Caching and Scheduling Techniques for Multimedia Storage Servers Based on Network-Attached Disks[1]

Nabil J. Sarhan *Chita R. Das*

Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802
Phone: (814) 865-0194
E-mail: {sarhan,das}@cse.psu.edu

Abstract

Multimedia-on-demand (MOD) has 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 multimedia-on-demand servers. In order to ensure enhanced performance, we propose a scheme called *distributed interval caching* (DIC) that utilizes the on-disk buffers for caching intervals between successive streams. We also propose another scheme called *multi-objective scheduling* (MOS) that increases the degrees of resource sharing by scheduling 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. The results also indicate that the performance benefits vary considerably with several architectural, system workload, and scheduling parameters. Furthermore, we develop an analytical model for an ideal DIC scheme to estimate the performance limits.

Keywords: batching, distributed interval caching, multimedia-on-demand (MOD), network-attached disks, scheduling, video-on-demand (VOD).

1 Introduction

Recent advances in storage and communication technologies have spurred a strong interest in *multimedia-on-demand* (MOD) servers. The delivery of rich media contents (video, audio, images, and text) is

[1]This research was supported in part by NSF grants CCR-9900701, CCR-0098149, CCR-0208734, and EIA-0202007.

projected to be an essential component of the overall e-business strategies for many companies and institutions [12]. MOD servers, which subsume *video-on-demand* (VOD) servers, eliminate the shortcomings of broadcast-based systems like cable TV by providing user-customized services.

The design of MOD servers faces significant challenges to efficiently store, access, and distribute media contents to a large number of concurrent clients. The main performance metric of MOD servers is the number of concurrent requests that can be serviced, while maintaining a "reasonable" quality of service (QoS). Unfortunately, this number is highly constrained by the requirements of the real-time playback and the high transfer rates. Therefore, a wide spectrum of techniques has been developed to enhance the performance of MOD servers. These techniques include *efficient striping* [30, 6], *replication* [13, 6, 7], *disk scheduling* [28, 25], *block allocation* [27, 13], *resource sharing* [8, 17, 32, 1, 20, 21], and *admission control* [22, 33].

Current MOD servers, such as Tiger Shark [18, 19] and Tiger Video [3], are based on the conventional fileserver architecture (also known as the *Server-Attached Disk Architecture*), where all the data flowing from disks to clients pass through the server. The required *store-and-forward* copying imposes unnecessary burden on these servers and severely limits their scalability.

Recently, the *network-attached disks* (NAD) architecture [16, 23, 24] has emerged as a cost-effective solution to design scalable storage servers. In this architecture, disks are connected to the network so that data can be transferred directly from disks to clients. The NAD architecture offers four major advantages over the traditional architecture. First, it scales inherently with storage capacity by 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 power and cache (buffer) effectively. The scalability and performance issues of the NAD architecture are discussed in [16, 24].

This paper investigates the design of scalable MOD servers based on the NAD architecture[2]. In particular, we propose an integrated resource sharing policy, which boosts the performance of these servers. The principal MOD application of interest in this study is video-on-demand (VOD).

Resource sharing strategies improve performance by exploiting the high locality of reference in video

[2] While writing this paper, we learned through personal communication that *EMC* is working on the design of media servers based on the NAD architecture.

access patterns. These techniques can be classified into five main categories: *batching* [8, 10, 32, 1], *patching* [20, 29], *piggy-backing* [17], *broadcasting* [21, 26, 4], and *buffer management* through *interval caching* [9, 11]. With batching, requests to same movies are accumulated over a time window and serviced together by utilizing the multicast facility. Batching, therefore, off-loads the storage subsystem and uses efficiently server bandwidth and network resources. In contrast with batching, patching expands the multicast tree dynamically to include new requests. When a request arrives, the server initiates a patching stream to service the trail of the requested video. Meanwhile, the server transmits the multicast data, and the client caches them. Patching reduces the request waiting time, but it requires additional bandwidth and buffer space at the client. Like patching, piggy-backing services a request almost immediately, but unlike patching, the playback rate is adjusted so that the request catches up with a preceding stream, resulting in a lower-quality initial presentation. Broadcasting techniques divide each movie into multiple segments and broadcast each segment periodically. Thus, the client has to wait until the next broadcast of the first segment. Because multiple segments of the same movie are transmitted concurrently, broadcasting requires relatively very high bandwidth at the client. Interval caching caches intervals between successive streams in the server. This technique does not sacrifice the quality of service, does not lengthen the waiting time, and does not expect much from the client. Moreover, it can be applied with other techniques for better resource sharing, and it has become more cost-effective with the diminishing prices of DRAMs. It has also been used efficiently to cache continuous media content in proxy servers [31, 2].

In this paper, we propose an integrated policy that employs caching, limited degrees of batching, and intelligent scheduling to enhance the performance of NAD-based media servers. Previous studies, however, examined either caching or batching, but not both. We propose a caching 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 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 of the NADs for caching intervals between successive streams. DIC requires low-level controllability of caches, which can be entertained by the growing on-disk intelligence. We also propose a scheduling scheme, called *multi-objective scheduling* (MOS), which selects requests intelligently based on four predefined criteria. MOS favors requests with lighter demands and/or those that increase the level of batching. The integrated policy combines the benefits

of caching, batching, and intelligent scheduling and allows the administrator to control the tolerable initial delay and implementation overhead and maximize performance based on the actual server workload. One of the unique features of this paper is studying VOD servers at the disk level and considering many design aspects such as scheduling, admission control, batching, and caching. Analyzing the integrated policy, rather than only the individual policies, helps in understanding various tradeoffs and making appropriate design decisions.

We study the effectiveness of the integrated policy and the interaction among various system and workload parameters through extensive simulation. We consider two main objective functions: the average number of concurrent requests that can be serviced and the average request waiting time (i.e. initial viewing delay). The results show that the integrated policy, unlike most previous policies, can improve simultaneously both these performance metrics. The improvement achieved by the integrated policy in the number of concurrent requests approximately ranges from 12% to 17% as the number of disks increases from 60 to 240, using only 30 MB cache per disk. The integrated policy also reduces the average request waiting time by about 8% in a 60-disk server to about 40% in a 240-disk server. MOS can provide an additional 4% improvement in the number of concurrent streams and an additional 20% - 65% improvement in the average request waiting time, with respect to the best performer of existing scheduling policies. The results also show that FCFS is the worst performer among all investigated scheduling policies. These results differ from previous studies primarily because we consider caching, whose performance degrades substantially by scheduling older requests.

Finally, we present an analytical model for an ideal DIC scheme to examine whether the performance of DIC can be improved further. The results indicate that the maximum achievable performance with DIC is very high, so 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 the analytical model for an ideal DIC. Finally, we draw conclusions and outline a future research in the last section.

2 The Distributed Interval Caching (DIC) Scheme

Movie rental patterns suggest that accesses to movies are highly localized, with only a small number of movies receiving most of the hits [5]. By exploiting this high locality of reference, caching can signif-

icantly improve the performance of multimedia servers. Traditional caching techniques use temporal and spatial locality of reference to bring selected objects to the cache. These techniques are well-suited for applications that consist of relatively small objects; however, in a multimedia server, where objects are usually very large, caching the entire most commonly used objects is not cost-performance effective.

A caching scheme for multimedia servers, called *interval caching*, was proposed in [9]. With this scheme, each playback request is paired (if possible) with an immediately preceding request for the same movie that is currently being serviced (either from disk or 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. When the server accepts a request for service from cache, it starts to cache the data of the preceding stream while it retrieves and transfers them to the client. The time during which the following stream is serviced from disks is called the *catchup time*. Interval caching uses the available cache (memory) space efficiently by victimizing longer intervals for caching shorter ones.

When the request arrival rate is fairly high, interval caching can achieve significant performance benefits, even in servers with modest cache sizes. The high request rate coupled with the large skewness in movie access reduce the average cached interval, thereby improving the effectiveness of caching.

With interval caching, data are cached in the main memory of the fileserver. In the NAD architecture, the fileserver (filemanager) is connected to the network as well as the NADs. Thus, using such caching in a NAD-based server involves extra network activity in the process of bringing the *to-be-cached* data from disks to the filemanager. More importantly, it negates the whole purpose of the NAD architecture (i.e., eliminating the fileserver as a bottleneck).

The DIC scheme proposed here extends interval caching and adapts it to the NAD architecture. In this scheme, 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, scheduling, and victimization. Before we explain the scheme, let us discuss briefly how a multimedia server operates. In a multimedia server, movies are striped across all or a collection of disks. The server handles multiple clients by proceeding in periodic rounds. During each round, a fixed duration of media, called interleaving unit (IU), is retrieved for each client from a disk. For simplicity, we assume that movies are striped across all N_d disks in a round-robin (RR) fashion.

A mapping function, given by $Disk\_map(m) = m \ mod \ N_d$, is used to map the first IU of movie m to a disk. This mapping helps to enhance the cache utilization and to balance the I/O load among all disks.

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

Let us now discuss what happens after a request is accepted for service from cache. Let us assume that during round i, a new request calls for the playback of a movie 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 cache, 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 cache until the completion of the movie.

Figure 1 further explains the scheme for an 8-disk system. It shows what happens after *S2* (the following stream of movie 2) is paired up with *S1* (the preceding stream of the same movie), 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 the arrow shows whether the stream is being serviced from cache or disk, respectively.

Striping movies in a round-robin fashion simplifies scheduling. 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 disks if there is adequate I/O bandwidth in disk f_disk. This condition guarantees sufficient I/O bandwidth for the request in all subsequent rounds. Deciding whether a request can be serviced from cache is a bit trickier 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

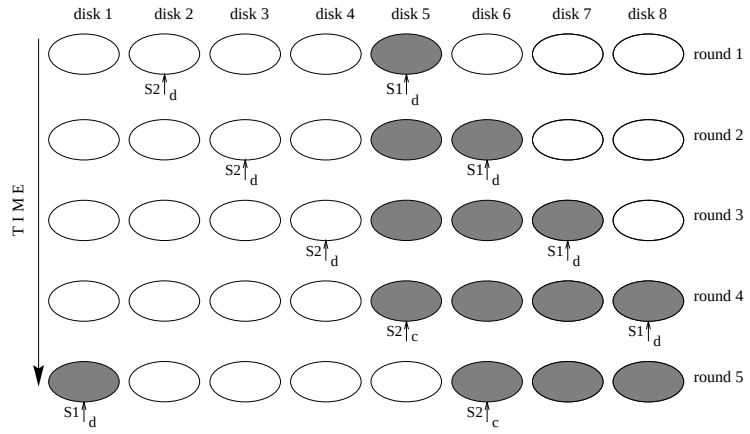


Figure 1: Explaining Distributed Interval Caching

between disk f_disk and disk p_disk, inclusively. (To clarify the word *between* in this context, these disks are f_disk, $(f\_disk + 1)$ mod N, $(f\_disk + 2)$ mod N, ..., and p_disk.) Note that in Figure 1, the availability of adequate cache space in disks 2, 3, 4, and 5 in round 1 guarantees that the interval between the two streams can be cached in all subsequent rounds. 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.

The server can apply victimization only if all the following four conditions are met. First, the potential victim can be serviced from disks. Second, the new request can be serviced from disks during catchup. Third, a portion of the cached interval of the potential victim is currently cached in disk p_disk. Fourth, there is a considerable difference between the interval length of the new request and that of the potential victim. For the last condition, we introduce a parameter VR (denotes Victimization Ratio), which must be tuned for a particular system to maximize the number of cached streams. VR can be defined as the largest ratio of the required interval length to that of the potential victim that is necessary for applying victimization.

We present the algorithm in a C++-like syntax in Figure 2. In the algorithm, 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 *cached-interval queue* (CIQ) and is used to determine whether victimization can be applied. The intervals in the cached-interval queue are examined from top to bottom for possible

victimization. In the algorithm, we assume that the disks have sufficient network bandwidth to handle their I/O rates.

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

Figure 2: The Distributed Interval Caching Algorithm

3 The Multi-Objective Scheduling (MOS) Scheme

A VOD server maintains a waiting queue for every movie, routes incoming requests to their corresponding queues, and applies a scheduling policy to select an appropriate queue whenever it has an

available *channel*. A channel is a set of resources needed to deliver a multimedia stream. All requests in the selected queue can be serviced together using only one channel. The two main performance metrics of scheduling policies are the average number of concurrent requests that can be serviced and the average request waiting time. The first metric is the most important because it corresponds to the overall customer defection (reneging) probability and server throughput. Unfairness is another metric that quantifies the bias of a policy against cold (i.e., unpopular) movies.

Scheduling policies for VOD servers include *First Come First Serve* (FCFS) [8], *Maximum Queue Length* (MQL) [8], *Maximum Factored Queue Length* (MFQL) [1], and *Group-Guaranteed Server Capacity* (GGSC) [32]. FCFS favors the queue with the oldest request while MQL favors the longest queue. Thus, FCFS is fair whereas MQL is biased against cold movies. MFQL attempts to minimize the average request waiting time at full load, so it selects the queue with the *largest factored queue length*. The factored queue length of a queue is defined as its length divided by the square root of the relative access frequency of its corresponding movie. The results in [1] show that MFQL serves as a compromise between FCFS and MQL. In particular, the average defection probability with MFQL is better than MQL but worse than FCFS. In addition, the average request waiting time with MFQL is better than FCFS but worse than MQL. Because this study aims at developing a scheduling scheme that can outperform both FCFS and MQL in both performance metrics, we will not consider MFQL. GGSC preassigns server channel capacity to groups of requests in order to optimize the average request waiting time. It was shown in [32] that this policy does not perform well for high-end servers. We will not consider this policy either since this study targets primarily high-end servers.

We propose a *multi-objective scheduling* (MOS) scheme, which combines the advantages of the following four scheduling criteria. The first two correspond to MQL and FCFS, and we propose the other two.

- **Criterion 1**: Favor the longest queue. This criterion can be captured by the optimization function $F_b(i) = B_i$, where i is the movie number, and B_i is the batch size (i.e., number of waiting requests) of movie i. This criterion selects the movie with the largest F_b.
- **Criterion 2**: Favor the queue with the oldest request. The optimization function in this case is $F_t(i) = T_i$, where T_i is the longest request waiting time for movie i.
- **Criterion 3**: Favor the queue whose requests require the shortest cached interval. This criterion

is important in the NAD environment because of the highly constrained victimization. So, it is important to select the shortest intervals for caching rather than to rely entirely on victimization (as done in traditional interval caching). The optimization function is $F_n(i) = 1/I_i$, where I_i is the required interval to cache the requests for movie i.

- **Criterion 4**: Favor the queue of the most popular movie among all non-empty queues. This criterion is biased towards popular movies, whose requests are likely to have closer preceding and following streams that can be paired up with using a smaller cache space. Moreover, requests have a higher tendency to accumulate in the waiting queue of a more popular movie. The optimization function here is $F_m(i) = 1/i$. (According to Zipf's distribution, which we will discuss in Subsection 5.2, the popularity of a movie decreases as its number increases.)

For the MOS scheme, we introduce 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 movie number among those with waiting requests. The weights should be tuned for performance. In this paper, we focus on the main performance metrics (number of concurrent requests and request waiting time), although MOS provides the flexibility in optimizing other performance metrics.

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). Figure 4 shows the overall architecture and how the integrated policy works during one round[3]. $Q1$, $Q2$,..., and Qm are the movie waiting queues, and CIQ is the cached-interval queue. 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* schedules requests in decreasing order of F and implements the rest of the integrated policy. The dashed lines show how a client requests the playback of movie 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.

[3]The clients can also be connected to the local network.

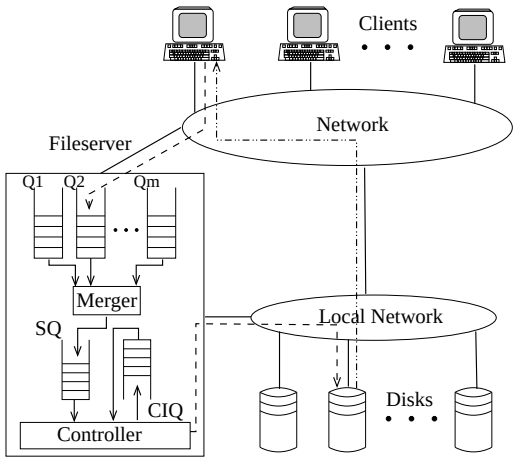


Figure 3: The Overall Architecture of the Integrated Policy

To obtain the algorithm of the integrated policy, we enhance the DIC algorithm by two refinements. The *first refinement* limits the number of requests examined in SQ. Thus, we introduce the parameter RCR (Request Check Ratio), which specifies the fraction of the requests in the scheduling queue that will be considered for scheduling. The high degree of filtering, attained from reducing RCR, may lead to the selection of only the "best" requests that are likely to boost performance while reducing the implementation overhead. The dark side, however, is introducing a longer delay in servicing some requests, which in turn may lead to lengthening the average cached interval and denying service to a larger number of requests. Therefore, this parameter should be selected as a reasonable tradeoff. This refinement requires changing line 1 of the DIC algorithm in Figure 2 from

for (each waiting request)

to

for (each of the top $(RCR \times 100)\%$ requests in the scheduling queue starting with the topmost)

The *second refinement* comes from introducing the parameter DSR (Delay Service Ratio). With this refinement, if the required interval cannot be cached, the refined algorithm does not directly attempt to service the request from disks. Instead, it delays the service with the hope that an adequate cache space for the interval will be found in the following rounds. This also helps to increase batching. 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 larger value of DSR increases both batching and caching, but it also increases the average request

waiting time. Therefore, this ratio should be selected as a compromise. This refinement requires changing line 33 of the DIC algorithm in Figure 2 from

Accept the waiting request for service from disks;

to

if (request waiting time $> DSR \times MWT$) Accept the waiting request for service from disks;

We summarize the main parameters in Table 1 for ease of reference in the subsequent analysis.

Table 1: Major Parameters

Input Parameters	**Input Parameters (Cont.)**
u: Size Interleaving Unit (s)	r: Movie Data Rate (MB/s)
N_d: Number of Disks	MWT: Max Req. Wait. Time
S: Cache Size per Disk (MB)	**Policy Parameters**
N_m: Number of Movies	VR: Victimization Ratio
D: Movie Length (s)	RCR: Request Check Ratio
λ: Req. Arrival Rate ($1/s$)	DSR: Delay Service Ratio
θ: Skewness Parameter	w_b, w_n, w_t, w_m: Scheduling P.

5 Performance Evaluation

We analyze the effectiveness of the proposed techniques through extensive simulation. In contrast with existing VOD servers, such as Tiger Shark [18, 19] and Tiger Video [3], the disks in the target server are connected directly to the network, and the on-disk memories (buffers) are used for caching intervals between successive streams. We start our discussion with the simulation environment and workload characteristic, and then we present the main results.

5.1 Simulation Platform

We built a simulator for NAD-based multimedia servers to study the effectiveness of the proposed policies and analyze the impact of and interaction among various system and workload parameters. The simulator takes numerous input parameters, including those in Table 1, and 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. The main output parameters of the simulator are the average number of concurrent requests that can be serviced, the overall customer defection rate, and the average request waiting time.

We evaluated the disk performance parameters using appropriate streaming media workload de-

scribed in the following subsection. We experimented primarily with *Quantum Atlas III*. Table 2 shows its main parameters. We used *DiskSim* [14] to find the disk performance parameters required by our simulator. In this process, we fed DiskSim with the synthetic workloads and with validated disk parameters obtained from [15]. To ensure accurate simulation, the seek time for each seek distance is based on actual measurements. Table 3 shows the main disk performance parameters, which are produced by DiskSim.

Table 2: Main Disk Parameters

Storage Capacity (GB)	9.1
Rotation Speed (rpm)	7200
Number of Cylinders	8,057
Number of Surfaces	10
Full Strobe Seek Time (ms)	15.36
Head Switch Time (ms)	0.999
Internal Transfer Rate (Megabits/s)	110 to 180

Table 3: Estimated Values of Disk Performance Parameters (By Simulation)

Mean Seek Time	7.778 ms
Mean Rotational Time	2.431 ms
Mean Transfer Rate	4.217 MB/s

5.2 Workload Characteristics

Like most prior studies, we assume that the arrival of the requests to the multimedia server follows a Poisson Process with an average arrival rate λ. Hence, the inter-arrival time is exponentially distributed with a mean $T = 1/\lambda$. We also assume that movies are stored using the MPEG-II compression standard with a data rate (r) of 3 Megabits per second, but we will also discuss the impact of changing this parameter. We also assume as in previous works that the accesses to movies follow Zipf's distribution [5]. With this distribution, the probability of choosing the n^{th} most popular of M movies is $C/n^{1-\theta}$ with a parameter θ and a normalized constant C. The parameter θ controls the skewness of movie access. Note that the skewness reaches its peak when $\theta = 0$, and that the access becomes uniformly distributed when $\theta = 1$. Moreover, we assume that customers can wait until a predefined maximum time. (We did not observe any considerable qualitative change in the results when we applied an exponential waiting tolerance.) Furthermore, we assume that movies are striped across all disks on 1/2-second intervals [6]. This assumption, however, can be relaxed to stripe

across any number of disks. The interval size should be chosen based on a reasonable tradeoff. A study for determining interval size for multimedia file servers was presented in [30]. For simplicity, VCR-like operations are not considered. In [10], it was shown that these interactive operations can be supported by allocating contingency channels. To keep the discussion focused, we assume that the network bandwidth for each disk is adequate to support the disk data rate (I/O bandwidth).

5.3 Simulation Results

This study investigates the impact of most input, workload, and policy parameters. We vary the number of disks in the server from 30 to 240, and we vary the number of movies accordingly to keep all disks nearly full. We investigate only a small range of the disk cache size (0 - 40 MB) because of the limited disk power budget [23]. The request arrival rate, λ, is usually selected such that the customer defection rate is about 10% when the cache size per disk is 30 MB. We also study the impact of MWT and θ. Table 4 shows the default values of the main parameters used in this section.

Table 4: Default Parameter Values

$N_d = 120$, $S = 30$ MB, $N_m = 480$, $r = 3$ Megabits/s
$D = 90$ minutes, $\lambda = 0.435$, $\theta = 0$, $MWT = 3$ minutes
$VR = 0.7$, $RCR = 0.5$, $DSR = 0.1$, $w_b = w_n = w_m = 0$, $w_t = 1$

In the evaluation of the effectiveness of the integrated policy, we study two variations: one is when the *policy is ON* (applied), and the other is when the *policy is OFF* (not applied). When the policy is OFF, caching is disabled, but both batching and scheduling are used, and the values of the workload, input parameters (including MWT), and scheduling parameters are same to those when the policy is ON to ensure a fair comparison. We also study the performance of the MOS scheme when DIC is enabled by varying only the values of the scheduling parameters.

The simulation results are obtained using a steady state analysis with 95% confidence interval. Only a limited set of the results are presented.

5.3.1 Effectiveness of the Integrated Policy

In this subsection, we demonstrate 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. Figure 4 plots the number of concurrent requests that can be serviced (N_{conc}) versus the cache size per disk and the number of disks.

In this figure (and all others), the integrated policy is turned off when the cache size is 0. As expected, N_{conc} increases with the number of disks and the cache size, while the number of disks plays a more significant role. Next, we fix the number of disks (at 120) and analyze the impact of the cache size per disk on N_{conc} and the customer defection rate. Figure 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. Figure 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 system can handle. Increasing λ shortens the intervals between successive streams, thereby allowing the server to service more requests from cache. In addition, as λ increases, batching improves since more requests accumulate in the waiting queues.

Figure 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. We observe that the improvement with the integrated policy in N_{conc} approximately ranges from 6% to 17% as the number of disks increases from 30 to 240. We also observe that the average contribution of caching to the total improvement is around 43.7% while the rest is due to batching. 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.

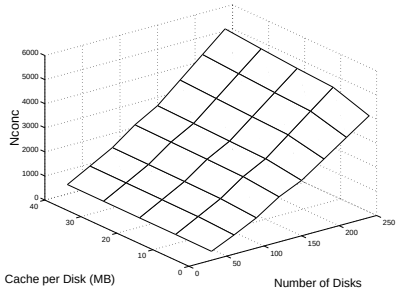


Figure 4: Effect of Number of Disks and Cache Size

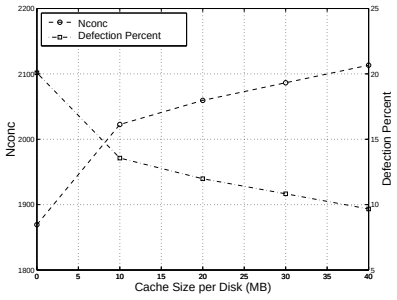


Figure 5: Effect of Cache Size

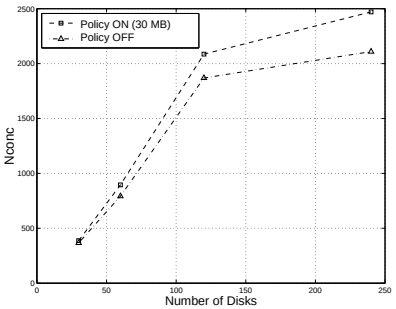


Figure 6: Effect of Number of Disks

Figure 8 depicts N_{conc} versus the average request inter-arrival time ($1/\lambda$). These results show that

the N_{conc} increases with λ. As we stated earlier, this is due to the increase in the average batch size and the number of cached requests. Note that the 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 rate is zero).

Figure 9 illustrates the effects of the number of disks and the cache size per disk on the average request waiting time (T_{wait}). The results show that when the policy is OFF (i.e., *cache size* = 0), T_{wait} increases by a factor of two as the number of disks varies from 30 to 240. This is because movies are striped across all disks, so requests wait longer to find empty slots for scheduling. With the integrated policy, however, T_{wait} decreases since caching services requests as soon as possible. The results also show that caching reduces T_{wait} better in larger-scale servers. Namely, the reduction in T_{wait} approximately ranges from 4% to 40% as the number of disks varies from 30 to 240. T_{wait}, however, remains more than 1.2 minutes, which is 41% of MWT (3 minutes). Such large T_{wait} implies that some customers defected just because of the long delay. We thus believe that dividing disks into partitions can enhance the performance by reducing T_{wait}.

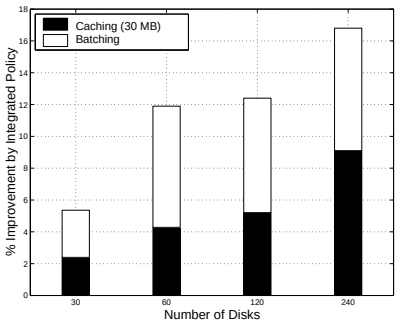


Figure 7: Contributions to Improvements ($VR = 0.99$)

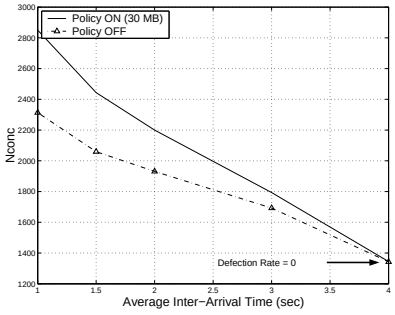


Figure 8: Effect of Average Inter-Arrival Time

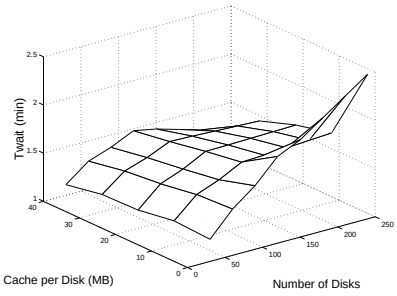
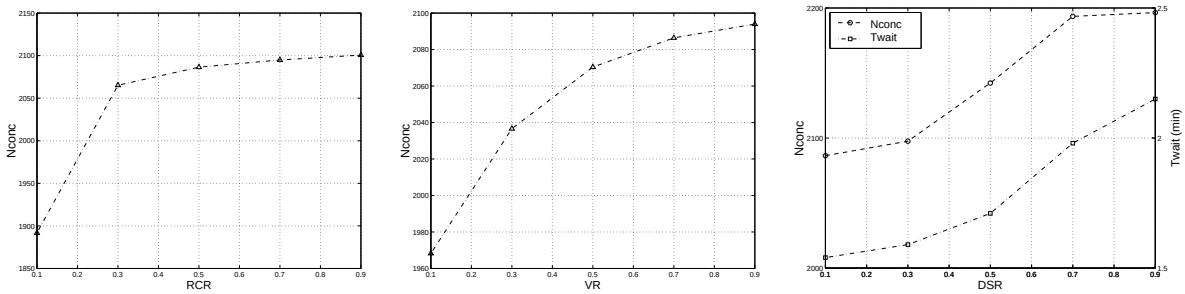


Figure 9: Effect of Number of Disks and Cache Size on T_{wait}

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

Let us now discuss how RCR, VR, and DSR can be tuned to optimize performance. Figure 10(a) plots N_{conc} 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. Figure 10(b) shows the effect of VR on performance. This figure indicates that designers should choose the highest

possible value for VR to enhance performance through better victimization.



(a) RCR (b) VR (c) DSR

Figure 10: Effects of the Tunable Parameters

The impact of DSR on N_{conc} and T_{wait} is shown in Figure 10(c). As expected, increasing DSR has a positive effect on N_{conc} and a negative effect on T_{wait}. Thus, this parameter should be selected as a tradeoff, based on the design objectives. Note that increasing DSR from 0.1 to 0.9 improves performance by more than 5.3%, but it increases T_{wait} by about 40%. If customers are not very sensitive to longer waiting times, then the performance of the server can be enhanced by using a larger value of DSR.

5.3.3 Effects of the Scheduling Parameters

Next, we discuss the impact of the scheduling parameters. We have investigated numerous selections of the scheduling parameters and have observed that the performance of the server is very sensitive to their values. Here, we compare the performance of the four scheduling criteria (discussed in Section 3) and two special cases of the MOS scheme: *MOS 1* and *MOS 2*. In *MOS 1*, the scheduling parameters are set as follows: $w_b = 1$, $w_t = 10^{-4}$, $w_n = 10^{-5}$, and $w_m = 0$. In *MOS 2*, these parameters are set as follows: $w_b = 1$, $w_t = 10^{-5}$, $w_n = 10^{-6}$, and $w_m = 10^{-4}$. The idea behind these two selections is to give the largest weight to the batch size, while allowing the other criteria to influence the scheduling decisions only when the batch size is the same in more than one waiting queue.

Figures 11 and 12 plot the percentage improvements achieved by the different criteria with respect to *Criterion 1* (MQL). The figures show the improvements in N_{conc} and T_{wait}, respectively, for three settings of λ and RCR. The results of *Criterion 2* (FCFS) are not shown in the figure because of

its poor performance: it performs 25% - 40% worse than *Criterion 1* in terms of N_{conc} and 120% - 150% worse than it in T_{wait}. These results differ from previous studies primarily because we consider caching, whose performance degrades substantially by scheduling older requests. N_{conc} with *Criterion 2* is relatively low because this criterion utilizes neither batching nor caching, and T_{wait} is relatively long because it favors older requests. *Criterion 3* performs relatively well in terms of T_{wait} because it favors newer requests. Similarly, T_{wait} is longer with *MOS 1* than that with *MOS 2* because *MOS 1* has a larger weight w_t. In all the three settings (of λ and RCR), *Criterion 3* and *MOS 2* reduce the T_{wait} better than *Criterion 1*, and *MOS 1* outperforms *Criterion 1* in terms of N_{conc}.

The main results can be summarized as follows. The values of the scheduling parameters should be based on the values of λ and RCR. With a careful tuning of these parameters, the MOS scheme can perform better than *Criterion 1* in terms of both N_{conc} and T_{wait}. Both performance metrics (N_{conc} and T_{wait}) can be improved with a very high value of w_b and a very small value of w_t. We leave the optimal selection of the scheduling parameters for a future study.

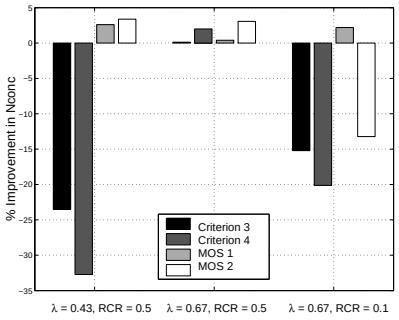


Figure 11: Improvements in N_{conc}

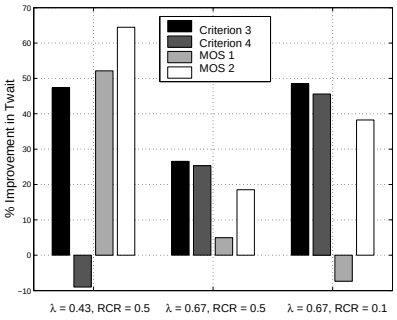


Figure 12: Improvements in T_{wait}

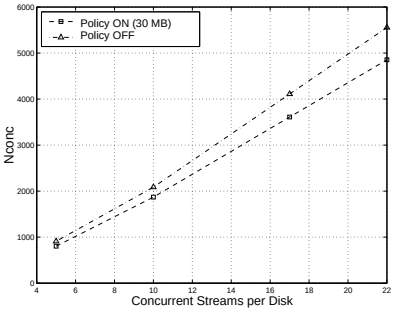


Figure 13: Effect of Disk

5.3.4 Effects of Other Parameters

Let us now discuss the effects of disk performance, the movie data rate, the skewness in access patterns, and the maximum request waiting time (MWT). Figure 13 shows the effect of disk performance (measured as the number of concurrent requests 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 policy ranges between 12% and 14%.

Figure 14 shows the impact of the movie data rate. The number of movies, here, is adjusted

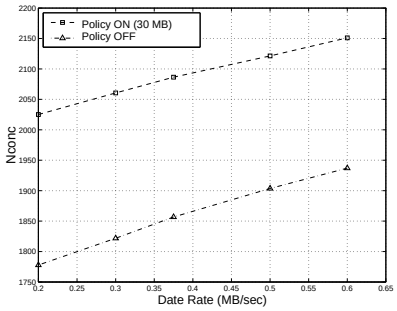


Figure 14: Effect of Movie Data Rate

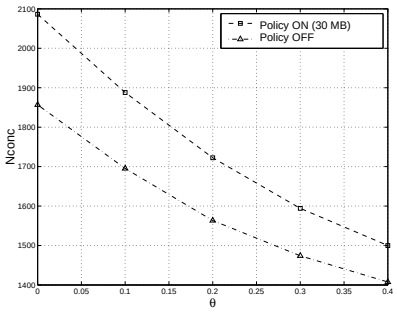


Figure 15: Effect of θ

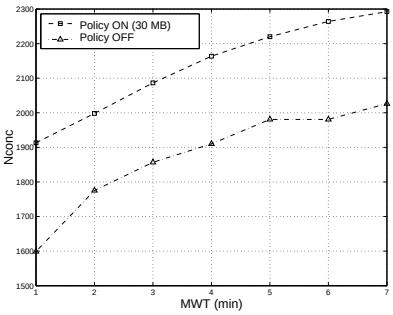


Figure 16: Effect of MWT

in order to keep all disks nearly full. Hence, as the data rate increases, the number of movies (N_m) decreases. The results show that N_{conc} increases with the data rate, which can be explained as follows. The movie locality of reference diminishes as N_m increases since every additional movie consumes part of the total access frequency. The reduction in the locality of reference has a negative impact on both caching and batching. In addition, as N_m increases, the ratio of the capacity of the total cache space to the data capacity of all movies decreases. The results also show that the percentage improvement of the integrated policy decrease with the data rate.

Figure 15 depicts N_{conc} versus θ. As expected, both N_{conc} and the improvement of the integrated policy diminish as the skewness in access patterns decreases because this skewness is proportional to the locality of reference.

The effect of MWT on performance is shown in Figure 16. As MWT increases, more requests can be serviced because they can wait longer, and performance also improves through a higher degree of batching.

6 Evaluation of the Effectiveness of the DIC Algorithm

In the last section, we demonstrated that the performance improvement achieved by the DIC scheme – with only 30 MB cache per disk – in N_{conc} ranges from 2% to 9% 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? In this section, we answer 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 cache. Therefore, the length of a cached interval is bounded by an upper value, denoted as L seconds. Hence, in order to find the number of concurrent cached streams, N, we count the number of arriving requests that fall within L-length distances from their preceding streams during one movie length, D. For this purpose, we find the number of cached requests for each movie individually as follows. (1) We divide a D-length time window into T L-length time windows, where $T = D/L$, and we assume that the leading request in each time window is serviced from disks or cache. We will eliminate this assumption later. (2) We find the total number of subsequent requests in each time window and use it as an initial estimate of the number of concurrent streams. (3) We account for the requests that fall in adjacent time windows but are within L-length distances from each other. After we get an estimate of the number of concurrent cached streams for each movie, we add them up in order to find N. Figure 17 further explains the counting process. The figure shows the arrival of requests for a certain movie 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.

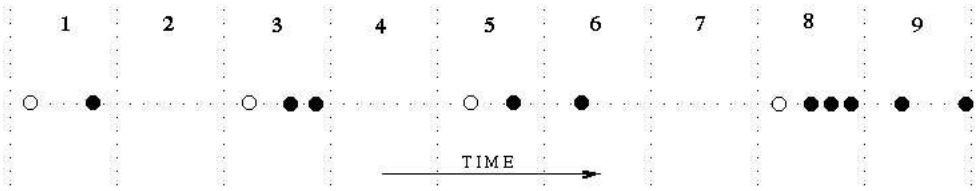


Figure 17: Explaining the Analytical Model

We now present the process of the model development. Table 1 describes many of the parameters used in this subsection. Because the arrival of requests follows a Poisson Process distribution with parameter λ, the inter-arrival time to movie 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 movie j. We also know that the probability of k arrivals for movie j within an L-length time window is $P_k^j = \frac{(\lambda_j L)^k}{k!} e^{-\lambda_j L}$. Thus, the expected number of requests for movie 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 equation (1), $k-1$ is the number of cached requests in each of those L-length time windows. The expected number of arriving requests for movie j that fall within the same L-length time windows as their preceding streams is

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

We need now 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 movie 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 movies is

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

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

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

Finally, the number of cached streams is constrained by the available cache space as follows:

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

Now, we reconsider the assumption that the leading request in each L-length time window is serviced. Instead, we assume that the server accepts that request with probability f. This probability

can be determined by the rejection rate, rr, of the server ($f = 1 - rr$). Thus, assuming that the service of a request is independent from the service of the other requests in the same L-length time window, the number of cached requests in each time window is

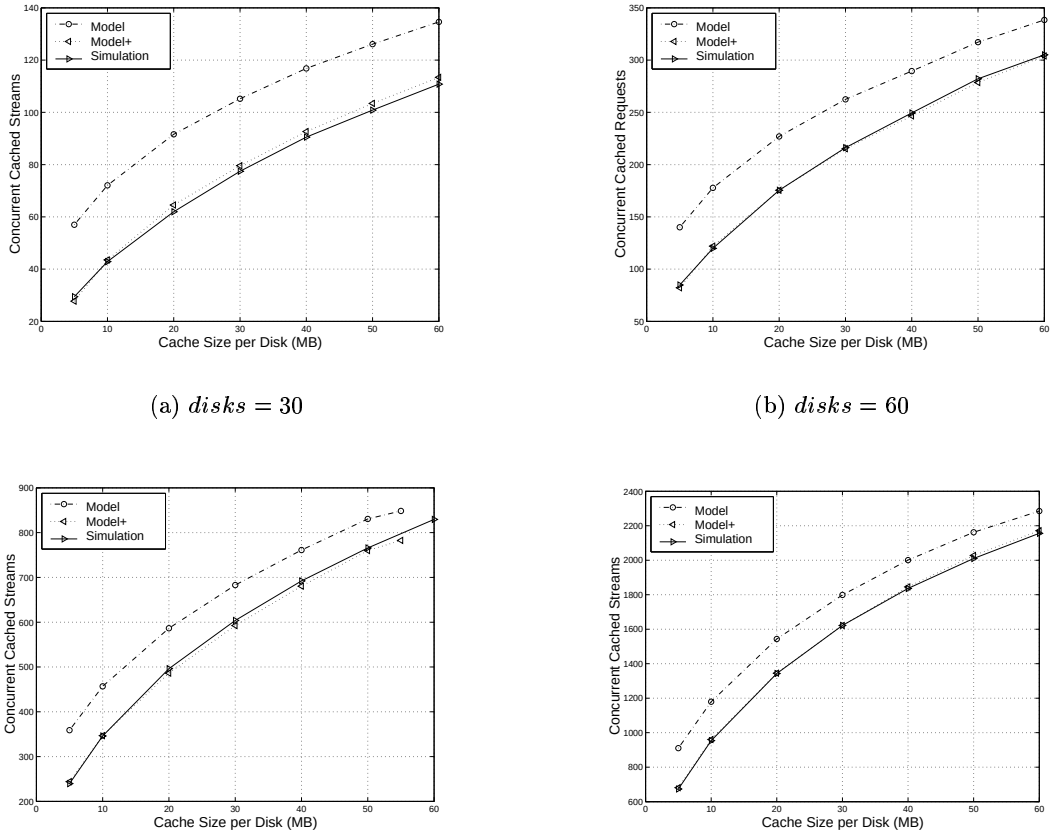
$$\begin{aligned} N_l &= f(k-1) + (1-f)f(k-2) + \cdots + (1-f)^{k-2}f(1) \\ &= f\sum_{j=1}^{k-1}(1-f)^{j-1}(k-j). \end{aligned} \tag{7}$$

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

We built a simulator for the ideal DIC to validate the analytical model. 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. Figure 18 depicts a comparison between the results of the analytical model and those of simulation for four 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 of $E[N]$.

6.2 Comparison between the Ideal DIC and the Proposed Algorithm

Figure 19 plots a comparison between the performance of the ideal DIC and that of the proposed algorithm for four disk configurations. The results show that the proposed DIC algorithm achieves between 23% and 33% 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. 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. Future research is needed to approach the upper limit more closely. We stress that improving the performance of DIC should not be attained by sacrificing the overall performance. For instance, applying (*Criterion 3*) can increase the number of cached streams by reducing the penalty of suboptimal victimization, but as we have shown earlier, that is detrimental to the overall performance. We outline two ways to enhance the performance of the DIC algorithm in the following section.



(a) $disks = 30$

(b) $disks = 60$

(c) $disks = 120$

(d) $disks = 240$

Figure 18: Comparison between Simulation and Analytical Results ($VR = 0.99$)

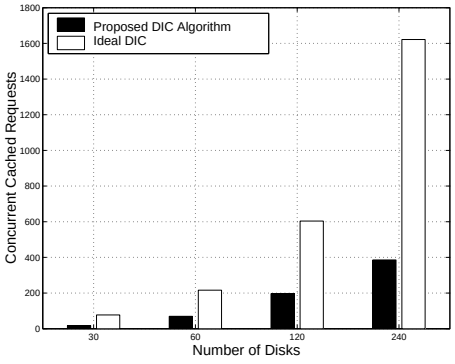


Figure 19: Ideal DIC versus the Proposed Algorithm

7 Conclusions and Future Work

In this paper, we have proposed using the network-attached disk (NAD) architecture to design highly scalable and cost-effective MOD servers. We have used caching, batching, and intelligent scheduling

in an *integrated 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 caches (buffers) to cache intervals between successive streams. We have also proposed a *multi-objective scheduling* (MOS) scheme, which schedules requests based on four predefined performance criteria. The integrated policy is highly configurable; by tuning certain parameters, the administrator can control the tolerable delay and the implementation overhead.

We have studied the effectiveness of the integrated policy and the interaction among various parameters through extensive simulation. The results show that the integrated policy performs better in larger scale servers. In particular, the policy increases the number of concurrent requests by about 12% in a 60-disk to about 17% in a 240-disk server. Similarly, the policy reduces the average request waiting time by about 8% in a 60-disk server to about 40% in a 240-disk server. Moreover, the results indicate that MOS can add 4% improvement in the number of concurrent streams and 20% - 65% improvement in the average request waiting time, with respect to the best performer of existing scheduling policies. The results also show that FCFS is the worst performer among all investigated scheduling policies. These results differ from previous studies primarily because we consider caching, whose performance degrades substantially by scheduling older request. We leave the optimal selection of the values of four scheduling parameters for a future study.

We have also presented an analytical model that estimates the upper bound on the performance of DIC. We 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 directions. First, we will examine using hashing 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. Second, we will evaluate the effectiveness of dividing the disks into partitions compared with merely striping movies across all the disks. In addition to improving the reliability, partitioning should reduce the waiting times before requests find empty slots for scheduling. Reducing such delay should boost performance by shortening the average cached interval. Finally, we will study the implementation of interactive multimedia in NAD-based MOD servers.

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 Transactions on Computers*, 50(2): 97-110, February 2001.

[2] J. Almeida, D. Eager, and M. Vernon. A Hybrid Caching Strategy for Streaming Media Files. *In Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 2001.

[3] W. J. Bolosky, M. B. Jones, and R. F. Rashid. The Tiger video file server. *In Proceedings of Workshop on Network and Operating System Support for Digital Audio and Video*, pages 97-104, April 1996.

[4] S. R. Carter, J.-F. Pâris, S. Mohan, and D. D. E. Long. A Dynamic Heuristic Broadcasting Protocol for Video-on-Demand. *In Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 657-664, April 2001.

[5] A. L. Chervenak. *Tertiary Storage: An Evaluation of New Applications.* Ph.D. thesis, U.C. Berkeley, December 1994. U.C. Berkeley Technical Report UDB/CSD 94/847, December 1994.

[6] A. L. Chervenak, D. A. Patterson, and R. H. Katz. Choosing the Best Storage Systems for Video Service. *In Proceedings of the ACM Conference on Multimedia*, pages 109-119, 1995.

[7] C. Chou, L. Golubchik, J. C .S. Lui. Striping Doesn't Scale: How to Achieve Scalability for Continuous Media Servers with Replication. *In Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 64-71, April 2000.

[8] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching. *In Proceedings of the ACM Conference on Multimedia*, pages 15-23, 1994.

[9] A. Dan, D. M. Dias, R. Mukherjee, D. Sitaram, R. Tewari. Buffering and Caching in Large-Scale Video servers. *In Digest of Papers. IEEE International Computer Conference*, 1995.

[10] A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley. Channel Allocation under Batching and VCR Control in Movie-on-Demand Servers. *Journal of Parallel and Distributed Computing*, 30(2): 168-179, November 1995.

[11] A Dan and D. Sitaram. *Multimedia Caching Strategies for Heterogeneous Application and Server Environments.* Technical Report RC 20670, IBM Watson Research Center, December 1996.

[12] EMC, *Delivering a World of Content Through Rich Media Solutions.* On-line article at `http://www.emc.com`, 2000.

[13] R. Flynn, and W. Tetzlaff. Disk Striping and Block Replication Algorithms for Video File Servers. *In Proceedings of the International Conference on Multimedia Computing and Systems*, pages 590-597, 1996.

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

[15] G. Ganger and J. Schindler. *Database of Validated Disk Parameters for DiskSim.* Available at: `http://www.ece.cmu.edu/~ganger/disksim/diskspecs.html`.

[16] G. Gibson, et al. A Cost-Effective, High-Bandwidth Storage Architecture. *In Proceedings of the Conference on Architecture Support for Programming Languages and Operating Systems*, 1998.

[17] L. Golubchik, J. C. S. Lui, and R. Muntz. Reducing I/O Demand in Video-On-Demand Storage Servers. *In Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 25-36, 1995.

[18] R. Haskin. The Shark Continuous Media File Server. *In Proceedings of IEEE 1993 Spring COMPCON*, pages 12-17, February 1993.

[19] R. Haskin and F. Stein. A System for the Delivery of Interactive Television Programming. *In Proceedings of IEEE 1995 Spring COMPCON*, pages 209-214, March 1995.

[20] K. A. Hua, Y. Cai, and S. Sheu. Patching: A Multicast Technique for True Video-on-Demand Services. *In Proceedings of ACM Multimedia*, pages 191-200, September 1998.

[21] L. Juhn and L. Tseng. Harmonic Broadcasting for Video-on-Demand Service. *IEEE Transactions on Broadcasting*, 43(3): 268-271, September 1997.

[22] S. Jamin, S. Shenkar, L. Zhang, and D. D. Clark. An admission Control Algorithm for Predictive Real-Time Service. *In Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 349-356, November 1992.

[23] K. Keeton, D. A. Patterson, and J. M. Hellerstein. *The Intelligent Disk (IDISK): A Revolutionary Approach to Database Computing Infrastructure, White Paper*, U.C. Berkeley, May 1998.

[24] G. Ma, A. Khaleel, and N. Reddy. Performance Evaluation of Storage Systems Based on Network-Attached Disks. *IEEE Transactions on Parallel and Distributed Systems*, 11(9): 956-968, September 2000.

[25] J. Nieh and M. S. Lam. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. *In Proceedings of the ACM Symposium on Operating Systems Principles*, pages 184-197, October 1997.

[26] J.-F. Pâris. A Fixed-Delay Broadcasting Protocol for Video-on-Demand. *In Proceedings of the International Conference on Computer Communications and Networks*, pages 418-423, October 2001.

[27] P. V. Rangan, and H. M. Vin. Designing File Systems for Digital Video and Audio. *In Proceedings of the ACM Symposium on Operating Systems Principles*, pages 81-94, October 1991.

[28] A. L. N. Reddy, J. Wyllie. Disk Scheduling in a multimedia I/O system. *In Proceedings of the ACM Conference on Multimedia*, pages 225-233, August 1993.

[29] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal Patching Schemes for Efficient Multimedia Streaming. *In Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, June 1999.

[30] P. Shenoy, and V. Harric. Efficient Striping Techniques for Variable Bit Rate Continuous Media File Servers. *Performance Evaluation Journal*, 38(2): 175-199, 1999.

[31] R. Tewari, H. M. Vin, A. Dan, and D. Sitaram. Resource-based Caching for Web Servers. *In Proceedings of the SPIE Conference on Multimedia Computing and Networking*, pages 191-204, January 1998.

[32] A. K. Tsiolis and M. K. Vernon. Group-Guaranteed Channel Capacity in Multimedia Storage Servers. *In Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 285-297, June 1997.

[33] H. M. Vin, P. Goyal, and A. Goyal. A Statistical Admission Control Algorithms for Multimedia Servers. *In Proceedings of the ACM Multimedia*, pages 33-40, October 1994.