# An Integrated Resource Sharing Policy for Multimedia Storage Servers Based on Network-Attached Disks*

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

*In this paper, we propose using the* network-attached disk *(NAD) architecture to design highly scalable and cost-effective* multimedia-on-demand *(MOD) servers. In order to ensure enhanced performance, we propose two schemes, called* distributed interval caching *(DIC) and* multi-objective scheduling *(MOS). The DIC scheme utilizes the on-disk buffers for caching intervals between successive streams, while the MOS scheme improves resource sharing by scheduling requests for service intelligently based on four predefined criteria. We then integrate the two schemes and study the overall performance benefits through extensive simulation. We also study the effectiveness of the proposed DIC scheme by developing an analytical model that estimates the performance limit of DIC. 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, and that the performance improvements scale with the number of disks in the server.*

## 1. Introduction

Recent advances in storage and communication technologies have spurred a strong interest in *multimedia-on-demand* (MOD) servers. These servers eliminate the shortcomings of broadcast-based systems like cable TV by providing customized services. The delivery of rich media contents (video, audio, images, and text) by these servers is projected to be an essential component of the overall e-business strategies for many companies and institutions [7].

The design of MOD servers faces significant challenges to efficiently store, access, and distribute media contents to a large number of concurrent clients because of the stringent requirements of the real-time playback and the high transfer rates. To address these challenges, a wide spectrum of techniques has been proposed, including *admission control* [16], *disk head scheduling* [18], *efficient striping* [21], *block allocation* [8], *block replication* [8], *block rearrangement* [19], and *resource sharing* [4, 12, 14, 15].

Resource sharing techniques exploit the high locality of reference in the access patterns of multimedia contents. These techniques can be classified into five main classes: *batching* [4, 22, 1], *patching* [14], *piggy-backing* [12], *broadcasting* [15], and *interval caching* [5]. Batching off-loads the storage subsystem and uses efficiently server bandwidth and network resources by accumulating the requests to the same contents and servicing them together by utilizing the multicast facility. Patching expands the multicast tree dynamically to include new requests, thereby reducing the request waiting time and improving resource sharing, but it requires additional bandwidth and buffer space at the client. Piggy-backing 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 initial presentation. Broadcasting divides each movie into multiple segments and broadcasts each segment periodically. The improved resource sharing and the fixed waiting times for the playbacks of popular movies come at the expense of requiring relatively very high bandwidth and buffer space at the client. Interval caching caches intervals between successive streams in the server. It does not sacrifice the quality of service, does not lengthen the waiting time, and does not expect much from the client. It can also be applied with other techniques for better resource sharing, and it has become more cost-effective with the diminishing prices of DRAMs.

Current MOD servers, such as Tiger Shark [13], are based on the conventional fileserver architecture (also

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

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 disk* (NAD) architecture [11, 17] 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. 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.

This paper uses the NAD architecture to design highly scalable and cost-effective MOD servers and ensures high performance by proposing an *integrated resource sharing* policy. The principal MOD application of interest in this study is *video-on-demand* (VOD).

The integrated resource sharing policy employs both caching and intelligent scheduling to enhance the performance of NAD-based multimedia servers. For caching, we propose 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 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 these caches for caching intervals between successive streams. The required low-level controllability of caches can be entertained by the growing on-disk intelligence. For scheduling, we propose a scheme, called *multi-objective scheduling* (MOS), which increases the degrees of resource sharing (batching) by selecting requests for service based on four predefined criteria. The integrated policy is highly configurable and can be used to optimize various performance metrics. Analyzing the integrated policy, rather than only the individual schemes, helps in understanding various tradeoffs and making appropriate design decisions. 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.

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 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 simultaneously both these performance metrics. The improvement 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. The MOS scheme can provide an additional $4\%$ improvement in the number of concurrent requests and an additional $20\%$ - $65\%$ improvement in the average request waiting time with respect to the best performer of existing scheduling policies.

Finally, we study the performance limit of DIC through analytical modeling. The model indicates that the limit is very high and thus 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. Finally, we draw conclusions in the last section.

## 2. Distributed Interval Caching (DIC)

By exploiting the high locality of reference in the access patterns of multimedia contents [2], caching can significantly improve the performance of multimedia servers. As caching the entire most commonly used movies is not cost-performance effective in multimedia servers, an alternative scheme, called *interval caching*, was proposed in [5]. 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. When the server accepts a request for service from cache, it starts to cache the data of the preceding stream while retrieving and transferring 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.

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 typical 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 examine 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 an 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

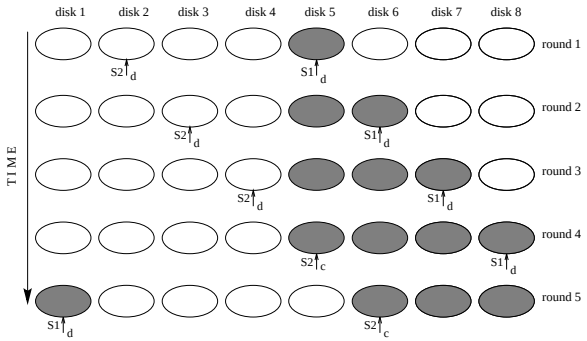



**Figure 1: Explaining Distributed Interval Caching**

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 between disk $f\_disk$ and disk $p\_disk$, inclusively. (Specifically, 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.

When no sufficient cache space exists for caching a new request, the server can victimize an existing request from cache only if all the following four conditions are met. (1) The potential victim can be serviced from disks. (2) The new request can be serviced from disks during catchup. (3) A portion of the cached interval of the potential victim is currently cached in disk $p\_disk$. (4) 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. The DIC algorithm is presented formally in [20].

## 3. Multi-Objective Scheduling (MOS)

A VOD server maintains a waiting queue for each movie and services all requests in a selected queue together using only one stream, whenever the necessary resources become available. The main objectives of scheduling are increasing the number of customers that can be serviced concurrently

and decreasing their waiting times for service.

Scheduling policies for VOD servers include *First Come First Serve* (FCFS) [4], *Maximum Queue Length* (MQL) [4], *Maximum Factored Queue Length* (MFQL) [1], and *Group-Guaranteed Server Capacity* (GGSC) [22]. FCFS selects the queue with the oldest request, while MQL selects the longest queue. Thus, FCFS is fair, whereas MQL is biased against cold (i.e. unpopular) movies. MFQL is similar to MQL, but it reduces the bias against cold movies. GGSC preassigns server channel capacity to groups of requests.

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

**Criterion 1** - This criterion selects the longest queue and can be captured by the optimization function $F_b(i) = B_i$, where $i$ is the movie number, and $B_i$ is the number of waiting requests for movie $i$.

**Criterion 2** - This criterion selects 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 movie $i$.

**Criterion 3** - This criterion 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 movie $i$. 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.

**Criterion 4** - This criterion selects the queue of the most popular movie among all non-empty queues and can be captured by $F_m(i) = 1/i$. (Movies are numbered in decreasing order of popularity.) It favors popular movies, whose requests have a higher tendency to accumulate in the waiting queues and are likely to have closer preceding and following streams.

The MOS scheme applies 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.

## 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. $Q1$, $Q2$,..., and $Qm$ are the movie 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 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* schedules requests in decreasing order of $F$, updates the CIQ, 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.

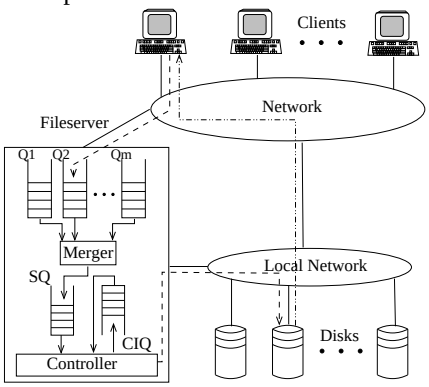



**Figure 2: The Overall Architecture of the Integrated Policy**

The integrated policy enhances the DIC algorithm by incorporating two parameters, called $RCR$ (Request Check Ratio) and $DSR$ (Delay Service Ratio), which should be carefully tuned. $RCR$ specifies the fraction of the requests in $SQ$ that will be considered for scheduling. 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 rate. 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 larger value of $DSR$ increases both batching and caching, but it also increases the average request waiting time.

## 5. Performance Evaluation

### 5.1. Workload Characteristics

Like most prior studies, we assume that the arrival of the requests to a multimedia server follows a Poisson Process with an average arrival rate $\lambda$. We also assume that movies are stored using the MPEG-II compression standard with a

data rate ($r$) of 3 Megabits per second. We also assume that the accesses to movies follow Zipf's distribution [2]. With this distribution, the probability of choosing the $n^{th}$ most popular of $M$ movies is $C/n^{1-\theta}$ with a parameter $\theta$ and a normalized constant $C$. Moreover, we assume that customers can wait until a predefined maximum time ($MWT$). Furthermore, we assume that movies are striped across all disks on $1/2$-second intervals [3]. To keep the discussion focused, we do not consider VCR-like operations and assume that the network bandwidth for each disk is adequate to support the disk transfer rate. VCR-like operations can be supported by allocating contingency channels [6].

## 5.2. Simulation Platform

We developed 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. We experimented primarily with *Quantum Atlas III* disks. We used *DiskSim* [9] to find the disk performance parameters required by our simulator. In this process, we fed DiskSim with synthetic workloads (described in the previous subsection) and with validated disk parameters obtained from [10]. The simulation results show that the mean seek time is $7.778$ ms, the mean rotational latency is $2.431$ ms, and the mean transfer rate is $4.217$ MB/s.

Table 1 shows the default values of the main parameters. We vary the number of disks in the server from $30$ to $240$ and 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 [17]. The request arrival rate, $\lambda$, is usually selected such that the customer defection rate is about $10\%$ when the cache size per disk is $30\ MB$.

**Table 1: Default Parameter Values**

| |
|---|
| Number of disks = 120, Cache size per disk = 30 MB, |
| Number of movies = 480, Movie data rate = 3 Megabits/s, |
| Movie duration = 90 min, $\lambda = 0.435$, $\theta = 0$, |
| $MWT = 3$ min, $VR = 0.7$, $RCR = 0.5$, $DSR = 0.1$, |
| $w_b = w_n = w_m = 0$, $w_t = 1$ |

We analyze primarily the number of requests that can be serviced concurrently and the average request waiting time, although the integrated policy can also be used to optimize other performance metrics. To evaluate the effectiveness of the integrated policy, we consider two cases: *caching is OFF* and *caching is ON*. In the first case, caching is disabled and the two optimizations in the integrated policy regarding $RCR$ 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 are kept the same to those when caching is on to ensure a fair comparison. We evaluate the contribution of the MOS scheme when caching is on by varying only the values of the scheduling parameters.

We will not consider MFQL in the performance study of MOS because MFQL serves as a compromise between FCFS and MQL [1], while this study aims at developing a scheme that can outperform both FCFS and MQL. Similarly, we will not consider GGSC because it does not perform well in high-end servers [22].

## 5.3. Result Presentation and Analysis

The following simulation results are obtained using a steady state analysis with $95\%$ confidence interval. Only a limited set of the results are presented here because of space limitation. Additional results can be found in [20].

### 5.3.1. Effectiveness of the Integrated Policy

We demonstrate here the effectiveness of the integrated policy and discuss the impacts of the cache size per disk, the number of disks, the request arrival rate, and maximum request waiting time. Figure 3 plots the number of requests that can be serviced concurrently ($N_{conc}$) and the customer defection rate versus the cache size per disk, while the number of disks is fixed at $120$. (Caching is turned off when the cache size is $0$). As expected 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 4 depicts the impact of the number of disks, while the cache size per disk is fixed at 30 MB. Note that the benefit of the integrated policy generally increases with the number of disks because of the server can handle a higher $\lambda$. Increasing $\lambda$ increases the number of cached requests by shortening the intervals between successive streams and improves batching through the accumulation of more requests in the waiting queues. Figure 5 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 $240$. Caching contributes a little less than half of the improvement on the average.

Figure 6 depicts $N_{conc}$ versus the average request interarrival time ($1/\lambda$). These results show that the $N_{conc}$ increases with $\lambda$. 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 $\lambda$. 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 7 illustrates the effects of the number of disks and the cache size per disk on the average request waiting

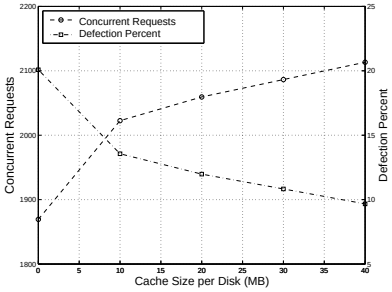



**Figure 3: Effect of Cache Size per Disk**

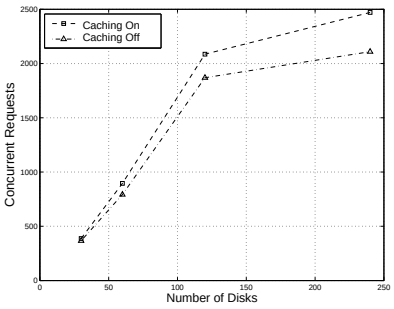



**Figure 4: Effect of Number of Disks**

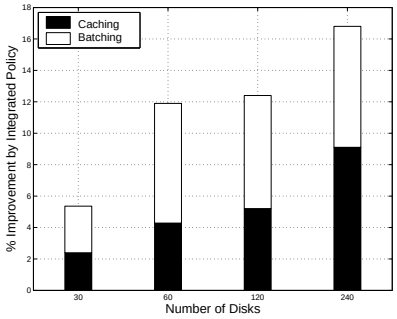



**Figure 5: Contributions to Improvements in Concurrent Requests** ($VR = 0.99$)

time ($T_{wait}$). The results show that when caching 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 8 demonstrates the effect of $MWT$ on performance. Note that performance improves with $MWT$ through higher degrees of resource sharing because requests can wait longer for service.

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

Let us now discuss how to tune $RCR$, $VR$, and $DSR$. Figure 9(a) plots $N_{conc}$ versus $RCR$. Note that $N_{conc}$ 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 9(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. The impact of $DSR$ on $N_{conc}$ and $T_{wait}$ is shown in Figure 9(c). As expected, increasing $DSR$ has a positive effect on $N_{conc}$ and a negative effect on $T_{wait}$. Note that increasing $DSR$ from $0.1$ to $0.9$ improves performance by more than $5.3\%$ but increases $T_{wait}$ by about $40\%$. A large value of $DSR$ should be chosen only if customers are not very sensitive to delay.

#### 5.3.3. Effects of the Scheduling Parameters

Let us now 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 10 and 11 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 $\lambda$ 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 $\lambda$ 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 $\lambda$ 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 these metrics can

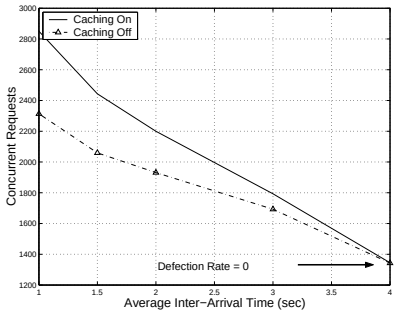



**Figure 6: Effect of Average Inter-Arrival Time**

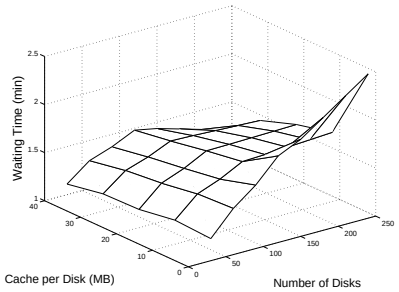



**Figure 7: Effect of Number of Disks and Cache Size on Waiting Time**

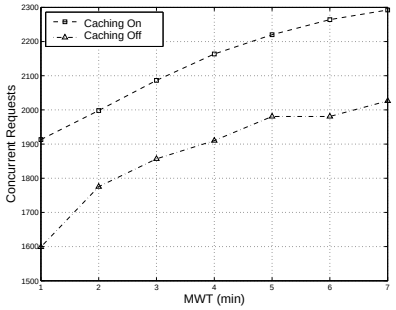
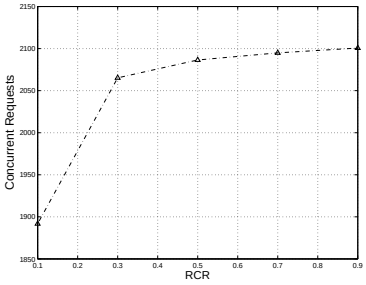
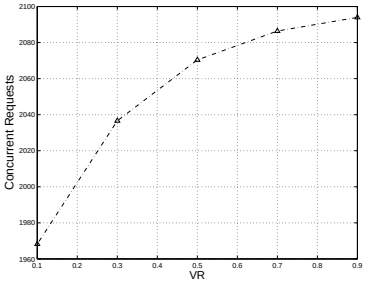


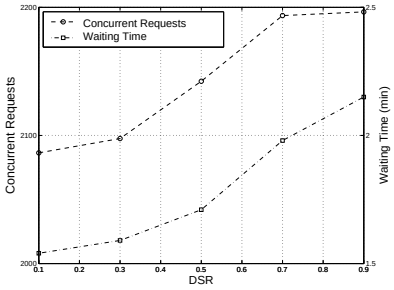

**Figure 8: Effect of Maximum Waiting Time (MWT)**




(a) $RCR$




(b) $VR$




(c) $DSR$

**Figure 9: Effects of the Tunable Parameters**

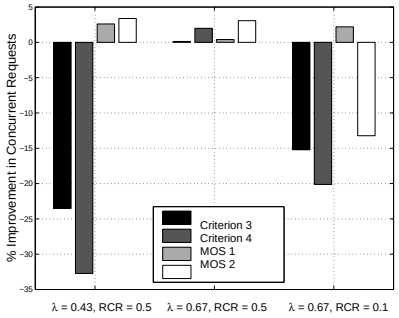



**Figure 10: Improvements in Concurrent Requests cf Criterion 1**

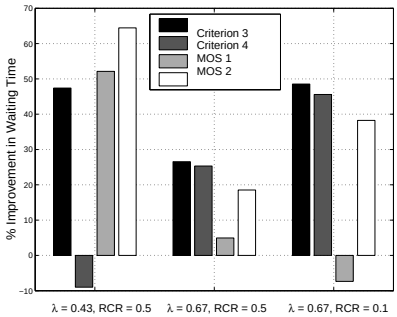



**Figure 11: Improvements in Waiting Time cf Criterion 1**

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.

## 5.4. Performance Limit of DIC

The performance improvement achieved by the DIC scheme are naturally less significant than those achieved by caching in general-purpose systems because of the much larger object size(s) in media servers. But, can we squeeze out additional performance gains by polishing the proposed scheme? To answer this question, we have estimated the performance limit in terms of the number of concurrent cached streams by developing an analytical model of an *ideal* DIC scheme, which minimizes the average cached interval by perfect victimization and fully utilizes all the on-disk caches. We have also validated this model by simulation. The model is not shown here for space limitation but can be found in [20]. The results indicate that the limit is very high and that the proposed DIC algorithm achieves between $23\%$ and $33\%$ of that limit. The performance gap is primarily because the load is not perfectly balanced among all caches with the proposed algorithm, and victimization is highly constrained. The limit, however, is somewhat loose because the constrained victimization is a nature of the problem, not of the algorithm. In future work, we will investigate two promising ways to improve the DIC algorithm: applying hash functions for mapping movies to disks (instead of the simple round-robin scheme) and divid-

ing disks into partitions. The first way should reduce the load imbalance, while the second should reduce the waiting times and defection probability.

## 6. Conclusions

In this paper, we have proposed using the *network-attached disk* (NAD) architecture to design highly scalable and cost-effective *multimedia-on-demand* (MOD) servers. We have used both caching 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) in caching intervals between successive streams. For scheduling, we have proposed a *multi-objective scheduling* (MOS) scheme, which schedules requests based on four predefined criteria.

We have studied the effectiveness of the integrated policy and the interaction among various parameters. The simulation results show that the integrated policy performs better in larger scale servers. In particular, it 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 the MOS scheme can add $4\%$ improvement in the number of concurrent requests and $20\%$ - $65\%$ improvement in the average request waiting time with respect to the best performer of existing scheduling policies. We have also studied the performance limit of DIC through analytical modeling and have outlined two ways, which will be explored in future work, to approach this limit more closely.

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

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

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

[4] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching. *In Proc. of the ACM Conf. on Multimedia*, pages 391-398, October 1994.

[5] A. Dan, D. M. Dias, R. Mukherjee, D. Sitaram, R. Tewari. Buffering and Caching in Large-Scale Video servers. *In Proc. of IEEE Int'l Computer Conf.*, pages 217-225, March 1995.

[6] 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.

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

[8] R. Flynn, and W. Tetzlaff. Disk Striping and Block Replication Algorithms for Video File Servers. *In Proc. of the Int'l Conf. on Multimedia Computing and Systems*, pages 590-597, June 1996.

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

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

[11] G. Gibson, et al. A Cost-Effective, High-Bandwidth Storage Architecture. *In Proc. of the Conf. on Architecture Support for Programming Languages and Operating Systems*, pages 92-103, October 1998.

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

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

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

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

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

[17] 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.

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

[19] N. J. Sarhan and C. R. Das. Adaptive Block Rearrangement Algorithms for Video-On-Demand Servers. *In Proc. of Int'l Conf. on Parallel Processing*, pages 452-459, September 2001.

[20] N. J. Sarhan and C. R. Das. *Caching and Scheduling Techniques for Multimedia Storage Servers Based on Network-Attached Disks*. Tech. Report CSE-02-013, Department of Computer Science and Engineering, The Pennsylvania State University, September 2002.

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

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