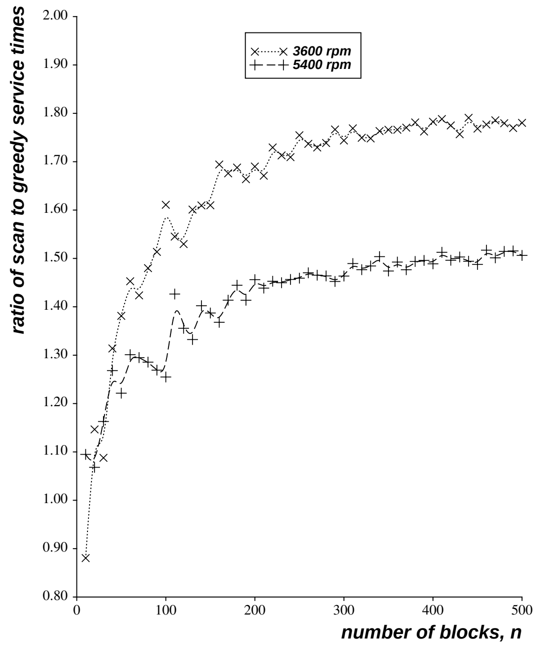

Figure 3 : Comparison of SCAN and greedy disk scheduling algorithms

the variation in the ratio of the service times derived for the SCAN and the greedy algorithms with increase in the number of blocks, n, retrieved during each round. Observe that the performance of the greedy scheduling technique improves with increase in n. This is because, at smaller values of n, the total seek time incurred during their retrieval dominates the performance (and hence, SCAN performs as well as the greedy algorithm). However, as the value of n increases, the cumulative rotational latency starts dominating the total service time, thereby enabling the greedy algorithm to outperform SCAN. Figure 3 also indicates that even at higher values of rotational rate, the gain in performance yielded by the greedy algorithm is significant.

In order to demonstrate the viability of employing the greedy algorithm for determining the retrieval sequence during each round, we compared the time required to derive the sequence with the service time. Whereas the time to compute the retrieval sequence shows a quadratic dependence on the

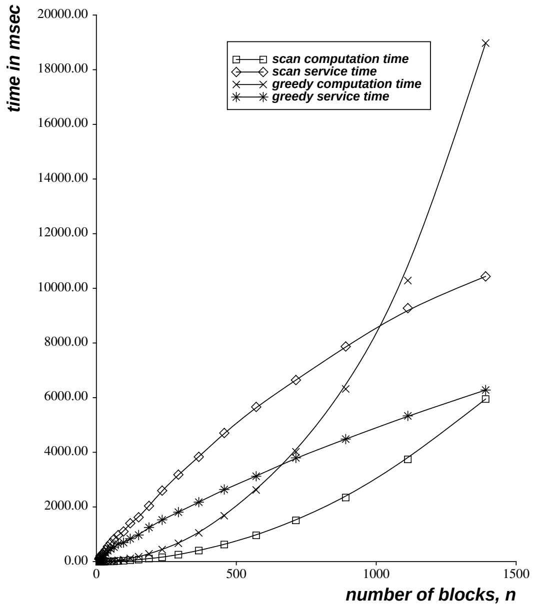

Figure 4 : Evaluation of the greedy algorithm - comparison of time to derive a retrieval sequence with the service time. The time to derive the retrieval sequence was measured on a moderately loaded SPARCstation 10 - Model 30.

number of blocks being retrieved, the time spent in retrieving the blocks increases almost linearly with increase in the number of blocks. Figure 4 illustrates this behavior. It also illustrates that as long as the number of blocks being retrieved in a round is less than 650, the time to derive a retrieval sequence is less than the service time. However, for $\mathcal{R} = 1$, the number of blocks required to be accessed during a round, is in the range $[100, 200]$. Hence, although theoretically possible, the time required to compute a retrieval sequence, in practice, does not exceed the service time. In fact, at $n = 150$, the time to compute the retrieval sequence is only 18% of the total service time.

Finally, we have also compared the performance of the greedy disk scheduling algorithm proposed in this paper with the Shortest Access Time First (SATF) algorithm proposed by Seltzer et al. [15]. Although the computational complexity of both algorithms is $O(N^2)$, where N is the number of blocks being accessed from disk, Figure 5 illustrates that the greedy algorithm yields consistently lower service times than the SATF algorithm.

6.2 Evaluation of the admission control algorithm

In order to enable the multimedia server to accurately predict the access time of a media block from disk, we maintain a history of the values of η observed during the most recent $W = 100$ rounds (an empirically derived value), and compute the values of η_{avg} and σ over W rounds. Figure 6 depicts the variation in η_{avg} with the number of blocks retrieved dur-

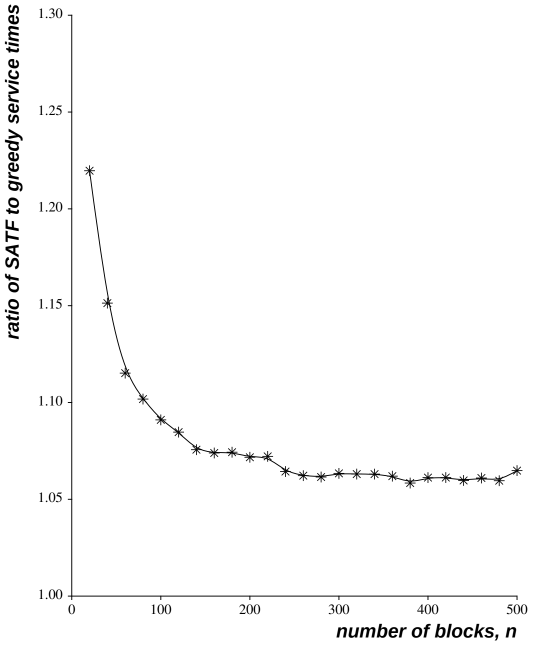

Figure 5 : Comparison of the greedy and the SATF disk scheduling algorithm

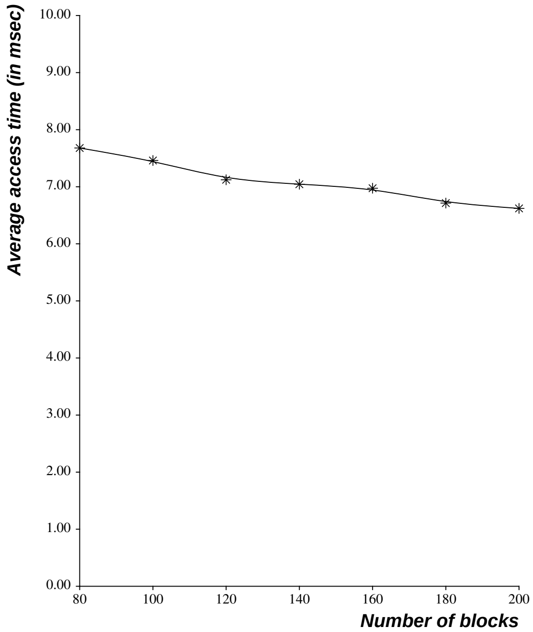

Figure 6 : Variation in the average access time per media block with increase in the number of media blocks

ing a round. It demonstrates that η_{avg} decreases very slightly with increase in the number of blocks accessed during a round (and thereby verifies our assumption that the amount of time spent for the retrieval of each media block does not change significantly even after a new client is admitted by the server). Furthermore, for our simulation environment, the values of δ_i and σ were observed to be significantly smaller than the corresponding values of μ_i and η_{avg}, respectively. Consequently, increasing the values of ϵ_1 and ϵ_2 only marginally affects the number of clients admitted by the adaptive admission control algorithm. Higher values of ϵ_1 and ϵ_2 do, however, yield a more robust admission control criteria. Hence, throughout our experiments, we have used $\epsilon_1 = \epsilon_2 = 0.5$.

The performance of the adaptive admission control algorithm was evaluated for various values of α_i. Figures 7(a) and 7(b) depict the variation in the number of admitted clients and the server utilization with α_i. As is evident from the figure, the number of clients admitted by the server as well as the server utilization increase slowly with decrease in α_i. Figure 7(a) also illustrates that there is no gain in the number of admitted clients when number of read-ahead blocks is increased from 0 to 1. This is because, $\widehat{\alpha}_i$ decreases very slowly with increase in read-ahead (see Equation (1)). The only exception is the scenario in which the entire clientele requires $\alpha_i = 1$. Since the adaptive admission control algorithm offers fairly reliable service but no absolute bounds, it is pertinent only when $\alpha_i < 1$. Hence, without any read-ahead, clients requesting $\alpha_i = 1$ must be admitted using the deterministic admission control criteria. However, even a small read-ahead yields $\widehat{\alpha}_i < 1$, thereby enabling the server to employ the adaptive admission control criteria, and provide *effective* deterministic service to clients.

The effectiveness of the adaptive admission control algorithm is also demonstrated by the fact that for all values of α_i in the range [0.9, 1.0), the utilization of the server is at least 96%. Moreover, the quality of service requested by the clients is empirically satisfied with a probability greater than 0.999. This indicates that employing any other admission control criteria will not yield any significant improvement in the number of clients admitted by a multimedia server.

As we had outlined in Section 2, read-ahead of a finite number of frames of a strand, prior to the initiation of its playback, enables the clients to absorb transient violation in the continuous retrieval of media information from disk. Consequently, a client with a finite read-ahead perceives a better quality of service as compared to a client with no read-ahead. Figure 8 depicts this variation in the effective quality of service provided to clients with increase in read-ahead. Figure 8 also demonstrates that the rate of improvement in the effective quality of service provided to clients decreases with increase in read-ahead. In fact, most of the gain is achieved with a read-ahead of only one media block.

Notice that if the server employs the deterministic admission control criteria (Equation (5)) to admit and service clients with $\alpha_i = 1$, then the maximum number of clients that can be serviced simultaneously by the server is bounded by 16. On the other hand, with a read-ahead of 1 media block, the adaptive admission control criteria can provide an effective 'deterministic' service to 84 clients simultaneously. Thus, in comparison with the deterministic admission control algorithm, the adaptive admission control algorithm increases the number of clients serviced simultaneously by

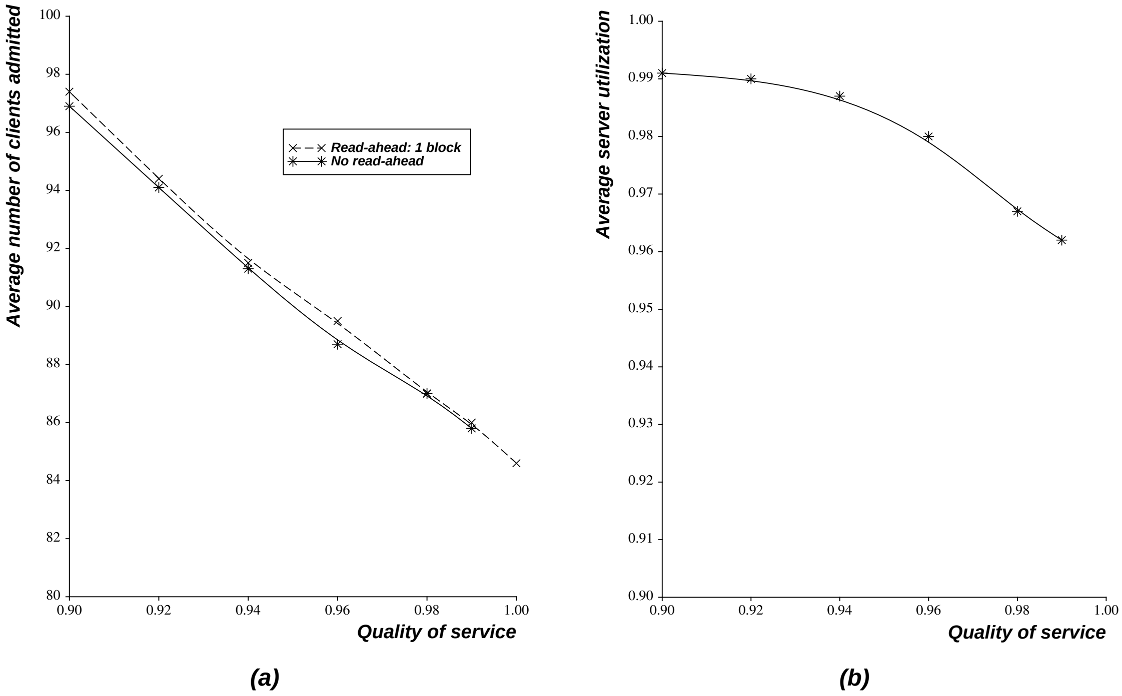

Figure 7 : Variation in the number of clients admitted and server utilization with α

about 400% (200% increase due to the difference in the worst-case and average-case data transfer requirements of clients, and another 200% due to the difference in the worst-case and average-case access times).

7 Conclusion

In this paper, we have presented an adaptive admission control algorithm, in which a client is admitted by a multimedia server if the extrapolation from the past measurements of the storage server performance characteristics indicate that the service requirements of all the clients can be met satisfactorily. The main goal of our admission control algorithm is to accept enough number of clients to efficiently utilize the server resources, while not accepting clients whose admission may lead to the violations of the service requirements of clients. The greedy disk scheduling algorithm as well as the technique for minimizing and distributing the discarded media blocks presented in this paper significantly improve the performance of the admission control algorithm. Furthermore, a read-ahead of a small number of frames significantly improves the quality of service provided to each admitted client.

We have demonstrated the effectiveness of the adaptive admission control algorithm and the greedy disk scheduling algorithm through simulations. Our simulations reveal that the greedy disk scheduling algorithm yields an 80% improvement in performance over the conventional SCAN disk scheduling algorithm. Furthermore, employing the greedy disk scheduling algorithm in conjunction with the adaptive

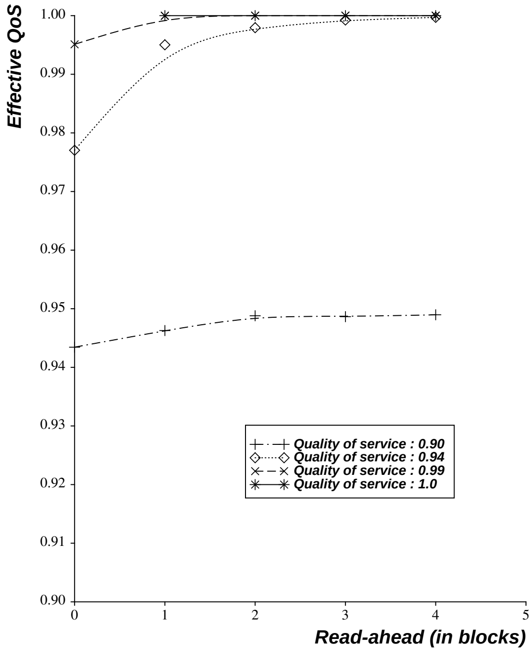

Figure 8 : Variation in the effective quality of service provided to clients with increase in read-ahead

admission control algorithm yields a server utilization of about 97%, and a 400% increase in the number of clients serviced simultaneously by the multimedia server. A prototype multimedia server, based on the algorithms presented in this paper, is being implemented at the UT Austin Distributed Multimedia Computing Laboratory.

REFERENCES

[1] D. Anderson, Y. Osawa, and R. Govindan. A File System for Continuous Media. *ACM Transactions on Computer Systems*, 10(4):311–337, November 1992.

[2] J.L. Bentley. Experiments on Geometric Travelling Salesman Heuristics. Technical Report Computing Science Technical Report No. 151, AT&T Bell Laboratories, Murray Hill, NJ, 1990.

[3] D.D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network. In *Proceedings of ACM SIGCOMM*, pages 14–26, August 1992.

[4] E. G. Coffman and M. Hofri. On Scanning Disks and the Analysis of their Steady State Behavior. *Proceedings of the Conference on Measuring, Modeling, and Evaluation Computer Systems, North-Holland*, pages 251–263, October 1977.

[5] K.P. Davies. Digital Television Broadcasting Dreams, Decisions, and Destinies. *SMPTE Journal*, pages 32–36, January 1993.

[6] E.A. Fox. Advances in Interactive Digital Multimedia Systems. *IEEE Computer - Special Issue on Multimedia Information Systems*, 24(11):9–19, October 1991.

[7] D. James Gemmell. Multimedia Network File Servers: Multi-Channel Delay Sensitive Data Retrieval. In *Proceedings of ACM Multimedia'93, Anaheim, CA*, pages 243–250, August 1993.

[8] J. Gemmell and S. Christodoulakis. Principles of Delay Sensitive Multimedia Data Storage and Retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, 1992.

[9] M. Hofri. Disk Scheduling: FCFS vs. SSTF Revisited. *Communications of the ACM*, 23(11):645–653, November 1980.

[10] S. Jamin, S. Shenker, L. Zhang, and D.D. Clark. An Admission Control Algorithm for Predictive Real-Time Service (extended abstract). In *Proceedings of Third International Workshop on Network and Operating Systems Support for Digital Audio and Video, San Diego, CA*, pages 308–315, November 1992.

[11] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A Fast File System for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.

[12] G. Miller, G. Baber, and M. Gilliland. News On-Demand for Multimedia Networks. In *Proceedings of ACM Multimedia'93, Anaheim, CA*, pages 383–392, August 1993.

[13] P. Venkat Rangan and Harrick M. Vin. Designing File Systems for Digital Video and Audio. In *Proceedings of the 13th Symposium on Operating Systems Principles (SOSP'91), Operating Systems Review, Vol. 25, No. 5*, pages 81–94, October 1991.

[14] A.L. Narasimha Reddy and J. Wyllie. Disk Scheduling in Multimedia I/O System. In *Proceedings of ACM Multimedia'93, Anaheim, CA*, pages 225–234, August 1993.

[15] M. Seltzer, P. Chen, and J. Ousterhout. Disk Scheduling Revisited. In *Proceedings of the 1990 Winter USENIX Conference, Washington, D.C.*, pages 313–323, jan 1990.

[16] D.R. Spears. Broadband ISDN - Service Visions and Technological Realities. *Journal of Digital and Analog Cabled Systems*, 1988.

[17] T. Teorey and T. B. Pinkerton. A Comparative Analysis of Disk Scheduling Policies. *Communications of the ACM*, 15(3):177–184, March 1972.

[18] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID: A Disk Storage System for Video and Audio Files. In *Proceedings of ACM Multimedia'93, Anaheim, CA*, pages 393–400, August 1993.

[19] Harrick M. Vin and P. Venkat Rangan. Designing a Multi-User HDTV Storage Server. *IEEE Journal on Selected Areas in Communications*, 11(1):153–164, January 1993.

[20] C.K. Wong. Minimizing Expected Head Movement in One Dimension and Two Dimension Mass Storage Systems. *Computing surveys*, 12(2):167–178, 1980.

[21] P. Yu, M.S. Chen, and D.D. Kandlur. Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management. *Proceedings of Third International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego*, pages 38–49, November 1992.