Caching and Scheduling in NAD-Based Multimedia Servers

Nabil J. Sarhan and Chita R. Das, *Fellow*, *IEEE*

Abstract—Multimedia-on-Demand (MOD) applications have grown dramatically in popularity, especially in the domains of education, business, and entertainment. Current MOD servers waste precious resources in performing store-and-forward copying. This excessive overhead increases cost and severely limits the scalability of these servers. In this paper, we propose using the Network-Attached Disk (NAD) architecture to design highly scalable and cost-effective MOD servers. In order to ensure enhanced performance, we propose a scheme, called *Distributed Interval Caching* (DIC), which utilizes the on-disk buffers for caching intervals between successive streams. We also propose another scheme, called *Multi-Objective Scheduling* (MOS), which increases the degrees of resource sharing by scheduling the waiting requests for service intelligently. We then integrate the two schemes and study the overall performance benefits through extensive simulation. The results demonstrate that the integrated policy works very well in increasing the number of customers that can be serviced concurrently while decreasing their waiting times for service. The performance benefits vary with several architectural, system workload, and scheduling parameters. We conclude this study by developing an analytical model for ideal DIC in order to estimate the performance limits which may be achieved through various optimizations.

Index Terms—Batching, Distributed Interval Caching (DIC), Multimedia-on-Demand (MOD), Multi-Objective Scheduling (MOS), Network-Attached Disks (NAD), Video-on-Demand (VOD).

1 Introduction

Recent advances in storage and communication technologies have spurred a strong interest in *Multimedia-on-Demand* (MOD) systems, which enable customers to access the multimedia contents they want at the times of their choosing and allow them to apply VCR-like operations. Besides its use for entertainment, MOD has been of great importance in education and distant learning in particular. *Video-on-Demand (VOD)* is the most common MOD application and is the application of interest in this study.

The design of VOD servers faces significant challenges to support large numbers of concurrent customers and to scale easily and cost-effectively in order to cope with the increasing demands. The number of customers that can be serviced concurrently is highly constrained by the requirements of the real-time playback and the high transfer rates. Hence, a wide spectrum of techniques has been developed to address this challenge, including *efficient data striping* [32], *data replication* [14], [9], *data block allocation and rearrangement* [27], [14], [29], *disk scheduling* [28], [26], *admission control* [3], and *resource sharing* [10], [19], [1], [22], [23].

Resource sharing strategies increase the number of serviced customers by exploiting the high skewness in video access patterns. These strategies can be classified into five main categories: *Batching* [10], [12], [1], *Patching* [22], [6], *Piggybacking* [19], *Broadcasting* [23], [21], and *Interval Caching* [11], [13], [33], [2]. Batching offloads the storage subsystem and efficiently uses server bandwidth and network resources by accumulating the requests for the same videos and servicing them together by utilizing the multicast facility. Patching is similar to batching, but it expands the multicast tree dynamically to include new requests, thereby improving resource sharing, but it requires additional bandwidth and buffer space at the client. Piggybacking offers similar advantages to patching, but it adjusts the playback rate so that the request catches up with a preceding stream, resulting in a lower-quality presentation of the initial part of the requested video. Broadcasting techniques divide each popular video into multiple segments and broadcast each segment periodically. The improved resource sharing here comes at the expense of requiring very high additional bandwidth and buffer space at the client. Interval caching caches intervals between successive streams for the same videos in the main memory of the server. It does not sacrifice the quality of playback, does not lengthen the waiting time, and does not expect much resource from the client. It can also be applied with other techniques and it has become more cost-effective with the falling prices of semiconductor memories.

The scalability challenge arises primarily because current VOD servers, such as Tiger Shark [20] and Tiger Video [4], are based on the conventional fileserver architecture (also known as the *Server-Attached Disk Architecture*), where all the data flowing from the disks to the clients pass through the server. The required *store-and-forward* copying imposes unnecessary burdens on these servers and severely limits their scalability.

Recently, the *Network-Attached Disk* (NAD) architecture [18], [17], [25] has emerged as a cost-effective solution to design scalable storage servers. In this architecture, the disks are connected to the network so that data can be transferred directly from the disks to the clients. The NAD architecture offers four major advantages over the conventional architecture. First, it scales inherently with storage capacity by

- *N.J. Sarhan is with the Department of Electrical and Computer Engineering, Wayne State University, 5050 Anthony Wayne Drive, Detroit, MI 48202. E-mail: nabil@ece.eng.wayne.edu.*
- *C.R. Das is with the Department of Computer Science and Engineering, The Pennsylvania State University, 220 Pond Lab, University Park, PA 16802. E-mail: das@cse.psu.edu.*

Manuscript received 21 Dec. 2002; revised 10 Dec. 2003; accepted 2 Mar. 2004.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 118034.

removing the server as a bottleneck. Hence, adding a new disk not only increases storage capacity, but also throughput and computing power. Second, it reduces cost by eliminating the resources used for store-and-forward copying. Third, it enhances performance with network striping and shorter data latency. Fourth, it helps to utilize the available on-disk computing powers and caches (buffers) effectively. The scalability and performance issues of the NAD architecture are discussed in [18], [25].

This study uses the NAD architecture to design highly scalable and cost-effective multimedia servers and ensures high performance by proposing an *Integrated Resource Sharing* policy. The main objectives of this policy are to increase the number of serviced customers while reducing their waiting times, to use server and network resources efficiently, to reduce the bandwidth and buffer space requirements at the client, to work well for large servers, and to provide high configurability. High configurability requires the policy to be tunable so as to make any necessary trade off (including the number of concurrent customers, waiting times, fairness, and implementation complexity) and to fit the current server workload, which may change with time, especially from daytime to nighttime and from business days to weekends.

The integrated resource sharing policy employs both caching and intelligent scheduling to enhance the performance of NAD-based multimedia servers. For caching, this study proposes a scheme, called *Distributed-Interval Caching* (DIC), which extends interval caching and adapts it to the NAD architecture. Modern hard disks have large internal caches and fast embedded processors, and these on-disk assets are likely to grow at a faster rate because of the scaling of technology and the widening market for NADs. DIC utilizes the internal caches for caching intervals between successive streams. Thus, DIC requires low-level control of the on-disk caches, which is enabled by the increased intelligence of modern disk drives. For scheduling, this study proposes a scheme, called *Multi-Objective Scheduling* (MOS), which increases the degrees of resource sharing by selecting the waiting requests for service based on four predefined criteria.

A further contribution of this work is studying VOD servers at the disk level and considering many design aspects, including admission control, batching, scheduling, and caching. Analyzing the integrated policy, rather than only the individual schemes, helps in understanding various trade offs, making appropriate design decisions, and ensuring that global optimizations can be attained.

The effectiveness of the integrated policy and the interactions among various system and workload parameters is demonstrated through extensive simulation. The primarily considered performance metrics are the average number of requests that can be serviced concurrently and the average request waiting time. The results show that the integrated policy, unlike most previous policies, can improve both these metrics simultaneously and can offer greater benefits for larger servers. The improvement in the number of concurrent requests approximately ranges from 12 percent to 17 percent as the number of disks increases from 60 to 240, using only 30 MB of cache per disk. The integrated policy also reduces the average request waiting time by about 8 percent in a 60-disk server to about 40 percent in a 240-disk server. The MOS scheme can simultaneously provide an additional 4 percent improvement in the number of concurrent requests and an additional 20 percent to 65 percent improvement in the average request waiting.

Finally, the performance limit of DIC is evaluated by analytical modeling. The limit is then used to determine whether further investigation is required to push the performance envelope.

The rest of the paper is organized as follows: We present the DIC scheme in Section 2, the MOS scheme in Section 3, and the integrated policy in Section 4. Then, we discuss the performance evaluation in Section 5. In Section 6, we develop an analytical model for an ideal DIC. Finally, we draw conclusions and outline the future work in the last section.

2 The Distributed Interval Caching (DIC)

Interval Caching [11] exploits the high skewness in video access patterns by attempting to pair each playback request with an immediately preceding request for the same video that is currently being serviced (either from the disks or the cache). Specifically, interval caching reuses the data brought by a stream in servicing a closely following stream if sufficient cache space exists to hold the data blocks in the cache. The two streams are called the *following* stream and the *preceding* stream, respectively. The required cache space for servicing the following stream depends on the time interval between the two streams and the compression method used. The server starts to cache the currently retrieved contents of the preceding stream after the two streams are paired up. Therefore, the following stream will have to be serviced from disks until the playback reaches the first cached block. The time during which the following stream is serviced from the disks is called the *catchup time*. Interval caching uses the cache (memory) space efficiently by victimizing longer intervals for caching shorter ones. This cache replacement policy is called *victimization*.

The DIC scheme proposed here extends interval caching and adapts it to the NAD architecture. With interval caching, data are cached in the main memory of the fileserver. Thus, using interval caching in the NAD architecture involves extra network activity in the process of bringing the *to-be-cached* data from the disks to the fileserver (filemanager) because the fileserver is no longer in the data path. More importantly, it negates the whole purpose of the NAD architecture (i.e., eliminating the fileserver as a bottleneck). In the proposed DIC scheme, however, an interval between two streams is cached concurrently in multiple disks using each disk's internal cache. Because of the distributed environment, the scheme has to deal with the arising complications in admission and victimization. DIC differs significantly from traditional caching techniques used in NAD systems [5], [25], which cache the most commonly/recently used objects rather than intervals between successive streams.

Before discussing the DIC scheme, let us discuss briefly how a typical multimedia server operates. In a multimedia server, videos are striped across all or a collection of disks.

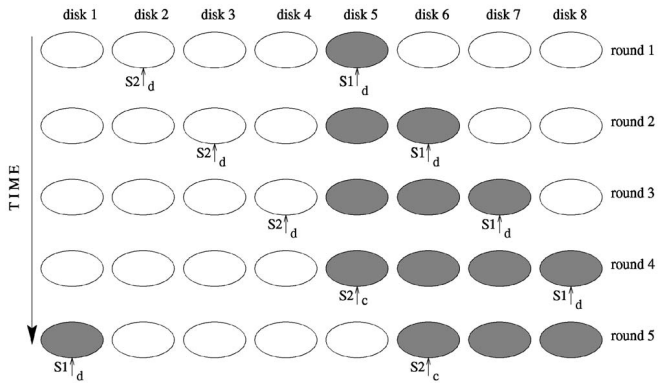


Fig. 1. Clarification of Distributed Interval Caching ($S1$: the preceding stream, $S2$: the following stream, f_disk: 2, p_disk: 5).

The server handles multiple clients by proceeding in periodic rounds and retrieving a fixed duration of media from one of these disks during each round. This fixed duration of media is called the *interleaving unit* (IU). For simplicity, let us assume that videos are striped across all N_d disks in a round-robin (RR) fashion. A mapping function, given by $Disk\_map(m) = m \bmod N_d$, is used to map the first IU of video m to a disk. This mapping helps to enhance the cache utilization and to balance the I/O load among all the disks.

The DIC scheme works as follows: First, it searches for the closest preceding stream for a new request. The scheme accepts the request for service from the on-disk caches if the request has a preceding stream, the corresponding interval can be cached in appropriate disks, and the request can be serviced from disks during catchup. If no sufficient cache space exists, then it tries to victimize a longer interval. Finally, if no victimization can be applied or the request has no preceding stream, then it resorts (if possible) to servicing the request directly from the disks (and not from their caches).

Let us now examine what happens after a request is accepted for service from the on-disk caches. Let us assume that, during round i, a new request calls for the playback of a video whose first IU is stored on disk f_disk and that disk p_disk will service the corresponding preceding stream in the next round. If the server accepts this request for service from the on-disk caches, then, in round $i+1$, disk p_disk caches the current IU of the preceding stream while it retrieves and transfers it to the client. Meanwhile, disk f_disk services the new request from its media (not cache). In the subsequent rounds, the next disks in the striping sequence will be involved in the process. After catchup, the following stream will be serviced from the on-disk caches until its completion.

Fig. 1 further explains the scheme for an 8-disk system. It shows what happens after *S2* (the following stream of video 2) is paired up with *S1* (the preceding stream of the same video), which came three rounds earlier than *S2* and is being serviced from disks. Here, f_disk is 2, p_disk is 5, the time interval length is $3u$, and the cached interval length (after catchup) is $(3+1) \times u = 4u$, where u is the size of the IU in seconds. The figure shows the process during the first five rounds in the service life of *S2*. The shading shows where the currently cached data are stored. The *c* or the *d* below an arrow shows whether the stream is being serviced from a cache or a disk, respectively. Note that *S2* is serviced for three rounds from disks (during catchup), and a wrap-around occurs as *S1* proceeds to round 5.

Next, the admission control and victimization issues are discussed and then the DIC algorithm is presented.

2.1 Admission Control

Striping videos in a round-robin fashion simplifies admission control. With the RR scheme, all the requests serviced by a disk in a round will be serviced by the next disk in the next round. Hence, a request can be serviced from the disks if there is sufficient I/O bandwidth in disk f_disk during the first round. This condition guarantees sufficient I/O bandwidth for the request in all subsequent rounds.

Deciding whether a request can be serviced from the on-disk caches is somewhat tricky because an interval may be cached concurrently in multiple disks. In this case, the server has to ensure that sufficient cache space exists in each disk between disk f_disk and disk p_disk, inclusively, during the first round. (Specifically, these disks are f_disk, $(f\_disk+1) \bmod N$, $(f\_disk+2) \bmod N$, ..., and p_disk.) Note that, in Fig. 1, the availability of sufficient cache space in disks 2 through 5 in round 1 guarantees that the interval between the two streams can be cached in all subsequent rounds. (With the round-robin scheme, if there is sufficient cache space in disks 2 through 5 in round 1, then there will be sufficient cache space in disks 3 through disk 6 in round 2, sufficient cache space in disk 4 through disk 7 in round 2, and so on.) In determining the required cache space of each of these disks, we must account for long intervals that may require storing more than one IU on each of these disks. Thus, caching can only be applied if sufficient cache space exists to store $\lceil \frac{I+u}{N_d \times u} \rceil \times u$ seconds in each disk between and including disk f_disk and disk p_disk, where u is the size of the IU and I is the time interval between the two streams, both in seconds.

2.2 Victimization

When no sufficient cache space exists for servicing a new request from the on-disk caches, the server can victimize an existing request from the caches only if all the following four conditions are met:

1. The potential victim can be serviced from the disks.
2. The new request can be serviced from appropriate disks during catchup time.
3. The victimization can create sufficient cache space for the new request.
4. The interval length of the new request is shorter than that of the potential victim.

Victimizations ensure efficient usage of the available cache space at the expense of increasing the implementation overhead. This overhead, however, is an important issue in the NAD environment because of the relatively modest on-disk computing powers. Therefore, a parameter, called *Victimization Ratio* (VR), is introduced and the last condition is changed so that the interval length required by the new request is shorter than VR times the

```
01 for (each waiting request){
02   set m to the request's video number;
03   f_disk = Disk_map(m); // map a video to a starting disk
04   // Check for available disk bandwidth
05   if (no request can be serviced from disk f_disk) return;
06   // Try to service the request from the caches
07   Search for a preceding stream;
08   if (there is a preceding stream) {
09     Set I to the length of the interval in seconds between the request and its preceding stream;
10     p_disk = (f_disk + I/u) mod N_d;
11     if (⌈(I+u)/(N_d×u)⌉ × u seconds can be cached in each disk between disks f_disk and p_disk, inclusive){
12       Accept the waiting request for service from the caches;
13       Insert the cached interval into CIQ;
14       return;
15     } // end of if
16     // Try to victimize
17     while (there are more elements to examine in CIQ){
18       Get the next interval, Im, from CIQ;
19       Set l to the length of the time interval of Im in seconds;
20       Set a_victim to the disk that will service the potential victim in the next round;
21       if (I > VR × l ) break; // stop trying to victimize
22       if (the potential victim can be serviced from a_victim in the next round and
             victimization can create sufficient cache space for the new request){
23         Victimize the request corresponding to Im;
24         Remove Im from CIQ;
25         Accept the waiting request for service from the caches;
26         Insert the new cached interval into CIQ;
27         return;
28       } // end of if
29     } // end of while
30   } // end of if
31   Accept the waiting request for service from the disks;
32 } //end of for
```

Fig. 2. The Distributed Interval Caching Algorithm.

interval length required by the potential victim. VR, which can take any value between zero and one, inclusive, should be tuned carefully in order to avoid victimizations that may not lead to worthwhile performance benefits.

2.3 DIC Algorithm

Fig. 2 presents the algorithm in a C++-like syntax. The algorithm assumes that the disks have sufficient network bandwidth to handle their I/O rates. Note that a request can be admitted only if it can be serviced from the disks during its entire service time or only during catchup if this request can be serviced from the on-disk caches afterward. Also, note that all cached intervals are inserted into a priority queue (in an appropriate format) and placed in descending order of interval length. This queue is called the *Cached-Interval Queue* (CIQ) and is used to determine whether a victimization can be applied. The intervals in CIQ are examined from top to bottom for a possible victimization. The search terminates successfully when the four conditions are met or unsuccessfully when the time interval of the new request is greater than $VR \times l$, where l is the time interval of the currently examined cached interval.

3 Multi-Objective Scheduling (MOS)

A VOD server maintains a waiting queue for each video, applies a scheduling policy in order to select one queue for service whenever the necessary resources become available, and services all the requests in the selected queue together using only one stream. The main objectives of scheduling policies are increasing the number of customers that can be serviced concurrently and decreasing their waiting times. Unfairness is another objective that measures the bias against unpopular videos. Scheduling for VOD services has been studied extensively [10], [1], [30] and many scheduling policies exist. Previous work shows that choosing the most appropriate scheduling policy greatly depends on server

workload and several design trade offs [30]. These results motivate combining scheduling policies for higher flexibility and performance.

The proposed *Multi-Objective Scheduling* (MOS) scheme combines the advantages of the following four scheduling criteria:

- **Maximum Queue Length** (MQL) [10]—This criterion maximizes the number of requests serviced at any time by selecting the longest queue and can be captured by the optimization function $F_b(i) = B_i$, where i is the video number and B_i is the number of waiting requests for video i. MQL performs well in terms of the number of serviced customers and the waiting times, but tends to be biased against unpopular videos.
- **First Come First Serve** (FCFS) [10]—This criterion acts fairly by selecting the queue with the oldest request and can be captured by $F_t(i) = T_i$, where T_i is the longest request waiting time for video i. FCFS can also help in reducing the number of unserviced customers because customers are more likely to leave the system without being serviced if they wait longer.
- **Shortest Interval First** (SIF)—This criterion, which we propose here, selects the queue that requires the shortest cached interval and can be captured by $F_n(i) = 1/I_i$, where I_i is the required interval to cache the requests for video i. SIF is important in the NAD environment because of the highly constrained victimizations. So, it is important to select the shortest intervals during scheduling rather than to rely entirely on subsequent victimizations as done in traditional interval caching.
- **Most Popular First** (MPF)—This criterion, which we propose here, selects the queue of the most popular video among all nonempty queues and can be captured by $F_m(i) = 1/i$. (Videos are numbered in decreasing order of popularity.) MPF favors popular videos which generally lead to higher degrees of resource sharing.

These criteria are not completely orthogonal. For instance, SIF is related to both MPF and MQL. SIF, however, exploits the situations when the video requiring the shortest cached interval is not the most popular requested video and is not the video with the largest number of waiting requests. The total probability with which these situations happen is significant. Similarly, MPF is related to both MQL and SIF, but it has strong ability to predict the streams that are more likely to have closer following streams, which can be paired up with using smaller cache space.

The MOS scheme combines the four criteria by applying the optimization function $F(i)$, which is a weighted sum of the normalized $F_b(i)$, $F_t(i)$, $F_n(i)$, and $F_m(i)$:

$$F(i) = w_b \frac{B_i}{\max B_j} + w_t \frac{T_i}{\max T_j} + w_n \frac{\min I_j}{I_i} + w_m \frac{\min j}{i},$$

where w_b, w_t, w_n, and w_m are the weights assigned for the above criteria, respectively, and $\min j$ is the lowest video number among those with waiting requests. Normalizations limit the contribution of each criterion to a value between zero and one, inclusive, and then the weights adjust the contributions based on the emphasis assigned to each criterion. The linear combination of various objectives serves as a heuristic in the absence of an optimal function, which is extremely difficult to obtain. Each of the four criteria can help in increasing the number of serviced customers. For other objectives, however, not all of them may have a constructive effect. Therefore, the weights, which are referred to as the *scheduling parameters* in the subsequent analysis, should be tuned carefully to achieve the desired trade off. For example, one way to enhance fairness is to increase w_t and reduce w_m. Enhancing caching performance, however, can be attained by increasing w_i and reducing w_t. In Section 5, we demonstrate how the scheduling parameters can be tuned in order to simultaneously increase the number of serviced customers and reduce their waiting times.

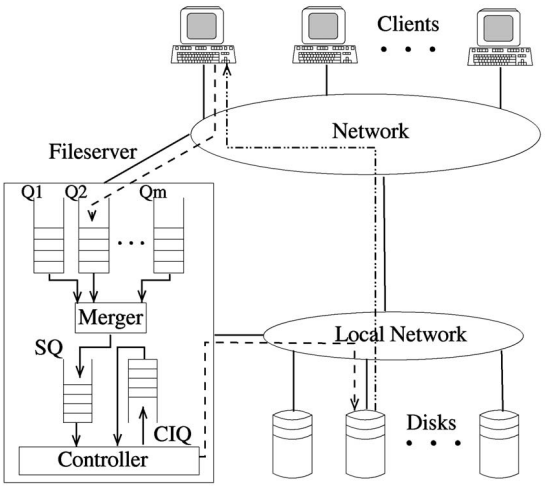


Fig. 3. Clarifications of the overall architecture and the integrated policy.

4 The Integrated Resource Sharing Policy

The integrated policy combines the DIC and the MOS schemes and is implemented by maintaining a priority queue, called the *Scheduling Queue* (SQ). Fig. 3 shows the overall architecture and how the integrated policy works during one round. $Q1, Q2, \ldots,$ and Qm are the video waiting queues and *CIQ* is the *Cached-Interval Queue*. CIQ stores all currently cached intervals in descending order of interval length. The intervals in CIQ are examined from top to bottom for a possible victimization. The *merger* combines the requests of each nonempty waiting queue i into one aggregate request and inserts it into SQ according to $F(i)$. The *controller* considers the requests for service in decreasing order of F, updates CIQ, and implements the rest of the integrated policy. Note that more than one aggregate request can simultaneously be admitted for service using different streams. This, of course, depends on the availability of the required resources, which may change from one round to another. The dashed lines show how a client requests the playback of video 2 and how the controller translates that into commands to a disk. The dash-dot-dot line shows the direct data path between the client and the disk.

TABLE 1
Main Disk Parameters

Parameter	Value
Storage Capacity (GB)	9.1
Rotation Speed (rpm)	7200
Number of Cylinders	8057
Number of Surfaces	10
Full Strobe Seek Time (ms)	15.36

The integrated policy enhances the DIC algorithm by incorporating two parameters: *Request Check Ratio* (RCR) and *Delay Service Ratio* (DSR). These parameters, which can take any value between zero and one, inclusive, should be carefully tuned. RCR specifies the fraction of the requests in SQ that will be considered for service. A small value of RCR reduces the implementation overhead and may lead to the selection of only the "best" requests, which are likely to boost performance. It, however, lengthens the delay in servicing some requests, which in turn may lead to lengthening the average cached interval and increasing the customer defection probability. (The defection probability is the probability that a new customer leaves the system without being serviced because of a waiting time exceeding the user's tolerance.) The idea behind DSR is to delay servicing from disks the requests that cannot be cached currently with the hope that they can be cached in subsequent rounds. DSR is the maximum ratio of the request waiting time to the *Maximum Waiting Time* (MWT). If the ratio of the request waiting time to MWT is less than DSR, then the server delays the request further. A large value of DSR improves both batching and caching, but it increases the average request waiting time.

The integrated policy requires some coordination between the filemanager and the disks. The filemanager manages SQ and decides whether to admit requests. The disks, however, need to know what data to be retrieved during each round and to what destinations. Hence, the filemanager may have to send this information periodically to the disks. Alternatively, the disks themselves can keep track of all necessary information about the running streams and the global organization of data, and the filemanager only needs to send updates upon the initiations of new streams, applications of victimization, and applications of VCR-like operations. The latter scheme reduces the control traffic, but increases the involvements of the individual disks.

5 Performance Evaluation

This section discusses the effectiveness of the proposed techniques through extensive simulation and studies the performance limit of DIC through analytical modeling. In the target server, the disks are connected directly to the network and the on-disk caches are used for caching intervals between successive streams.

5.1 Workload Characteristics

In accordance with prior work, the evaluation study here assumes that the arrivals of the requests to multimedia servers follow a Poisson Process with an average arrival rate λ. Hence, the interarrival time is exponentially distributed with a mean $T = 1/\lambda$. It also assumes that videos are stored using the MPEG-II compression standard and that the accesses to videos follow Zipf-like distribution [7]. Moreover, it assumes that customers can wait until a predefined maximum time (MWT). Furthermore, it assumes that videos are striped across all disks on 1/2-second intervals [8]. This assumption, however, can be relaxed to stripe across any number of disks. To keep the discussion focused, VCR-like operations are not considered and the network bandwidth for each disk is assumed to be sufficient for supporting the disk transfer rate. VCR-like operations can be supported by allocating contingency channels [12].

TABLE 2
Estimated Values of Disk Performance Parameters

Parameter	Value
Mean Seek Time (ms)	7.778
Mean Rotational Time (ms)	2.431
Mean Transfer Rate (MB/s)	4.217

5.2 Simulation Platform

The evaluation study here was conducted by developing a simulator for NAD-based multimedia servers. The simulator takes numerous workload, system, and policy parameters in addition to three disk performance parameters: mean seek time, mean rotational latency, and mean data transfer rate. For simplicity, the simulator does not consider the variability in disk access times. This study experimented primarily with *Quantum Atlas III* disks. Table 1 shows its main parameters. The study used *DiskSim* [15] to find the disk performance parameters required by the developed simulator. In this process, DiskSim was fed with synthetic workloads (described in the previous subsection) and with validated disk parameters obtained from [16]. Table 2 shows the main disk performance parameters, which are found by simulation.

5.3 Evaluation Methodology

The primary objective functions analyzed here are the number of requests that can be serviced concurrently and the average request waiting time. The integrated policy, however, can also be used to optimize other performance metrics.

Table 3 shows the main input parameters and their default values. The impact of each of these parameters is studied. The number of disks in the server is varied from 30 to 240 and the number of videos is varied accordingly to keep all disks nearly full. Only a small range of the disk cache size (0 to 40 MB) is investigated because of the limited disk power budget [24]. The request arrival rate, λ, is usually selected such that the customer defection probability is about 0.1 when the cache size per disk is 30 MB.

To evaluate the effectiveness of the integrated policy, two cases are considered: *Caching is Off* and *Caching is On*. In the first case, caching is disabled and the two parameters, RCR and DSR, are not utilized (effectively, $RCR = 1$ and $DSR = 0$), but both batching and scheduling are employed, and the values of the other parameters

TABLE 3
Main Input Parameter and Default Values

Parameter	Symbol	Default Value
Number of Disks	N_d	120
Cache Size per Disk (MB)	S	30
Number of Videos	N_v	480
Video Data Rate (MB/s)	r	0.375
Video Duration (s)	D	90×60
Request Arrival Rate (Request/s)	λ	0.435
Video Skewness Parameter	θ	0
Maximum Request Waiting Time (s)	MWT	3×60
Victimization Ratio	VR	0.7
Request Check Ratio	RCR	0.5
Delay Service Ratio	DSR	0.1
Scheduling Parameters	$\{w_b, w_n, w_m, w_t\}$	$\{0, 0, 0, 1\}$

are kept the same as those when caching is turned on to ensure a fair comparison.

Then, the contribution of the MOS scheme is evaluated when caching is on by varying only the values of the scheduling parameters. *Maximum Factored Queue Length* (MFQL) [1] is not considered in this study because it serves as a compromise between FCFS and MQL [1], while this study aims at developing a scheme that can outperform both FCFS and MQL in both primary objectives.

5.4 Simulation Results

The following simulation results were obtained using a steady state analysis with 95 percent confidence interval.

5.4.1 Effectiveness of the Integrated Policy

Let us now examine the effectiveness of the integrated policy and discuss the impacts of the cache size per disk, the number of disks, and the request arrival rate. Fig. 4 plots the number of concurrent requests that can be serviced versus the cache size per disk and the number of disks. In this section, caching is turned off when the cache size is 0. As expected, the number of concurrent requests increases with the number of disks and the cache size, but the number of disks plays a more significant role. Next, the number of disks is fixed (at 120) and the impact of the cache size per disk is analyzed in terms of the number of concurrent requests and the customer defection probability (shown as a percentage). Fig. 5 shows that both these metrics exhibit a faster growth/decay when the cache size is small. This happens primarily because the integrated policy improves performance through both batching and caching, but the cache size does not contribute to batching. Fig. 6 depicts the impact of the number of disks while the cache size per disk is kept constant (30 MB). Note that the benefit of the integrated policy generally increases with the number of disks. This behavior is due to the increase in λ that the server can handle. Increasing λ shortens the intervals between successive streams, thereby allowing the server to service more requests from the on-disk caches. In addition, as λ increases, batching improves since more requests accumulate in the waiting queues.

Fig. 7 plots the percentage improvement achieved by the integrated policy versus the number of disks and it also shows the contributions of caching and batching. Note that the improvement by the integrated policy in the number of concurrent requests approximately ranges from 6 percent to 17 percent as the number of disks increases from 30 to 240. Caching contributes to a little less than half of the improvement on the average. The improvement resulting from caching does not follow a linear relationship with the number of disks because of the complex interaction between caching and batching: Caching performance improves by servicing requests immediately, whereas batching benefits primarily from delaying their service.

Fig. 8 depicts the number of concurrent requests versus the average request interarrival time ($1/\lambda$). These results show that the number of concurrent requests increases with λ. As stated earlier, this is due to the increase in the average batch size and the number of cached requests. Note that the

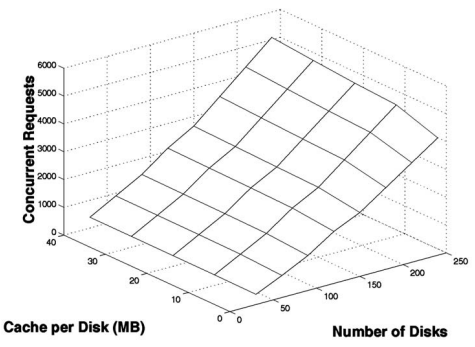


Fig. 4. Effect of the number of disks and cache size.

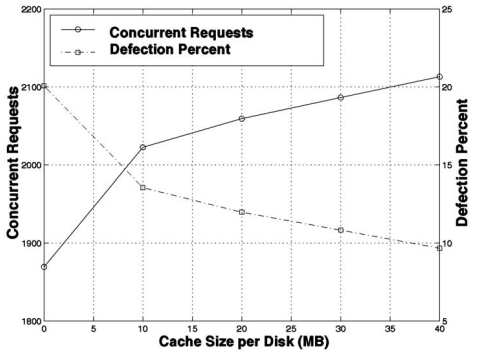


Fig. 5. Effect of cache size.

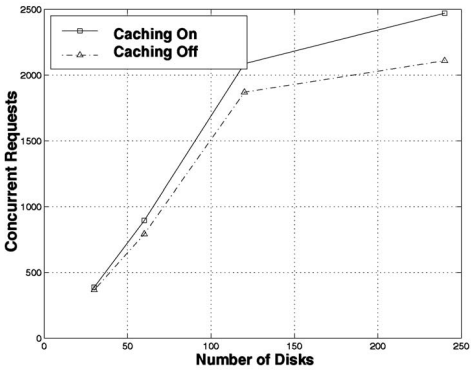


Fig. 6. Effect of the number of disks.

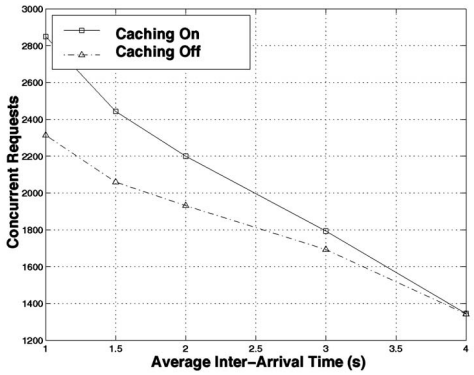


Fig. 8. Effect of average interarrival time.

improvement attained by the integrated policy also increases with λ. The zero-improvement point happens when the server is lightly loaded and, so, it is able to service all incoming requests (defection probability is zero).

Fig. 9 illustrates the effects of the number of disks and the cache size per disk on the average request waiting time. The results show that, when caching is turned off (i.e., $cache\ size = 0$), the average request waiting time increases by a factor of two as the number of disks varies from 30 to 240. This behavior is because videos are striped across all disks, so requests wait longer to find empty slots for scheduling. With the integrated policy, however, the average request waiting time increases less dramatically with the number of disks up to a certain point and then starts to decrease since caching services requests as soon as possible. With 30 MB of cache per disk, the reduction in waiting times increases from about 4 percent to about 40 percent as the number of disks varies from 30 to 240. The average request waiting time, however, remains more than 1.2 minutes, which is 41 percent of MWT (3 minutes). Such a long average delay leads to a considerable number of customer defections. Thus, dividing disks into partitions can enhance the performance by reducing the waiting times.

5.4.2 Effects of the Tunable Parameters: VR, RCR, and DSR

Let us now discuss how VR, RCR, and DSR can be tuned. Fig. 10a shows the effect of VR on performance. This figure indicates that designers should choose the largest possible value of VR whose incurred implementation overhead can be tolerated. The value 0.7 may lead to a good trade off because the number of concurrent requests increases slightly with VR after that point.

Fig. 10b plots the number of concurrent requests versus RCR. Note that the number of serviced requests increases with RCR since more requests are examined for scheduling. Ideally, the value of RCR should be kept between 0.3 and 0.5 unless the implementation overhead resulting from using larger values can be tolerated.

The impact of DSR on the number of concurrent requests and waiting times is shown in Fig. 10c. As expected, increasing DSR has a positive effect on the number of concurrent requests and a negative effect on waiting times. Note that increasing DSR from 0.1 to 0.9 increases the number of concurrent requests by about 5.3 percent and increases the waiting times by about 40 percent. Hence, if customers are not very sensitive to longer waiting times, the performance of the server can be enhanced by using a larger value of DSR.

5.4.3 Effects of the Scheduling Parameters

Let us now examine the effectiveness of the MOS scheme. Although this scheme can be used to meet any design trade off, the main goal of the evaluation study here is to demonstrate that it can increase the number of concurrent customers while reducing their waiting times for service. Simulation results show that, in order to meet this objective, MQL should have a large weight and the other criteria should have small weights. The idea behind these selections of weights is to make MQL determine primarily the scheduling

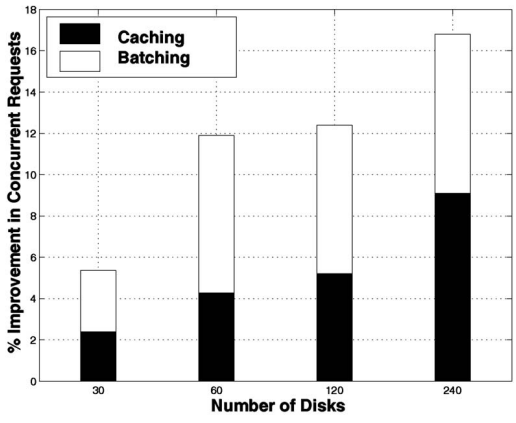


Fig. 7. Contributions to improvements in concurrent requests ($VR = 0.99$).

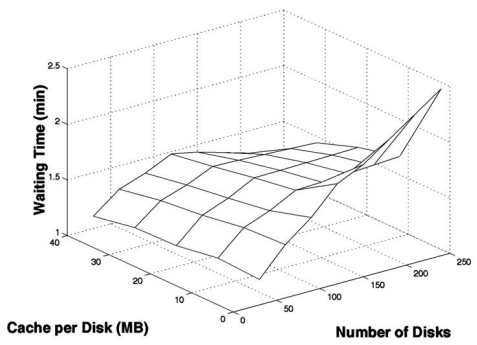


Fig. 9. Effect of the number of disks and cache size on waiting time.

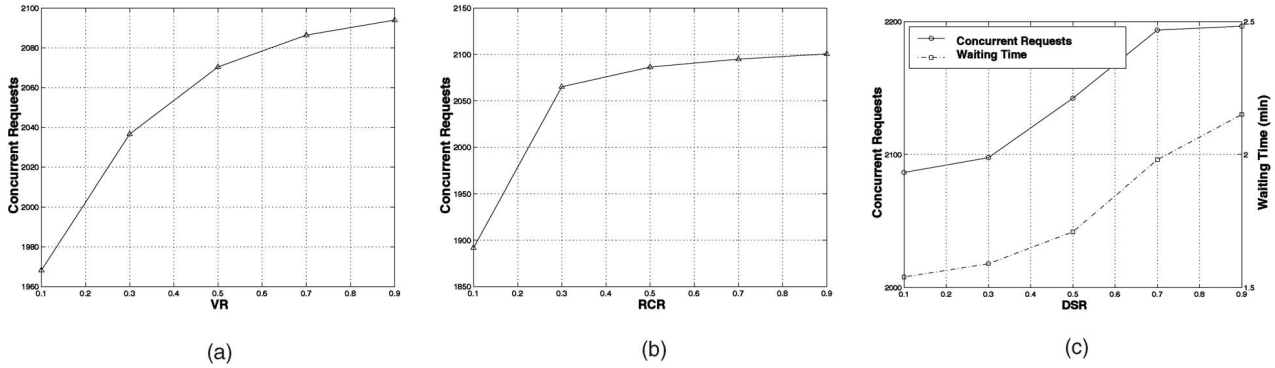


Fig. 10. Effects of tunable parameters. (a) Victimization Ratio (VR), (b) Request Check Ratio (RCR), and (c) Delay Service Ratio (DSR).

decisions, while allowing the other criteria to influence these decisions only when multiple queues have almost the same number of waiting requests. (These situations happen with significant probability because the queue length is finite and, typically, a small number.) In addition to the four scheduling criteria (discussed in Section 3), two promising special cases of the MOS scheme are examined: *MOS 1* ($w_b = 1, w_t = 10^{-4}$, $w_n = 10^{-5}, w_m = 0$) and *MOS 2* ($w_b = 1, w_t = 10^{-5}, w_n = 10^{-6}$, $w_m = 10^{-4}$).

Figs. 11 and 12 plot the percentage improvements achieved by the different criteria with respect to MQL in terms of the number of concurrent requests and the waiting times, respectively, for three settings of λ and RCR. The results of FCFS are not shown in the figure because of its poor performance: It performs 25 percent to 40 percent worse than MQL in terms of the number of concurrent requests and 120 percent to 150 percent worse than it in the waiting times. These results differ from prior work primarily because this study considers caching, whose performance degrades substantially by selecting older requests. The number of concurrent requests with FCFS is relatively small because this criterion utilizes neither batching nor caching and the average waiting time is relatively long because it favors older requests. SIF performs relatively well in terms of the waiting times because it favors newer requests. Similarly, the waiting times are longer with MOS 1 than those with MOS 2 because MOS 1 has a larger weight w_t. In all the three settings (of λ and RCR), SIF and MOS 2 reduce the waiting times better than MQL and MOS 1 outperforms MQL in terms of the number of concurrent requests.

The main results can be summarized as follows: The values of the scheduling parameters should be based on the server load (which depends on λ) and RCR. With a careful tuning of these parameters, the MOS scheme can perform better than MQL in terms of both the number of concurrent requests and the waiting times. Both these performance metrics can be improved with a very large value of w_b and a very small value of w_t. A detailed analysis of the MOS scheme and how the scheduling parameters should be selected is left for a future study.

5.4.4 Effects of Other Parameters

Let us now discuss the impacts of disk performance, the maximum request waiting time (MWT), the number of videos, the video data rate, the video duration, and the skewness parameter (θ). Fig. 13 shows the effect of disk performance (measured as the number of concurrent streams it can service) on the overall performance. The value 10 in the x-axis corresponds to Atlas III, while higher values correspond to more advanced disks. Note that the performance improvement achieved by the integrated policy ranges between 12 percent and 14 percent.

The effect of MWT on the number of concurrent requests is shown in Fig. 14. As MWT increases, more requests can be serviced because they can wait longer and the degree of resource sharing improves.

Fig. 15 shows that the number of concurrent requests decreases with the number of videos, which can be explained as follows: The video locality of reference diminishes as the number of videos increases since every additional video consumes part of the total access frequency. The reduction in

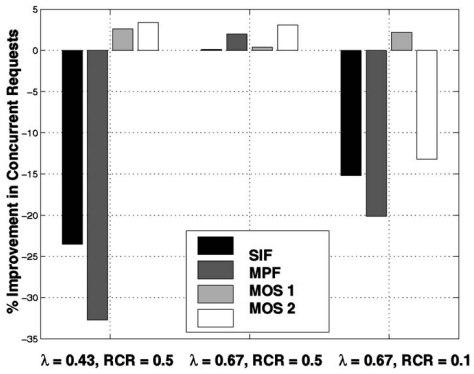


Fig. 11. Improvements in concurrent requests with respect to MQL.

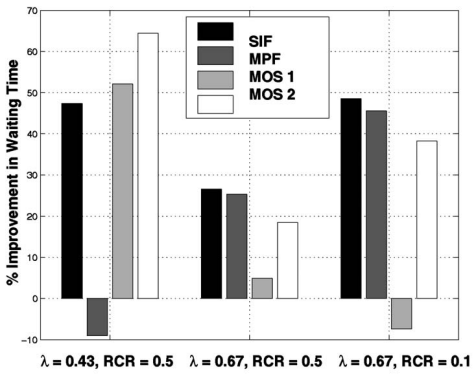


Fig. 12. Improvements in waiting times with respect to MQL.

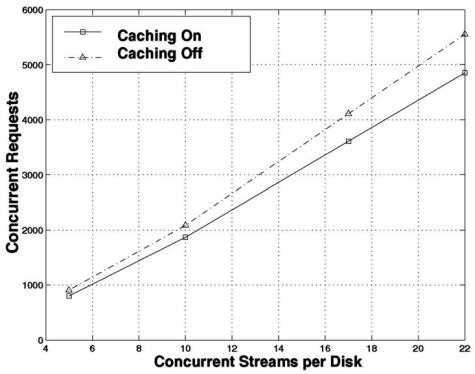


Fig. 13. Effect of disk performance.

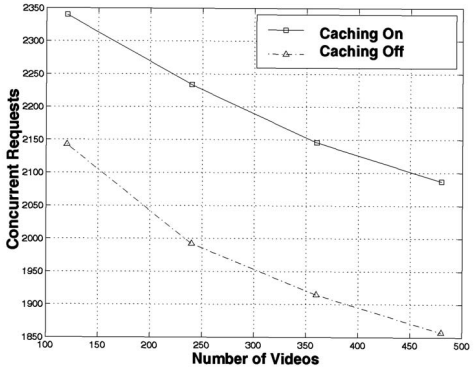


Fig. 15. Effect of number of videos.

the locality of reference has a negative impact on both caching and batching. In addition, as the number of videos increases, the ratio of overall cache size to the overall data size decreases. The figure also shows that the percentage improvement of the integrated policy increases with the number of videos. Increasing the video data rate has the same effect as decreasing the number of videos because this number is adjusted to keep all disks nearly full. Similarly, increasing the video duration has the same impact as decreasing the number of videos.

Finally, the impact of the skewness parameter (θ) on the number of concurrent requests is depicted in Fig. 16. As expected, both the number of concurrent requests and the improvement of the integrated policy diminish as the skewness in access patterns decreases.

6 Evaluation of the Effectiveness of the DIC Algorithm

The last section demonstrates that the performance improvement achieved by the DIC scheme—with only 30 MB cache per disk—in the number of concurrent requests ranges from 2 percent to 9 percent as the number of disks varies from 30 to 240. Naturally, these improvements are less significant than the improvements achieved by caching in general-purpose systems because of the much larger object size(s) in media servers. But, can we squeeze additional performance out of NAD-based media servers by polishing the proposed scheme? This section answers this question by finding the performance limits on DIC. For this purpose, we model an *ideal* DIC that minimizes the average cached interval by perfect victimization, and that fully utilizes all the on-disk caches. We then show how well the proposed DIC scheme performs in comparison with the ideal scheme.

6.1 Analytical Model for the Ideal DIC

The idea of the model can be explained as follows: The ideal DIC allows only the shortest intervals between successive streams to be serviced from the caches. Therefore, the length of a cached interval is bounded by an upper value, denoted as L seconds. Hence, the number of concurrent cached streams, N, can be found by counting the number of arriving requests that fall within L-length distances from their preceding streams during one video duration, D. For this purpose, the number of cached requests for each video should be found individually. This number can be estimated through the following process.

- Divide a D-length time window into T L-length time windows, where $T = D/L$.
- Assume that the leading request in each time window is serviced from the disks or the caches. This assumption will be eliminated later.
- Find the total number of subsequent requests in each time window and use it as an initial estimate of the number of concurrent streams.
- Account for the requests that fall in adjacent time windows, but are within L-length distances from each other.

Subsequently, N can be found by adding up the number of concurrent cached streams for each video. Fig. 17 further

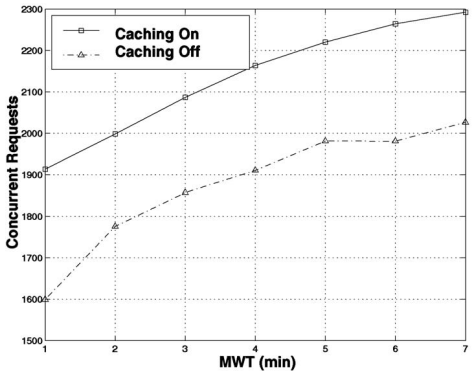


Fig. 14. Effect of maximum waiting time (MWT).

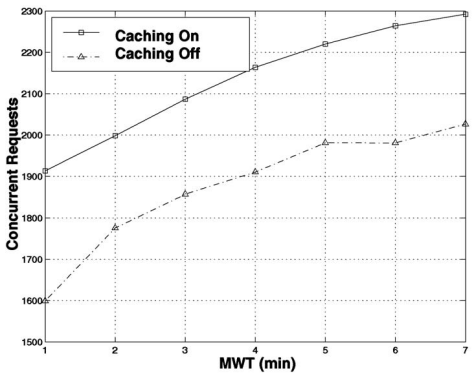


Fig. 16. Effect of skewness parameter (θ) on concurrent requests.

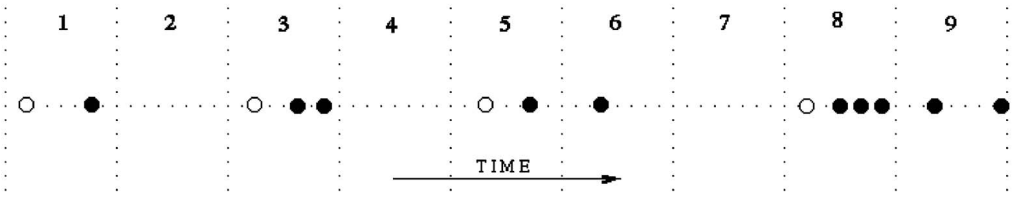


Fig. 17. Clarification of the analytical model.

explains the counting process. The figure shows the arrival of requests for a certain video within nine consecutive L-length time windows. A dark circle represents an arriving stream that can be cached, while a white circle represents an arriving stream that cannot be cached. Observe the effect of the overlap between time windows 5 and 6 and time windows 8 and 9.

Let us now discuss the entire process of the model development. Some of the parameters used in the model are described in Table 3. Because the arrival of requests follows a Poisson Process distribution with parameter λ, the interarrival time to video j is exponentially distributed with a mean $1/\lambda_j$, where $\lambda_j = \lambda \times A_j$, and A_j is the probability of selecting video j. Therefore, the probability of k arrivals for video j within an L-length time window is $P_k^j = \frac{(\lambda_j L)^k}{k!} e^{-\lambda_j L}$. Consequently, the expected number of requests for video j during D seconds that have preceding streams in their L-length time windows, assuming exactly k arrivals in each of these time windows, can be found using the Bernoulli distribution as follows:

$$E[\hat{N}_k^j] = (k-1)\sum_{i=1}^{\lceil T \rceil} i \binom{\lceil T \rceil}{i} P_k^{j^i} (1 - P_k^j)^{\lceil T \rceil - i}. \tag{1}$$

In (1), $k-1$ is the number of cached requests in each of those L-length time windows. The equation is precise when D is a multiple of L. The expected number of arriving requests for video j that fall within the same L-length time windows as their preceding streams is

$$E[\hat{N}^j] = \sum_{k=2}^{\infty} E[\hat{N}_k^j]. \tag{2}$$

We now need to account for the requests arriving in adjacent time windows, but still falling within an L distance from each other. The number of such requests can be found using the Bernoulli distribution. Hence, the expected number of concurrent cached requests for video j is

$$E[N^j] = E[\hat{N}^j] + \sum_{i=1}^{\lceil T \rceil} i \binom{\lceil T \rceil}{i} P_2^{j^i} (1 - P_2^j)^{\lceil T \rceil - i} \tag{3}$$

and the expected number of concurrent cached streams for all videos is

$$E[N] = \sum_{j=1}^{N_v} E[N^j]. \tag{4}$$

In (3), P_2 is the probability of exactly two arrivals within an L-length time window.

To find $E[N]$, we need to know L, which depends on the average cached interval. The average cached interval in seconds for video j can be calculated using the probability density function of the corresponding Poisson process, $f_x(x)$, as follows:

$$\begin{aligned} E[I^j] &= \int_0^L x f_x(x)\, dx = \lambda_j \int_0^L x e^{-\lambda_j x}\, dx \\ &= \frac{1 - (\lambda_j L + 1) e^{-\lambda_j L}}{\lambda_j}. \end{aligned} \tag{5}$$

Finally, the number of concurrent cached streams can be found subject to constraint on the total cache space:

$$\sum_{j=1}^{N_v} E[I^j] \times E[N^j] \times r <= N_d \times S. \tag{6}$$

Let us now reconsider the assumption that the leading request in each L-length time window is serviced. Instead, let us assume that the server services that request with probability f. This probability can be determined by using the average request defection probability, dr, of the server ($f = 1 - dr$). Thus, assuming that the service of a request is independent of the service of the other requests in the same L-length time window, the number of cached requests in each time window is

$$\begin{aligned} N_l &= f(k-1) + (1-f)f(k-2) + \cdots + (1-f)^{k-2} f(1) \\ &= f \sum_{j=1}^{k-1} (1-f)^{j-1} (k-j). \end{aligned} \tag{7}$$

Therefore, $k-1$ in (1) should be replaced with N_l. Because dr is an unknown output parameter, the value of f should be adjusted so that $dr = 1 - f$.

This study validates the analytical model by developing a simulator for the ideal DIC. In the simulator, all the internal disk caches are viewed as one big cache because the ideal DIC utilizes the caches perfectly and ensures that only the shortest intervals are cached at any time. Fig. 18 depicts a comparison between the results of the analytical model and those of simulation for three different systems. We observe that the difference between the analytical model and simulation can be predicted using a simple heuristic. For a wide range of systems, we found that the difference can be estimated as follows: $Error = (0.00934 \times S - 1)N_d - 0.13379 \times S$. Incorporating this error into the analytical model, we obtain *Model+*, which gives a very accurate estimate of the expected number of concurrent cached streams, $E[N]$.

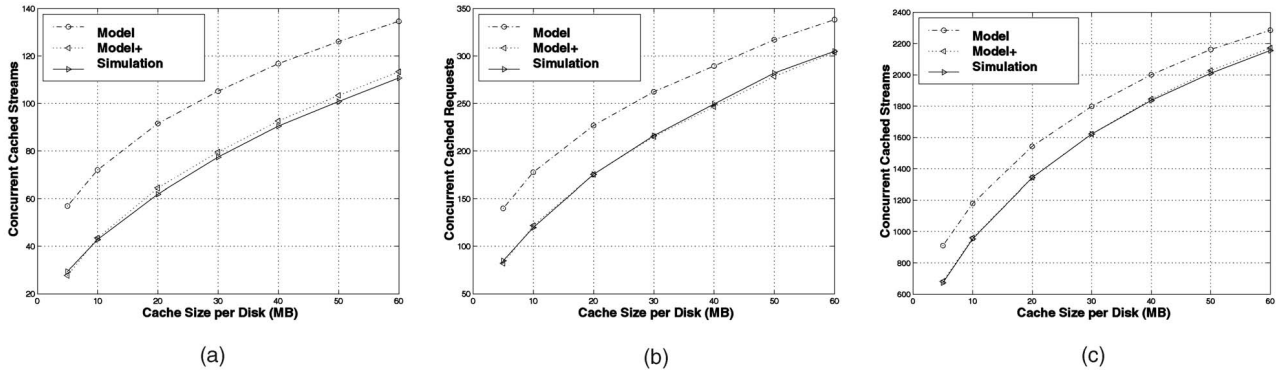


Fig. 18. Comparison between simulation and analytical results ($VR = 0.99$).

6.2 Comparison between the Ideal DIC and the Proposed Algorithm

Fig. 19 compares the performance of the proposed DIC algorithm with that of the ideal DIC for four disk configurations. The results show that the proposed algorithm achieves between 23 percent and 33 percent of the upper limit. The performance gap is primarily because the load is not perfectly balanced among all caches with the proposed algorithm, and the victimization in real DIC is highly constrained. The load balancing problem is due to the distributed nature of the algorithm. These results also indicate that the upper limit on DIC is very high, so there may be plenty of room for improving the DIC algorithm. This limit, however, is somewhat loose because the constrained victimization is a nature of the problem, not of the algorithm. We outline two ways to enhance the performance of the DIC algorithm in the following section.

7 Conclusions and Future Work

In this paper, we have proposed using the NAD architecture to design highly scalable and cost-effective MOD servers. We have used caching, batching, and intelligent scheduling in an *Integrated Resource Sharing* policy to enhance the performance of NAD-based multimedia servers. For caching, we have proposed a *Distributed Interval Caching* (DIC) scheme, which utilizes the on-disk buffers to cache intervals between successive streams. We have also proposed a *Multi-Objective Scheduling* (MOS) scheme, which schedules the waiting requests based on four predefined performance criteria. The integrated policy is highly configurable: It can be tuned in order to make any necessary design trade off and to fit the current server workload.

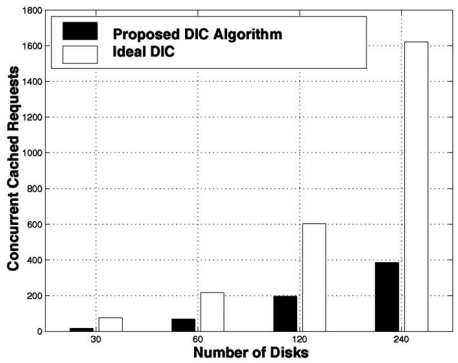


Fig. 19. Ideal DIC versus the proposed algorithm.

We have studied the effectiveness of the integrated policy and the interactions among various parameters through extensive simulation. The results show that the integrated policy can increase the number of concurrent customers that can be serviced while reducing their waiting times and can offer greater benefits for larger servers. In particular, it increases the number of concurrent customers by about 12 percent in a 60-disk to about 17 percent in a 240-disk server. Moreover, it reduces the average customer waiting time by about 8 percent in a 60-disk server to about 40 percent in a 240-disk server. The results also show that the MOS scheme can provide simultaneously an additional 4 percent improvement in the number of concurrent requests and an additional 20 percent to 65 percent improvement in the average request waiting time.

We have also presented an analytical model that estimates the upper bound on the performance of DIC. We have found that the theoretical limit is very high. This indicates that there may still be plenty of room for further improvements, but it is unclear how much of this limit is in fact achievable.

We plan to extend this work in three main directions. First, we will conduct a detailed study of MOS and investigate how its scheduling parameters can be tuned. Second, we will examine using hash functions instead of the simple round-robin scheme to improve the performance of the DIC scheme; efficient hashing can reduce the imbalance in cache utilization. Third, we will evaluate the effectiveness of dividing the disks into partitions compared with merely striping videos across all the disks. In addition to improving the reliability, partitioning should reduce the waiting times before requests find empty slots for service. Reducing such delay should boost performance by shortening the average cached interval.

Acknowledgments

The authors would like to thank the anonymous reviewers for their many helpful comments. This research was supported in part by US National Science Foundation grants CCR-9900701, CCR-0098149, CCR-0208734, and EIA-0202007.

REFERENCES

[1] C.C. Aggarwal, J.L. Wolf, and P.S. Yu, "The Maximum Factor Queue Length Batching Scheme for Video-on-Demand Systems," *IEEE Trans. Computers,* vol. 50, no. 2, pp. 97-110, Feb. 2001.

[2] J. Almeida, D. Eager, and M. Vernon, "A Hybrid Caching Strategy for Streaming Media Files," *Proc. SPIE/ACM Conf. Multimedia Computing and Networking,* Jan. 2001.

[3] D.P. Anderson, Y. Osawa, and R. Govindan, "A File System for Continuous Media," *ACM Trans. Computer Systems,* vol. 10, no. 4, pp. 311-337, Nov. 1992.

[4] W.J. Bolosky, M.B. Jones, and R.F. Rashid, "The Tiger Video File Server," *Proc. Workshop Network and Operating System Support for Digital Audio and Video,* pp. 97-104, Apr. 1996.

[5] R. Burns, R. Rees, and D. Long, "Safe Caching in a Distributed File System for Network Attached Storage," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* pp. 155-162, May 2000.

[6] Y. Cai and K.A. Hua, "An Efficient Bandwidth-Sharing Technique for True Video on Demand Systems," *Proc. ACM Int'l Conf. Multimedia,* pp. 211-214, Oct. 1999.

[7] A.L. Chervenak, "Tertiary Storage: An Evaluation of New Applications," PhD thesis, Univ. of California Berkeley, Univ. of California Berkeley Technical Report UDB/CSD 94/847, Dec. 1994.

[8] A.L. Chervenak, D.A. Patterson, and R.H. Katz, "Choosing the Best Storage Systems for Video Service," *Proc. ACM Conf. Multimedia,* pp. 109-119, Nov. 1995.

[9] C. Chou, L. Golubchik, and J.C.S. Lui, "Striping Doesn't Scale: How to Achieve Scalability for Continuous Media Servers with Replication," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS),* pp. 64-71, Apr. 2000.

[10] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching," *Proc. ACM Conf. Multimedia,* pp. 391-398, Oct. 1994.

[11] A. Dan, D.M. Dias, R. Mukherjee, D. Sitaram, and R. Tewari, "Buffering and Caching in Large-Scale Video Servers," *Digest of Papers, IEEE Int'l Computer Conf.,* pp. 217-225, Mar. 1995.

[12] A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley, "Channel Allocation under Batching and VCR Control in Movie-on-Demand Servers," *J. Parallel and Distributed Computing,* vol. 30, no. 2, pp. 168-179, Nov. 1995.

[13] A. Dan and D. Sitaram, "A Generalized Interval Caching Policy for Mixed Interactive and Long Video Workloads," *Proc. Multimedia Computing and Networking Conf. (MMCN),* pp. 344-351, Jan. 1996.

[14] R. Flynn and W. Tetzlaff, "Disk Striping and Block Replication Algorithms for Video File Servers," *Proc. Int'l Conf. Multimedia Computing and Systems,* pp. 590-597, June 1996.

[15] G. Ganger, B. Worthington, and Y. Patt, *The DiskSim Simulation Environment, Version 2, Reference Manual, CMU.* Dec. 1999.

[16] G. Ganger and J. Schindler Database of Validated Disk Parameters for DiskSim, 2004. http://www.ece.cmu.edu/ganger/disksim/diskspecs.html.

[17] G. Gibson et al., "Filesystems for Network-Attached Secure Disks," Technical Report CMU-CS-97-118, Computer Science Dept., Carnegie Mellon Univ., July 1997.

[18] G. Gibson et al., "A Cost-Effective, High-Bandwidth Storage Architecture," *Proc. Conf. Architecture Support for Programming Languages and Operating Systems,* pp. 92-103, Oct. 1998.

[19] L. Golubchik, J.C.S. Lui, and R. Muntz, "Reducing I/O Demand in Video-on-Demand Storage Servers," *Proc. ACM SIGMETRICS Conf. Measurements and Modeling of Computer Systems,* pp. 25-36, May 1995.

[20] R. Haskin, "The Shark Continuous Media File Server," *Proc. IEEE 1993 Spring COMPCON,* pp. 12-17, Feb. 1993.

[21] K.A. Hua and S. Sheu, "Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand System," *Proc. ACM Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '97),* pp. 89-100, Sept. 1997.

[22] K.A. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast Technique for True Video-on-Demand Services," *Proc. ACM Conf. Multimedia,* pp. 191-200, Sept. 1998.

[23] L. Juhn and L. Tseng, "Harmonic Broadcasting for Video-on-Demand Service," *IEEE Trans. Broadcasting,* vol. 43, no. 3, pp. 268-271, Sept. 1997.

[24] K. Keeton, D.A. Patterson, and J.M. Hellerstein, "The Intelligent Disk (IDISK): A Revolutionary Approach to Database Computing Infrastructure," white paper, Univ. of California Berkeley, May 1998.

[25] G. Ma, A. Khaleel, and N. Reddy, "Performance Evaluation of Storage Systems Based on Network-Attached Disks," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 9, pp. 956-968, Sept. 2000.

[26] J. Nieh and M.S. Lam, "The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications," *Proc. ACM Symp. Operating Systems Principles,* pp. 184-197, Oct. 1997.

[27] P.V. Rangan and H.M. Vin, "Designing File Systems for Digital Video and Audio," *Proc. ACM Symp. Operating Systems Principles,* pp. 81-94, Oct. 1991.

[28] A.L.N. Reddy and J. Wyllie, "Disk Scheduling in a Multimedia I/O System," *Proc. ACM Conf. Multimedia,* pp. 225-233, Aug. 1993.

[29] N.J. Sarhan and C.R. Das, "Adaptive Block Rearrangement Algorithms for Video-on-Demand Servers," *Proc. Int'l Conf. Parallel Processing,* pp. 452-459, Sept. 2001.

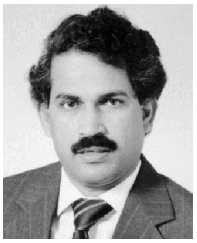
[30] N.J. Sarhan and C.R. Das, "A Simulation-Based Analysis of Scheduling Policies for Multimedia Servers," *Proc. Ann. Simulation Symp.,* pp. 183-190, Mar.-Apr. 2003.

[31] N.J. Sarhan and C.R. Das, "An Integrated Resource Sharing Policy for Multimedia Storage Servers Based on Network-Attached Disks," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS),* pp. 136-143, May 2003.

[32] P. Shenoy and V. Harric, "Efficient Striping Techniques for Variable Bit Rate Continuous Media File Servers," *Performance Evaluation J.,* vol. 38, no. 2, pp. 175-199, Dec. 1999.

[33] R. Tewari, H.M. Vin, A. Dan, and D. Sitaram, "Resource-Based Caching for Web Servers," *Proc. SPIE Conf. Multimedia Computing and Networking,* pp. 191-204, Jan. 1998.

Nabil J. Sarhan received the BS degree in electrical engineering from Jordan University of Science and Technology, Irbid, Jordan, in 1995. He received the MS and the PhD degrees in computer science and engineering from The Pennsylvania State University in 2003. He currently works as an assistant professor of electrical and computer engineering at Wayne State University. His main research areas are multimedia systems and networking, I/O, multiprocessing, computer architecture, and performance evaluation.



Chita R. Das received the MSc degree in electrical engineering from the Regional Engineering College, Rourkela, India, in 1981, and the PhD degree in computer science from the Center for Advanced Computer Studies, University of Louisiana, Lafayette, in 1986. Since 1986, he has worked at The Pennsylvania State University, where he is currently a professor in the Department of Computer Science and Engineering. His main areas of interest are parallel and distributed computer architectures, cluster systems, communication networks, resource management in parallel systems, mobile computing, performance evaluation, and fault-tolerant computing. He has published extensively in these areas in all major international journals and conference proceedings. He was an editor of the *IEEE Transactions on Parallel and Distributed Systems* and is currently serving as an editor of the *IEEE Transactions on Computers*. Dr. Das is a fellow of the IEEE, and a member of the ACM and the IEEE Computer Society.