Retrieval Scheduling for Multimedia Presentations[*] (Extended Abstract)

Martha L. Escobar-Molano[†], David A. Barrett,
{mescobar,barrettd}@asgard.com
Asgard Systems

Zornitza Genova, Lei Zhang
{zgenova,lzhang}@csee.usf.edu
University of South Florida

Abstract

Advances in computer graphics, authoring tools and the explosive growth of the Internet has increased the use of multimedia presentations. This article presents a new retrieval scheduling technique to support the display of multimedia presentations in a multi-user environment. A multimedia presentation consists of a collection of objects with temporal constraints that define when the objects are rendered. A scheduling algorithm must determine when objects are retrieved from disk to satisfy the temporal constraints of the presentation. The time elapsed between the arrival of a request and the onset of its display (latency) depends upon the resources (CPU, disk, and memory) available to the system. The resources available depend upon those consumed by other presentations already being displayed. Therefore, the latency must be computed when the new request for a presentation arrives and that latency must include its computation time.

Prior scheduling techniques applicable to arbitrary resource requirements have quadratic time complexity. Unlike prior work, our scheduling algorithm has linear time complexity. We compare the performance of our scheduling technique with one that exhaustively searches for the earliest time to schedule a presentation. Our simulation results show that our technique significantly reduces the latency of a presentation as compared with the exhaustive search.

1 Introduction

A multimedia presentation consists of a collection of objects with temporal constraints that define when the objects are rendered. For example, a computer generated animation consists of a collection of 3-D objects that represent the characters and background of the animation with their time of appearances. To display a presentation, the storage system must deliver the participant objects to the renderer according to their time of appearances.

We assume that the unit of retrieval from disk into memory is a fixed-size page. Objects are not restricted by the page size. Smaller objects are clustered together into a page and larger objects are partitioned into multiple pages. We partition time into fixed-size time intervals. A retrieval schedule of a presentation defines for each time interval what pages to

[*]This research was supported, in part, by the National Science Foundation CAREER Award No. 9875528
[†]This work was done while this author was at the University of South Florida

retrieve from disk and what pages to discard from memory to satisfy the temporal constraints of the presentation. The retrieval schedule also determines the memory and disk bandwidth requirements of the presentation at each time interval.

The retrieval schedule overlaps the display of the presentation. To minimize memory requirements, the retrieval schedule aims to fetch pages from disk during the interval that preceeds their display. However, the disk bandwidth capacity might not be sufficient to retrieve all pages displayed at interval i during its preceeding interval $i-1$. Therefore, some pages might be pre-fetched at an earlier time interval. Only pages displayed during the first time interval and pages required to be pre-fetched before the beginning of the presentation are retrieved before the display of the presentation.

The system maintains a *system availability* that contains the resources still available while presentations are being displayed. The system availability is a sequence of tuples, one for each interval that the system has allocated resources for presentations being displayed. For a system with D disks, each tuple has $1+D$ elements and represents the available memory and bandwidth for each disk at the corresponding interval.

When a request for a presentation arrives, the system determines when to schedule the request so that the presentation requirements would not exceed the memory and disk bandwidth in the system availability.

An approach to determine when to schedule the request is to exhaustively search for the earliest sequence of time intervals when the presentation can be scheduled. Suppose that a request for a presentation with a retrieval schedule of k time intervals arrives at interval $t-\tau$, where τ is the time to search for this sequence. The system tries to start the retrieval schedule of the presentation at interval t. If there is a time interval when the memory and disk bandwidth requirements in the retrieval schedule exceed the the system availability, then it tries starting at $t{+}1$. If starting at $t{+}1$ also exceeds the availability, it tries at $t{+}2$, and so forth. When trying to start the retrieval schedule at t, the system compares pair-wise the system availability at intervals $t, t+1, \ldots, t+k-1$ with the requirements at intervals $0, 1, \ldots, k-1$ in the retrieval schedule. Since there are $D+1$[1] resources, it takes $k \times (D+1)$ comparisons to check whether the presentation can be scheduled starting at t. If the number of tuples in the system availability is n, it takes up to $k \times (D+1) \times n$ comparisons to determine when the presentation requirements would not exceed the memory and disk bandwidth available in the system. Therefore the time complexity of the exhaustive search is quadratic.

To illustrate, consider a system configuration with 36 disks ($D = 36$) and a CPU of 400 MHz. Suppose that we partition time into 4-second time intervals. When trying to schedule a 2-hour ($k = 1800$ time intervals) retrieval schedule on a system that has allocated resources for 24 hours ($n = 21{,}600$ time intervals), the system performs up to $1,800 \times 37 \times 21,600 = 1,438,560,000$ comparisons. Suppose that it takes 23 cycles to compare the required and available amounts of a single resource for a single time interval. Then, it takes up to $1,438,560,000 \times 23/400,000,000 = 82.71$ seconds to search for the earliest sequence of time intervals when the presentation can be scheduled ($\tau \leq 82.71$). Since the system does not know in advance how long this search will take, it must consider the worst case ($\tau = 82.71$) to guarantee that all the temporal constraints of the presentation are satisfied. Starting the display before this search ends might introduce disruptions at the end of the presentation because the system might not have available resources to retrieve the pages on time for their

[1] One for each disk and one for memory.

display.

In this paper, we present a technique that determines when to schedule a request in linear time. Our technique limits the number of comparisons to $lf \times n \times (D+1)$, where lf is a linear factor. For a linear factor of 50, our technique takes up to $50 \times 21,600 \times 37 \times 23/400,000,000 =$ 2.30 seconds to search for the time to schedule the presentation in the example above. As illustrated in this example, the search time following the exhaustive search is significantly higher than our technique.

2 Related Work

Previous studies [9, 1, 2] have investigated scheduling for continuous media (e.g., audio, stream-based video). These studies conceptualize a presentation as a file that is read sequentially at a pre-specified rate. They assume a data layout so that the disk reference pattern is regular, e.g., read the first block of a presentation from disks 0, 1, and 2 during the first time cycle, read the second block from disks 3, 4, and 5 during the second time cycle, and so on. Presentations sharing objects might reference them in different order. Therefore, finding a placement of objects that yields a regular disk reference pattern might be infeasible. Our scheduling technique works for arbitrary data layouts.

Retrieval scheduling for composite multimedia objects have been studied before [6, 8]. They conceptualize a presentation as a collection of multimedia streams with temporal constraints. However, their scheduling techniques have quadratic time complexity for presentations whose resource requirements (memory and disk bandwidth) vary arbitrarily over time.

Retrieval scheduling for presentations whose resource requirements vary arbitrarily over time have been studied before. In [5], an optimal retrieval scheduler for single disk architectures was proposed. This scheduler minimizes both latency and memory requirements. The complexity of resource scheduling for multi-disk architectures was studied in [4]. This study demonstrated that the computation of a resource schedule that satisfies a pre-specified display schedule and minimizes the startup latency is NP-Hard. In [3], a taxonomy of resource scheduling techniques that satisfy the display schedule of a presentation was proposed. This study introduced three resource scheduling techniques for multi-disk architectures and quantified their trade-offs. However, both [5] and [3] assume a single-user environment.

In a multi-user environment, the system has to find a sequence of time intervals where the memory and disk bandwidth available is sufficient to support the display. This problem can be stated as follows: given a sequence of requirements $r_1, \ldots, r_{m'}$, find a subsequence in the system availability $sys_{i+1}, \ldots, sys_{i+m'}$ such that for each $j \in [1, m']$: $r_j \leq sys_{i+j}$. For multiple resources, sys_j and r_j are vectors. We can pose the problem of finding a match for a string in a document in a similar manner. The string corresponds to the requirements and the document to the system availability. And, the matching criteria is equality: for each $j \in [1, m']$: $r_j = sys_{i+j}$. Like the string matching technique in [7], our scheduling techniques encodes the scheduling decision process into an Finite State Automaton (FSA). However, having $\leq$ as the matching criteria makes the encoding and execution of an FSA significantly different from the encoding and execution in the string matching problem.

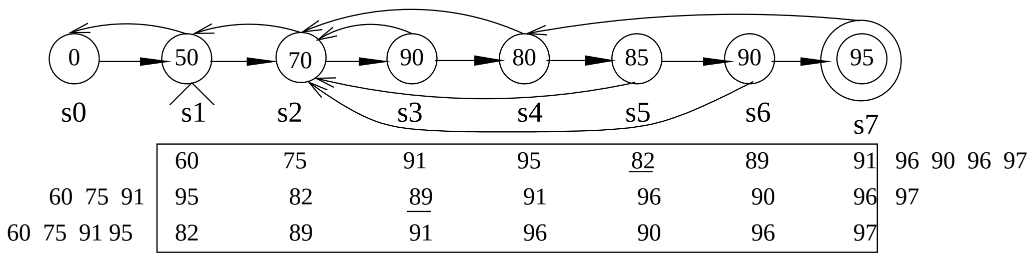

Figure 1: An example of an FSA for a single resource with a system availability being recognized by the FSA.

3 Scheduling Techniques

The proposed scheduling technique pre-computes the memory and disk bandwidth requirements of a presentation. The single-user retrieval scheduling techniques in [3] can be used to precompute these requirements. Based on these requirements, it builds an FSA and stores it on disk. This FSA recognizes the sequence of intervals when the resources (disk bandwidth and memory) in the system availability are greater than or equal to the resources required by the presentation. When a request arrives, the system loads and executes the FSA to determine when the presentation can be scheduled.

An FSA is a tuple $(P, \delta_f, \delta_b, s_1, s_{m'})$, where P is the set of states, δ_f and δ_b transition mappings, s_1 the starting state, and $s_{m'}$ the accepting state. If the retrieval schedule of a presentation requires m' time intervals, its FSA has $m' + 1$ states $(s_0, s_1, \ldots, s_{m'})$. There is a state for each time interval in the retrieval schedule and a sentinel state (s_0). Each state (s_i) is associated with the memory and bandwidth requirements of the presentation at the corresponding time interval (i). Transition mappings δ_f and δ_b represent forward and backward transitions, respectively. Forward transitions occur when the memory and disk bandwidth at the current time interval in the system availability are greater than or equal to the requirements associated with the current state in the FSA. Backward transitions occur where the requirements associated with the current state in the FSA exceed the system availability at the current time interval. The FSA starts at state s_1 and reaches state $s_{m'}$ when recognizes the sequence of intervals in the system availability when the presentation can be scheduled.

An FSA can be represented as a directed graph. To illustrate, suppose that we have only one resource in our system (say memory). Figure 1 represents the FSA for a presentation whose memory requirements for each interval are $50, 70, 90, 80, 85, 90, 95$. Each state is represented by a node (circle) with its requirements inside. The state with a zero in it is the sentinel state. Forward and backward transitions are represented by forward and backward links, respectively. The starting state is marked by a carat (s_1) and the accepting state is represented by a double circle (s_7).

Because of space limitations, we describe the algorithms here and refer the reader to the full paper for the pseudo-code of the algorithms.

3.1 Running the FSA

When a request for a presentation arrives, the system loads the FSA associated with the presentation and executes it. The FSA reads the first tuple in the system availability (memory and bandwidth available during the first time interval) and compares it with the requirements in state s_1 (memory and bandwidth required by the presentation during the first time interval). If the tuple in the system availability is greater than or equal to[2] the requirements in s_1 (the current state), the FSA advances to the next state by following a forward link and reads the next tuple from the system availability. If the tuple in the system availability is smaller than the requirements in the current state, the FSA advances to the next state by following a backward link but does not read the next tuple from the system availability. It continues comparing tuples in the system availability with the requirements in the states of the FSA until it reaches the accepting state or the end of the system availability. Following a backward link from s_i to s_j increases the latency by $i - j$.

To illustrate consider the example in Figure 1. The system availability is represented by the sequences of numbers below the FSA. When the FSA starts, it reads 60 from the system availability and compares it with 50 (the amount in s_1). Since $60 \geq 50$, the FSA follows the forward link and reads the next element in the system availability (75). Similarly, it compares $75, 91, 95$ with $70, 90, 80$, advances to state s_5, and reads 82. Since $82 < 85$, the FSA follows the backward link from s_5 to s_2. Following this backward link increases the latency by three time intervals. This increase is represented by a shift of 3 time intervals in the system availability as shown in the second line at the bottom of Figure 1. Then, it compares 82 with 70. Since $82 \geq 70$, the FSA follows the forward link and reads the next element in the system availability (89). Since $89 < 90$, the FSA follows the backward link from s_3 to s_2. Following this backward link increases the latency by one time interval, as shown in the third line at the bottom of Figure 1. Then $89, 91, 96, 90, 96, 97$ are compared with $70, 90, 80, 85, 90, 95$ and the FSA reaches the accepting state (s_7). Therefore, the request can be scheduled during the sequence of time intervals with $82, 89, 91, 96, 90, 96, 97$ as their memory available. The total latency is four time intervals. The sentinel state (s_0) is used to force reading the next element in the system availability. For example, suppose that the FSA runs on a system availability whose first element is 45. Since $45 < 50$, the FSA follows the backward link to s_0 and compares 45 with 0. Since $45 \geq 0$, it reads the next element in the system availability.

As shown in the full paper, the complexity of the algorithm to execute an FSA is $\mathcal{O}(n)$, where n is the number of tuples in the system availability.

3.2 Constructing the FSA

When constructing the FSA, a forward link is created between the states associated with two consecutive time intervals in the retrieval schedule of the presentation. When executing an FSA, the tuples in the system availability are compared with the requirements of the presentation in sequential order. Once a tuple t satisfies the requirements of the request in the current state, a new tuple is read and t is not referenced by the FSA anymore. For

[2]A tuple in the system availability is greater than or equal to the requirements in a state if and only if: (1) the memory available in the tuple is greater than or equal to the memory required in the state, and (2) for each disk d, the bandwidth available for d in the tuple is greater than or equal to the bandwidth required for d in the state.

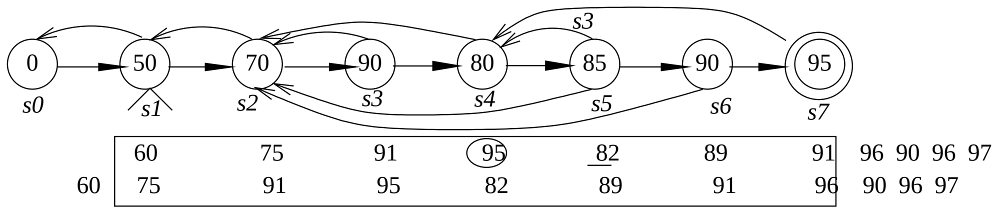

Figure 2: An example of an FSA for a single resource

example when processing 82 in the system availability, the FSA followed the backward link from s_5 to s_2 and did not have to verify that the memory available in the preceding time interval of the system availability (95) is greater than or equal to the requirements in s_1 (50). Therefore, the FSA must be constructed so that following a backward link during execution does not require comparing preceding tuples in the system availability with corresponding requirements. Thus, the FSA is constructed so that there is a backward link from s_i to s_j having the shortest distance $i - j$ that satisfies the following two conditions:

(i) The requirements in s_i are greater than the requirements in s_j, and

(ii) For all states s_k such that $1 \leq k < j$, the requirements in s_k are smaller than or equal to the requirements in s_{k+i-j}.

3.3 Refinements

To reduce the latency incurred by a presentation, we introduce additional backward links (termed *conditional links*). A conditional link from state s_i to s_j is represented by a labeled link. The labels specify the condition that must be satisfied in order to follow the link. A label is a sequence of states $s_{i_1}, \ldots, s_{i_k}$ whose requirements might be higher than the system availability. In order to follow a conditional link, the system must verify that the requirements in states $s_{i_1}, \ldots, s_{i_k}$ are smaller than or equal to the system availability. Conditional links do not satisfy Condition (ii) in Section 3.2. The labels in the link are the states s_k that violate Condition (ii). Figure 2 shows a conditional link from s_5 to s_4 with label s_3. A backward link from s_5 to s_4 increases the latency by one time interval. Therefore, the requirements of each pair of consecutive states before s_5 are compared to obtain the states that violate Condition (ii). Since s_3 is the only state that violates Condition (ii) ($s_3 > s_4$), the label of this link is s_3. This label will be used during execution to determine whether or not to follow the link. To illustrate, suppose that we are executing the FSA in Figure 2. During this execution, it reads 82 (in the system availability) and compares it with the requirements in s_5 (85). Since $82 < 85$, then it follows a backward link. There are two backward links: from s_5 to s_4 and from s_5 to s_2. It first tries the shortest link (from s_5 to s_4). Since this link is conditional, it checks first whether there is enough memory available to satisfy the requirements of s_3. If the link is followed, the element in the system availability that would correspond to s_3 is 95. Since $95 \geq 90$, the link is followed.

An FSA with conditional links is a tuple $(P, \delta_f, \delta'_b, s_1, s_{m'})$. δ'_b maps a state s to a sequence $L = [< s_{k_1}, seq_{k_1} >, < s_{k_2}, seq_{k_2} >, \ldots, < s_{k_l}, seq_{k_l} >]$ of pairs. Each pair $< s_{k_i}, seq_{k_i} >$ represents a target state and the label on the link. The sequence L is sorted by the increase of latency incurred by the link: $k_1 > k_2 > \ldots > k_l$. A backward link from s_i to s_{k_1} increases the latency by $i - k_1$, while a link to s_{k_2} increases the latency by $i - k_2$. Hence, the higher the subscript $(k_1, \ldots, k_l)$ is the lower the latency is.

The algorithm to execute an FSA with conditional links selects the backward link with the shortest latency such that the condition on its label is satisfied. Checking this condition increases the number of comparisons performed by the FSA. Therefore, our algorithm to construct the FSA selects the backward links that would bound the total number of comparisons during execution of the FSA to a linear factor lf of n $(lf \times n)$, where n is the number of tuples in the system availability. This bounding is achieved by selecting the backward links such that the number of comparisons while transiting a cycle from state s_{i-j} to s_i back to s_{i-j} is less than or equal to $lf \times j$. The full paper describes this bounding in detail.

There might be more than one set of backward links that satisfy the above selection criteria. To decide which link to include in $\delta'_b(s_i)$, our technique assigns priorities to each link. The priority of a link from s_i to s_j with label L is defined as $1 - \frac{|L|}{j-1}$. The value of $|L|$ is bounded by $j - 1$, therefore $\frac{|L|}{j-1}$ represents the percentage of number of states in the label over all possible states. The lower this percentage is the higher the priority of the link.

4 Evaluation

We compared our scheduling technique with a scheduler that exhaustively searches for the earliest time intervals in the system availability when the presentation can be scheduled. The performance of both techniques was evaluated using a simulation study and synthetic data.

As described in the full paper, we generated display schedules for 16 presentations of 100 minutes, 16 of 45 minutes, and 16 of 40 seconds. Once the display schedules of the 48 presentations were generated, we applied the memory-based scheduling technique in [3] to compute the memory and disk bandwidth requirements of each presentation assuming four different system configurations: (1) 1 GBytes of memory and 12 disks, (2) 2 GBytes of memory and 24 disks, (3) 3 GBytes of memory and 36 disks, and (4) 4 GBytes of memory and 48 disks. All configurations have a single CPU of 400 MHz and a page size of 128 KBytes. Each disk supports a 338.1 mbps transfer rate, 11.24 millisecond seek time, and 6 millisecond rotational latency.

The computed memory and disk bandwidth requirements are used by the exhaustive search to find the earliest time to start the retrieval schedule. These requirements are also used to build an FSA for each presentation. The FSAs were constructed with linear factors of 100 for 100-min and 45-min presentations and 6 for 40-sec presentations.

We assumed that the frequency of access of the presentations follows the Zipf distribution and the inter-arrival time of requests follows the Poisson distribution. Based on these assumptions, we generated lists of requests for arrival rates varying from 0.1 to 4.0 arrivals per minute. The span of request arrivals for these lists was 2 hours. We also assumed that besides displaying presentations on demand, the storage system supports pre-scheduled presentations. For our simulations, we assumed that the system pre-scheduled presentations for 8 hours.

We then schedule the requests in each list using both approaches: the exhaustive search and the FSA-Based technique. For each request, we compute the latency incurred by the presentation. This latency has 5 components:

(1) Time to retrieve pages referenced at the beginning of the display and to pre-fetch some pages

Table 1: Computation Time for Exhaustive Search vs FSA.

Movie	12 Disks		24 Disks		36 Disks		48 Disks		Linear
Length	ES	FSA	ES	FSA	ES	FSA	ES	FSA	Factor
40	0.064	0.032	0.124	0.064	0.184	0.092	0.244	0.120	6
2700	3.644	0.540	7.008	1.036	10.376	1.532	13.740	2.028	100
6000	8.088	0.540	15.552	1.036	23.016	1.532	30.484	2.028	100
Avg	4.422	0.433	8.587	0.822	12.620	1.213	16.645	1.606	NA

Computation time for the FSA algorithm depends upon a implementor-chosen linear factor. Avg is the average over the mix of all arrivals in the simulations. Values other than the Linear Factor are in seconds.

in preparation for the display. This component is derived from the memory-based scheduler and is identical for both approaches.

(2) Time to compute when to start the retrieval schedule of current request. This computation is dominated by the comparisons between available resources in the system and the requirements of the presentation. Therefore, we assume that this component is the time taken by the comparisons. For the exhaustive search, this time is $k \times n \times (D+1) \times 23/CPUfreq$, where k is the number of time intervals in the retrieval schedule of the presentation, n is the number of tuples in the system availability, D is the number of disks, 23 is the number of cycles per comparison, and $CPUfreq$ is the number of cycles per second. For the FSA-Based, this time is $lf \times n \times (D+1) \times 23/CPUfreq$, where lf is the linear factor.

(3) Delay due to a CPU conflict with the scheduling of previous requests. When a new request arrives while the CPU is still searching for the time to schedule a previous request, the system has to wait until the search is finished before it starts the search for the new request.

(4) Time to retrieve meta-data from disk. For the exhaustive search, this component is the time to retrieve the file containing the memory and bandwidth requirements of the presentation. For the FSA-Based, this component is the time to retrieve the file containing the FSA computed by our technique.

(5) Delay due to memory or disk bandwidth conflicts with other presentations being displayed. This component is the increase of latency due to shortage of resources in the system availability for the presentation requirements. For the FSA-Based, this component is the latency incurred when following the backward links.

The latency incurred by the FSA-Based technique does not include the time to build the FSA. The construction of the FSAs is done before the requests of presentations arrive. This construction is based on the memory and bandwidth requirements of each presentation, which is pre-computed (Section 3).

4.1 Results

Table 1 shows how the average time to compute the retrieval schedule (component 2) for our FSA algorithm compares with the exhaustive-search algorithm. In all cases the FSA consumes less computation time and the gap widens significantly as the as the movie length grows.

We seek to determine whether this reduction in computation for FSA will result in lower latency than exhaustive-search. Because FSA limits the number of comparisons by a linear

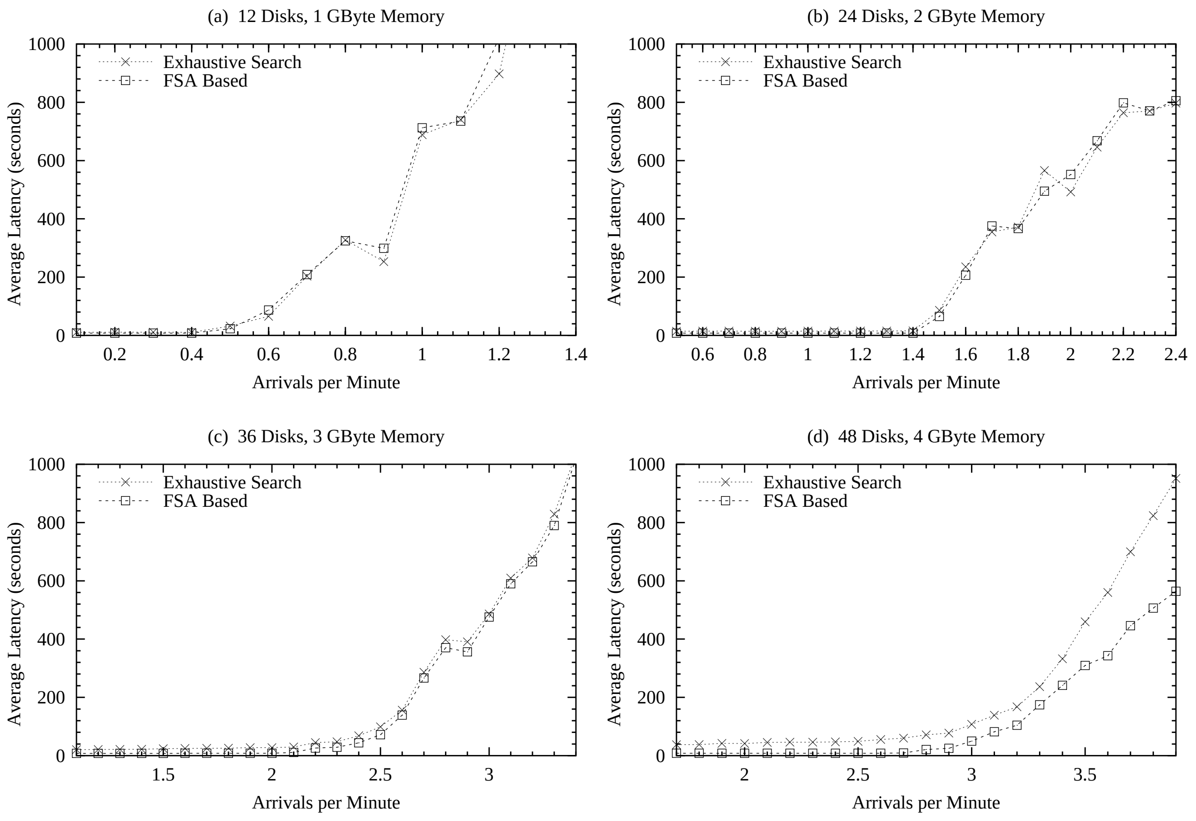

Figure 3: Total Latency. Total net average latency incurred for presentations as a function of request arrival rate for four system configurations.

factor, it will occasionally incur increased latencies because of resource conflicts with disk bandwidth or memory requirements.

Figure 3 shows the total latency for all components (1-5 combined). In Figure 3(a), for arrival rates below 0.5, the average total latency for our FSA technique was 8 seconds and varied from 10.7 to 11.2 seconds with exhaustive-search. Since the exhaustive-search always finds the earliest time intervals to schedule a presentation, the delay due to resource conflicts with other presentations is expected be lower for the exhaustive-search. For arrival rates greater than .5, the average latency with exhaustive search was up to 24% lower than with our scheduling technique (occurred at .6 arrivals per minute, too small to see in the figure).

In Figure 3(b), for arrival rates smaller than 1.5, FSA incurred 8 seconds total latency and exhaustive-search varied from 14.2 to 15.5 seconds. For all arrival rates, the average latency with exhaustive search was up to 13% lower (@1.9Arr/min) than the latency with our scheduling technique. In Figure 3(c) FSA slightly outperformed exhaustive-search for all arrival rates. CPU computation time (Table 1) and CPU contention (component 3) start to have a stronger influence for this configuration. Figure 3(d) shows that our FSA technique significantly outperformed the exhaustive-search algorithm for all arrival rates. The average latency of exhaustive-search was between 39% and six times higher (@2.7 Arr/min) than FSA. The average latency varied from 38 to 951 seconds for exhaustive-search and from 8.5 to 569 seconds for FSA. Computation time was significant for this configuration. Table 1 shows average computation time for exhaustive search was 16.6s as compared to 1.6s for FSA. Arrival rates of higher than 3.6 requests per minute yield inter-arrival times of less than 16.6

seconds. Therefore exhaustive-search suffers progressively further delays because the CPU is busy computing previous schedules when new requests arrive.

Thus, our FSA algorithm's improvement in computation time translates into a significant reduction in latency over exhaustive-search in this case.

5 Conclusions

This paper introduced a linear retrieval scheduling technique to support the display of multimedia presentations. We compared the performance of our technique with a scheduling technique that exhaustively searches for the earliest time intervals where the request can be scheduled. Simulation results show that the reduction on the computation time of our technique results in lower latencies (up to six times lower) than the exhaustive search, as the number of resources (disks) in the system increases.

Our scheduling technique reduces the computation time by budgeting the number of comparisons during the scheduling of a presentation. This paper introduced one alternative to budget the comparisons. One question that arises is how different alternatives to budget comparisons affect the outcome of the scheduling technique. Other future research directions include: retrieval scheduling techniques that maximize throughput, retrieval scheduling for dynamically generated display schedules such as in video games, and fault tolerant retrieval scheduling techniques.

References

[1] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of ACM-SIGMOD*, pages 79–89, May 1994.

[2] S. Chaudhuri, S. Ghandeharizadeh, and C. Shahabi. Avoiding retrieval contention for composite multimedia objects. In *Proceedings of Very Large Databases*, 1995.

[3] M. L. Escobar-Molano and S. Ghandeharizadeh. On Coordinated Display of Structured Video. *IEEE Multimedia*, 4(3):62–75, July-September 1997.

[4] M. L. Escobar-Molano and S. Ghandeharizadeh. On the Complexity of Coordinated Display of Multimedia Objects. *Theoretical Computer Science*, 242(1-2):169–197, 2000.

[5] M. L. Escobar-Molano, S. Ghandeharizadeh, and D. Ierardi. An Optimal Resource Scheduler for Continuous Display of Structured Video Objects. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):508–511, June 1996.

[6] M. N. Garofalakis, Y. E. Ioannidis, and B. Ozden. Resource Scheduling for Composite Multimedia Objects. In *Proceedings of Very Large Databases*, pages 74–85, August 1998.

[7] D. E. Knutt, J. H. Morris, and V. A. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[8] C. Shahabi, S. Ghandeharizadeh, and S. Chaudhuri. On Scheduling Atomic and Composite Multimedia Objects. *IEEE Transactions on Knowledge and Data Engineering*, To Appear.

[9] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID-A Disk Array Management System for Video Files. In *First ACM Conference on Multimedia*, pages 393–400, August 1993.