Multimedia Tools and Applications, 20, 237–262, 2003

A Design Framework for Multi-Resolution Video Servers

JINSUNG CHO chojs@khu.ac.kr
MINYOUNG SUNG my.sung@samsung.com
HEONSHIK SHIN shinhs@snu.ac.kr
School of Electronics & Information, Kyung Hee University, Yongin 449-701, Korea

Abstract. Video can be encoded into multiple-resolution format in nature. A multi-resolution or scalable video stream is a video sequence encoded such that subsets of the full resolution video bit stream can be decoded to recreate lower resolution video streams. Employing scalable video enables a video server to provide multiple resolution services for a variety of clients with different decoding capabilities and network bandwidths connected to the server. The inherent advantages of the multi-resolution video server include: heterogeneous client support, storage efficiency, adaptable service, and interactive operations support.

For designing a video server, several issues should be dealt with under a unified framework including data placement/retrieval, buffer management, and admission control schemes for deterministic service guarantee. In this paper, we present a general framework for designing a large-scale multi-resolution video server. First, we propose a general multi-resolution video stream model which can be implemented by various scalable compression techniques. Second, given the proposed stream model, we devise a hybrid data placement scheme to store scalable video data across disks in the server. The scheme exploits both concurrency and parallelism offered by striping data across the disks and achieves the disk load balancing during any resolution video service. Next, the retrieval of multi-resolution video is described. The deterministic access property of the placement scheme permits the retrieval scheduling to be performed on each disk independently and to support interactive operations (e.g. pause, resume, slow playback, fastforward and rewind) simply by reconstructing the input parameters to the scheduler. We also present an efficient admission control algorithm which precisely estimates the actual disk workload for the given resolution services and hence permits the buffer requirement to be much smaller. The proposed schemes are verified through detailed simulation and implementation.

Keywords: Video server, multi-resolution video stream model, data placement and retrieval, admission control

1. Introduction

During the past decade we paid great attention to a video-on-demand (VOD) service which provides the combined facilities of a video rental store over high-speed networks. The realization of such services requires the development of VOD servers that support efficient mechanisms for storing and delivering video streams. The fundamental problem in developing video servers[1] is that the delivery and playback of video streams must be performed at real-time rate [16]. A great deal of work has been done on video servers with the research issues: guaranteed retrieval and delivery of video streams [1, 35], disk scheduling algorithms for improving disk performance [8, 28], data placement on a single disk or multiple disks for high throughput of storage subsystems [2, 33, 36], variable bit rate stream scheduling and

admission control for the deterministic/statistical QoS guarantee [24, 29, 34], efficient buffer management and caching [13, 25, 37], interactive operations support [7, 9, 38], hierarchical storage management for cost-effective design [15, 21], and scalable server architecture and scheduling algorithms in parallel or distributed environment [4, 12, 17, 22]. The issues, however, are closely coupled with each other.

Recent advances in video coding technology make it possible to create a multi-resolution or scalable video stream. In general, a multi-resolution video stream permits the extraction of lower resolution subsets from the full resolution stream that may be decoded independently. Employing the multi-resolution video in VOD servers provides the following benefits: *heterogeneous client support, storage efficiency, adaptable service, and interactive operations support*.

First, clients in a VOD service are likely to request various QoS parameters, such as color depth, window size, and frame rate, because they have different decoding capabilities and network bandwidths connected to the server. Second, servicing single (full) resolution video streams for a wide range of clients results in wasting server resources such as disk and network bandwidths. Since multiple versions with different resolutions for each video stream lead to storage inefficiency, it is required to employ scalable video. Third, on the transient overloaded condition, the server is able to provide adaptable services by gracefully degrading the resolution levels. Furthermore, even when the admission of new clients fails, the storage of scalable video permits the server to gracefully degrade the resolution levels of existing clients in order to service new clients. In addition, the server can provide adaptability to the fluctuation in network bandwidth, which is one of fundamental problems in mobile computing environment. Finally, the lower resolution streams enable the server to efficiently support interactive operations such as fastforward and rewind.

There exist several works related to multi-resolution video servers whose research issues include multi-resolution video model, data placement and retrieval of video data, and interactive operations support. Chiueh and Katz [11] employ the specific multi-resolution video representation coded in a Laplacian or Gaussian pyramid and lay out video data on a two-dimensional disk array. The result of a simulation study shows that under synthetic workload the multi-resolution scheme performs significantly better in terms of I/O rate, average waiting time, and average physical data bandwidth requirement as compared with full-rate single resolution video. Keeton and Katz [19] propose several strategies for the layout of multi-resolution video data. Each of their strategies explores different aspects of the parallelism and concurrency offered by striping data across multiple disks. They evaluate their layout strategies through simulation in a standard file server environment. Chang and Zakhor [5, 6] present a constant frame grouping method to order data rate layers within one storage unit on a disk and a periodic interleaving method to arrange the storage unit on multiple disks. Paek et al. [26] explore a flexible data placement strategy for independent parallel disk arrays. The placement strategy integrates both schemes of [19] and [5] into a multiple segmentation scheme and hence provides a trade-off between disk utilization efficiency and interactive delay. Chen et al. [10] suggest an idea of staggering scalable data blocks in order to achieve better load balancing and reduce buffer requirements. Shenoy and Vin [31] present an encoding technique for MPEG video using a combination of temporal and frequency scaling to support interactive scan operations. They

demonstrate that the lowest resolution stream provides acceptable visual quality for scan operations.

However, there has been little work integrating all of the research issues which may include data placement/retrieval, buffer management, interactive operations support, and admission control schemes for deterministic service guarantee. In addition, many of the previous works assume the specific video stream such as MPEG and the server architecture. In this paper, we intend to present a general design framework by proposing and assuming the general video stream model and server model. Under the given model, we deal with the individual issues in a unified manner.

First of all, we propose a general multi-resolution video stream model which can be implemented by various scalable compression techniques. Next, under the scalable disk-array-based server model, we devise a hybrid data placement scheme which exploits both concurrency and parallelism offered by striping data across disks in the server and propose the *request scheduling* which determines the start point of each service to minimize the maximum number of blocks retrieved in a disk. The placement scheme with the request scheduling is shown to evenly distribute the disk workload, so that it increases the concurrent clients, since the most heavily loaded disks determines the maximum number of concurrent clients. Furthermore, the deterministic access property of the data placement scheme permits that the retrieval scheduling can be performed on each disk independently and that the admission control algorithm can precisely estimates the actual disk workload. The precise admission control algorithm makes the buffer requirement much smaller. The simple retrieval scheduler with the multi-resolution video stream can support interactive operations just by reconstructing the input parameters without any additional disk workload.

The rest of the paper is organized as follows: In Section 2, we describe the video stream model and the server model. Sections 3 and 4 present the data placement and retrieval scheme for multi-resolution video, respectively. Data placement and retrieval are closely coupled with each other. In Section 4, we also describe how to support interactive operations given the data placement and retrieval, and propose the admission control algorithm. In Section 5, we validate the proposed schemes through the detailed simulation with scalable video trace data and implement a small-scale prototype for the multi-resolution VOD system. Finally, Section 6 concludes the paper.

2. The model

2.1. Multi-resolution video stream model

In general the notion of video resolution is defined in three dimensions: chroma, spatial, and temporal. In these dimensions, video streams can be compressed into multiple-resolution format by various scalable compression algorithms. A multi-resolution or scalable video stream is a video sequence encoded such that subsets of the full resolution video bit stream can be decoded to recreate lower resolution video streams.

For the purpose of modeling multi-resolution video streams, we propose a z-level multi-resolution video stream model in figure 1. A multi-resolution video stream is a set of

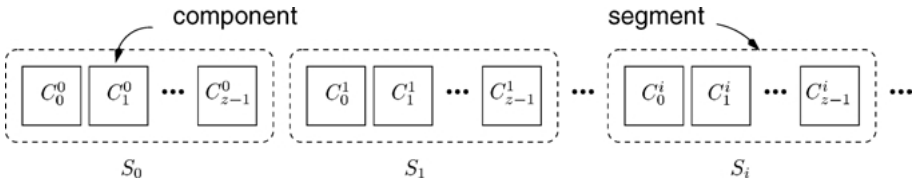


Figure 1. z-level multi-resolution video stream model.

segments in which a segment consists of z components. In other words, for a video stream V,

$$V = \{S_s \mid S_s \text{ is a segment},\ 0 \le s < l\}$$
$$S_s = \left\{C_c^s \;\middle|\; C_c^s \text{ is a component},\ 0 \le c < z\right\},$$

where s and c denote the segment number and the component number, respectively, and l is the number of segments, or the length of V. The k-level resolution can be obtained by integrating k components from the lowest one; so, $\{C_c^s \mid 0 \le s < l,\ 0 \le c < k\}$ are serviced. In our multi-resolution video model, each video stream can be provided with z levels of quality and the QoS parameter is represented by the number of components in a segment, or k. Full resolution quality dictates the use of all the components.

The multi-resolution video stream described above can be implemented by various coding technologies. The current scalable video compression techniques include DCT-based schemes, subband (wavelet) schemes, fractal-based schemes, and object-based schemes [18]. Of the standard codecs, only MPEG-2 addresses scalable video streams. Four techniques, namely data partitioning, SNR scalability, spatial scalability, and temporal scalability, can be used. Since a frame consists of multiple layers, a frame can be mapped to a segment in our video stream model and the sub-layers of the frame constitute components of the segment. Alternatively, multiple frames may be mapped to a segment because a large storage/retrieval unit is beneficial to the disk performance [6]. For example, Paek et al. implement a three layer MPEG-2 video stream in [26]. In their scheme, the base layer provides the initial resolution video while an additional spatial enhancement layer allows for the upsampling and hence increases in frame size of the base layer. A further SNR enhancement layer provides increase in the visual quality of the base + spatial enhancement layers of video. One possible mapping from their video stream to our model is that a group-of-picture (GOP) corresponds to a segment, that is,

$$V = \{S_s \mid S_s = \text{GOP}^s,\ 0 \le s < l\}$$
$$S_s = \left\{C_c^s \;\middle|\; C_c^s = \text{GOP}_c^s,\ 0 \le c < 3\right\},$$

where GOP^s denotes the s-th GOP. The components GOP_0^s, GOP_1^s, and GOP_2^s are the base layer, the spatial enhancement layer, and the SNR enhancement layer for GOP^s, respectively.

MPEG-1, which is another DCT-based coding scheme, can also exploit scalability techniques such as data partitioning with slight modification to the existing codecs [30]. In

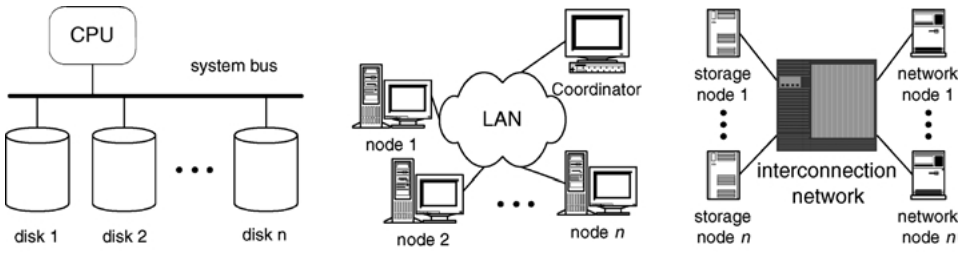
addition, without modifying the codecs, we can reconstruct MPEG-1 video into our multi-resolution video model in temporal dimension as follows: (1) A GOP is mapped to a segment. (2) An I frame is the first component in a segment. (3) P frames constitute the next one or more components. (4) B frames constitute the rest of the components in the segment. This is similar to the rearrangement scheme of Chang and Zakhor [5] which stores the frames within a GOP in a specific order.

Taubman and Zakhor [32] propose and implement a scalable codec capable of generating bit rates from tens of kilo bits to several mega bits per second with fine granularity of available bit rates. The codec is based on 3-D subband coding and multi-rate quantization of subband coefficients, followed by arithmetic coding. Chang and Zakhor [6] use 11-layer scalable video streams produced by the codec which range from 190 Kbps and 1330 Kbps in their work for storage and retrieval of scalable video. We can reconstruct the video streams into our video model in the same way as the scalable MPEG-2 which is described above.

Bogdan [3] proposes a multi-scale fractal video coding. The scheme combines the still image pyramid coding and the ITT (iterated transformation theory) inter-frame video coding methods to generate a hierarchy of bit-streams. MPEG-4 is scalable in the sense that multiple objects can be added or removed to compose a frame. The fractal-based and object-based coding schemes are also consistent with our model. Consequently, we can conclude that we can utilize 'off-the-shelf' technology in order to implement our multi-resolution video stream model.

2.2. *Server model*

Video servers range from a standard PC for small-scale systems to massively parallel or distributed computers for large-scale systems, as depicted in figure 2. Since the most demanding resources in video servers are I/O (disk and network) bandwidth and storage, the single server model adopts disk arrays for large bandwidth and storage capacity [2, 36]. However, the single server approach has its limitations of scalability and fault tolerance [22]. In order to overcome the limitations, distributed or parallel servers have been designed and implemented [4, 12, 17, 22]. The striping techniques in disk arrays can be incorporated into



(a) single server (b) distributed server (c) parallel server

Figure 2. The video server architecture.

the distributed/parallel server model for high aggregated throughput and load balancing among nodes which comprise the server. In addition, each node in distributed or parallel servers should have high-performance and reliable disk subsystems such as RAID.

The architecture in figure 2 can be modeled into a disk array model where the server has d disks[2] and video data are striped across the disks. In distributed or parallel servers, a disk corresponds to a disk subsystem of each node. In this paper, we assume the disk array model for our video server architecture and consider the large-scale case (distributed or parallel server) for system parameters. Many works are also founded on the model but most of them analyze the performance of a disk (i.e. seek time, rotational latency, and transfer rate) based on single server architecture in figure 2(a). The analysis is not directly applicable to the RAID disk subsystem in distributed or parallel servers. For flexibility, we consider only the effective bandwidth Φ for the performance of a disk subsystem. The value can be measured from a calibration program that determines the maximum number of blocks that can be read within the given time interval [24]. For convenience's sake, we use term 'disk' instead of 'disk subsystem' in the rest of this paper. In general, the bandwidth of interconnection network between nodes in large-scale servers is larger than that of disk subsystem. So, we generalize the server model where the server has d disks of which the effective bandwidth is Φ.

A video server proceeds in periodic rounds due to its periodic nature. In each service round of which the length is T_R, a video server retrieves the required amount of data with respect to its playback duration and transmits them to remote clients. A double buffer scheme enables the disk and network bandwidths to be effectively utilized. In other words, in each round, while data are retrieved to maximize the disk performance, the data retrieved in the previous round are transmitted to ensure the real-time playback capability considering the buffer space of each client. Assuming that the network bandwidth is large enough for the transmission, we are concerned about the effective disk bandwidth management for multi-resolution video data. For effectiveness, we define the performance metrics of video servers as *concurrency*, *interactivity cost*, and *service latency*. Video servers should be able to provide services for *as many concurrent clients as possible* while guaranteeing their real-time playback. In addition, *interactive operations* such as pause, resume, fastforward, rewind, and slow playback, should be supported with reasonable cost. The *service latency* upon startup or interactive operations should be acceptable.

3. Data placement for multi-resolution video

The performance of video servers is closely related with data placement. A data placement scheme should explore the followings: First, it should provide deterministic access for simple retrieval scheduling. Second, the performance efficiency should be considered such as throughput and service latency. Third, it should support interactive operations with reasonable cost. Next, the disk load balancing should be achieved so that the server may be able to fully utilize the aggregate disk bandwidth.

Before placing data on disks, we first have to determine storage units by which data are written to or read from disk. Constant bit rate (CBR) video streams require the equal amount of data in each round, but variable bit rate (VBR) streams do not. There exist two methods for VBR streams [6]. The constant time length (CTL) method is to store and retrieve video

data in unequal amounts with respect to its real-time playback duration. In contrast, the constant data length (CDL) is to store and retrieve data in equal-sized blocks while utilizing buffer memory to provide real-time playback. The former provides advantages in buffer usage and disk throughput but has the fragmentation problem. The latter is consistent with the current disk storage technology but requires large buffer space and complicated retrieval scheduling. In order to alleviate the problems, we can employ a hybrid method in which data are stored in fixed-size blocks, but the number of blocks to be retrieved varies with the playback duration. The CTL method is more efficient in a read-only environment such as VOD because it reads a large chunk of data contiguously while there exist seek operations in the hybrid method [6]. On the other hand, the hybrid method is a viable approach for the design of integrated multimedia file system where multimedia data are created, edited, and deleted frequently [36].

We model the hybrid approach in consideration of flexibility and allocate a variable number of fixed-size blocks for a component, that is, $size(C_c^s) = b_c^s B$, where B denotes the size of a disk block. However, if we choose the smallest allocation unit for B (one sector, or 512 bytes) and place blocks in a component contiguously, it results in the CTL scheme. As for the component size $size(C_c^s)$ in our multi-resolution video stream model, we construct a segment based on the round length T_R, so that a segment is serviced in each round. This leads to a large and logically contiguous data chunks, and hence, the high disk throughput can be achieved.

We now intend to place multi-resolution video data on a disk array. There exist two straightforward strategies which explore different aspects of the concurrency and parallelism offered by striping data across disks, as depicted in figure 3. The degree of concurrency is defined as the number of outstanding requests at one time and the parallelism describes the number of disks that service a single request. Chang and Zakhor [5] propose the periodic interleaving scheme using the concurrency of multiple disks and Paek et al. [26] define the second strategy (parallelism) as the balanced placement scheme. The periodic interleaving scheme accesses only one disk in a round for a segment and, in the balanced placement, a segment is divided into d equal amount of data and placed across all d disks.

Two extremes of data placement show a tradeoff of disk throughput versus service latency. The periodic interleaving scheme achieves high disk throughput owing to the large and

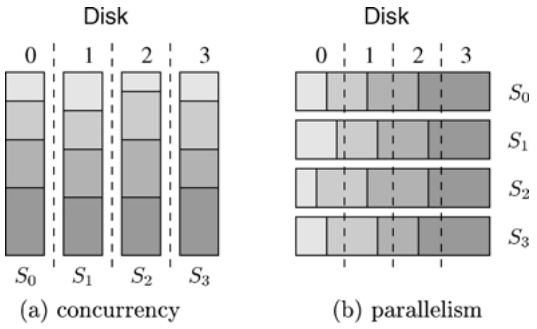


Figure 3. Striping strategies: concurrency vs. parallelism.

logically contiguous data chunks, but the worst case service latency is d rounds [26] because a service should be delayed considering the load balancing of disk bandwidth. The service delay consists of the waiting time plus one round for filling a buffer in the double buffer system. On the contrary, in the balanced placement, Paek et al. argue that the service latency is one round all the time although relatively small data chunks result in lower disk throughput. They also present a hybrid multiple segmentation scheme on the basis of the tradeoff analysis. In the scheme, they define a segmentation level S which represents the degree of parallelism. Each segmentation group (S disks) is performed in parallel and d/S disks concurrently.

All the schemes are based on full-resolution services. For lower resolution services, a small quantity of data are retrieved in the periodic interleaving and a subset of disks participate in the retrieval in the balanced placement scheme. Hence, both schemes cannot guarantee their advantage (i.e. throughput and service delay, respectively) for lower resolution services. In addition, they do not consider the disk access boundaries for each component so that each component in a segment is not accessible independently. Furthermore, the load balancing issue of disk bandwidth for VBR streams is not described.

To take advantage of both concurrency and parallelism for each resolution service, we place each segment of a video stream in parallel and each component in a segment concurrently. In other words, since the independent access unit is a component, we place each component contiguously in a disk and stripe the components in a segment across disks. The finer storage granularity provides the advantages over the periodic interleaving scheme: better load balancing and less disk bandwidth fragmentation. Disk bandwidth fragmentation refers to a situation where the available bandwidth in each disk is not sufficient to accommodate an incoming request, although there is sufficient aggregate bandwidth across disks in the array [10]. On the other hand, the proposed placement scheme guarantees the sequential and independent access to each component, which the balanced scheme does not provide. The balanced scheme incurs the load imbalance problem on lower resolution services because the lower resolution components are placed on the same disk (see figure 3).

We begin by introducing an example of data placement in figure 4. Three cases are identified according to the resolution level of stream z and the number of disks d. First, in case of $z \leq d$, components in a segment are distributed across all the disks and successive

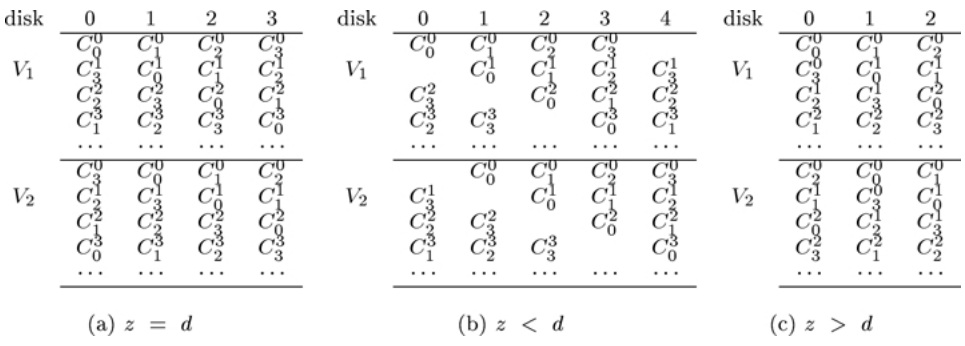


Figure 4. An example of data placement for multi-resolution video.

components are placed on adjacent disks.[3] Next, for load balancing, the first components of successive segments S_i and S_{i+1} (i.e., C_0^i and C_0^{i+1}) are assigned on adjacent disks (disk j and disk $j+1$), as shown in figure 4(a) and (b). In addition, we distribute the starting point (the first component in the first segment, or C_0^0) of each stream across disks for load balancing when multiple streams are requested concurrently. For the service of V_1 in figure 4(a) with the second level resolution, for example, the first segment is retrieved from disks 0 and 1 (C_0^0 and C_1^0, respectively) and the second from disks 1 and 2 (C_0^1 and C_1^1). Next, when $z > d$, multiple components in a segment may be placed on a disk. However, the strategy is similar to the case of $z \leq d$. That is, successive components in a segment are placed on adjacent disks and the first components of successive segments are assigned on consecutive disks as depicted in figure 4(c). The data placement scheme allows deterministic access to disks. For $V = \{C_c^s \mid 0 \leq s < l,\ 0 \leq c < z\}$, the disk which contains a component C_c^s is identified as follows[4]:

$$D\left(C_c^s\right) = [s + c + StartDisk_V]_d, \tag{1}$$

where $StartDisk_V$ indicates the disk in which the starting point (C_0^0) of V is stored.

We now show the disk load balancing property of our placement scheme. Since the number of concurrent clients in a video server with multiple disks depends on the most heavily loaded disks [36], this property should be examined carefully. Let $\mathcal{V}_{i,k}$ denote a set of components retrieved from disk i during k-level service of V. From Eq. (1), we obtain

$$\mathcal{V}_{i,k} = \left\{C_c^s \;\middle|\; D\left(C_c^s\right) = i, 0 \leq s < l, 0 \leq c < k\right\}. \tag{2}$$

Theorem 1. *Given the parameters above, $|\mathcal{V}_{i,k}|$ is computed as follows*:

$$|\mathcal{V}_{i,k}| = \left\lfloor \frac{l}{d} \right\rfloor \times k + \alpha \quad (0 \leq \alpha < d) \tag{3}$$

Proof: See Appendix B. □

Theorem 1 indicates that, regardless of the resolution level of video service, components are evenly distributed across all the disks. Disk load balancing in CBR video streams can be directly derived from Theorem 1 because CBR streams have equal-sized components of $size(C_c^s) = bB$, $0 \leq s < l$, $0 \leq c < z$.

When a VBR scalable coding algorithm is employed for compression efficiency, the size of each component varies, that is, $size(C_c^s) = b_c^s B$ is not constant. This may lead to the load imbalance. In [31], Shenoy and Vin suggest a hybrid scheme for determining the block size in which the block size can vary across sub-streams (lower resolution streams) but is fixed for a given sub-stream to maximize performance. Their data placement strategy, however, stores blocks of sub-streams that are likely to be accessed together adjacent to each other on disk and hence results in the concurrency model in figure 3(a). The variable size of components in VBR streams can be solved in our multi-resolution video stream model by fixing the number of blocks in the same component level, or $b_c^s = b_c$, $0 \leq s < l$. $n(\mathcal{V}_{i,k})$, or

the number of blocks for storing components in $\mathcal{V}_{i,k}$, is calculated from Eq. (3).

$$n(\mathcal{V}_{i,k}) = \sum_{s} \sum_{c} \sum_{C_c^s \in \mathcal{V}_{i,k}} b_c^s = \sum_{c=0}^{k-1} \left(\sum_{s, C_c^s \in \mathcal{V}_{i,k}} b_c \right) \approx \sum_{c=0}^{k-1} \left\lfloor \frac{l}{d} \right\rfloor b_c \tag{4}$$

From Eq. (4), we can observe that the disk workload in VBR streams is also evenly distributed for any resolution video stream. Even when b_c^s varies within the same component level, it is expected that the following equation holds statistically:

$$n(\mathcal{V}_{i,k}) \approx \sum_{c=0}^{k-1} \left\lfloor \frac{l}{d} \right\rfloor E[b_c], \tag{5}$$

where $E[b_c]$ is the mean of b_c^s for $0 \leq s < l$, or $E[b_c] = \frac{1}{l} \sum_{s=0}^{l-1} b_c^s$. We will validate Eq. (5) through experiments in Section 5.

Observe that we achieve the disk load balancing for any resolution video service. However, we have to consider another load balancing issue for the workload induced by concurrent clients. The issue should be treated in the retrieval scheduling upon startup or interactive operations.

4. Data retrieval for multi-resolution video

As mentioned earlier, a video data retrieval proceeds in periodic rounds. In our video server, multi-resolution video data are constructed such that a segment is played back for T_R. So, for each video stream V_j, a segment is serviced in a round, and hence, k components are retrieved in parallel across disks for k-level resolution service. Since our data placement scheme allows deterministic access to each component, each disk can retrieve data independently.

In figure 5, we present a simple retrieval scheduling procedure performed at disk i in each round. The input parameters of a video stream consist of its resolution level (k_j), the current segment number (s_j), the number of segments retrieved in a round (K_j) which is one in normal playback, $StartDisk_{V_j}$, and playback *direction* which is set to 1 or -1 according to forward and reverse playbacks, respectively. The scheduler generates a set of components $\mathcal{D}_i$ to be retrieved from its disk in each round. In Line 10 of figure 5, we can incorporate a disk scheduling algorithm to optimize the performance of its disk subsystem. When a client degrades its resolution level, the request can be serviced simply by changing the input parameters to the scheduler without any additional disk workload. The service delay of the request is only one round, or T_R.

While each disk performs data retrieval independently, we have to schedule the requests of clients globally (*request scheduling*), as mentioned in Section 3, so that the disk workload induced by concurrent clients may be evenly distributed across the disks. The strategy is to delay the start point of service considering the disk load balancing. Since the workload in a disk is shifted to the next disk in the next round, we can calculate the average workload in each disk for the next r rounds ($r \leq d$). Our observation is that *the maximum number*

```
1.  Procedure Scheduler_i
2.  input: V_j = (k_j, s_j, K_j, StartDisk_{V_j}, direction), 1 ≤ j ≤ N
3.  output: D_i, a set of components to be retrieved from disk i in a round
4.  begin
5.      clear D_i
6.      for j := 1 to N
7.          for m := 1 to K_j
8.              for c := 0 to k_j − 1
9.                  if ([s_j + c + StartDisk_{V_j}]_d = i)
10.                     insert C_c^{s_j} into D_i
11.                 end if
12.             end for
13.             s_j = s_j + direction
14.         end for
15.     end for
16. end
```

Figure 5. Scheduler at disk i.

of blocks retrieved in a disk should be minimized, because the most heavily loaded disk determines the number of clients that can be serviced simultaneously. We delay the start point of service until the maximum number of blocks retrieved in each disk is minimized. The worst case service latency is r rounds. The look-ahead parameter r presents a tradeoff between disk load balancing and service latency. We assume $r = d$ in the rest of this paper because it is worthwhile to increase the number of concurrent clients at the expense of acceptable service latency in VOD servers. We present an experimental result in Section 5 which shows that the number of concurrent clients increases significantly at the expense of acceptable service latency. When multiple clients concurrently request the same video of the same resolution, we may employ a buffer sharing policy for reduced disk workload. That is, the scheduler assigns one video stream for the concurrent clients and later assigns new streams for the clients requesting interactive operations. For clients with different playback phases, we may also minimize the required disk bandwidth by exploiting sophisticated caching mechanisms such as interval caching [14].

4.1. Support for interactive operations

Interactive operations are essential for VOD services. Clients are likely to perform VCR-like operations on video they are watching, such as pause, resume, fastforward, rewind, and slow playback. Fast scan operations, namely fastforward and rewind, should be treated carefully because they require additional server resources. In general, two approaches support them: encode separate streams and skip frames. The first approach needs extra storage space and the second approach may lead to load balancing problems [22].

Our multi-resolution video server supports fast scan operations without any additional overhead by degrading the resolution level and retaining the data rate of the video stream. For example, let us assume that a fastforward operation is requested for a video stream with the fourth level resolution. If we lower the resolution level to the second level and the first level, the two-times-fastforward and the four-times-fastforward can be accomplished,

respectively, without any additional disk and network bandwidth. Shenoy and Vin [31] validate this idea by a scalable encoding technique in which the low-resolution-based substream provides acceptable video quality for scan operations. For the case of the lowest resolution level where it is impossible to degrade the level, a segment skipping scheme [9] can be integrated with our scheme. We choose the segment skipping approach for two reasons. First, it can be easily integrated into our scheduling scheme simply by changing the *direction* parameter. And second, segment skipping achieves storage efficiency compared with the separate stream approach.

The scheduler in figure 5 can be updated in order to support interactive operations as follows. We assume that an interactive operation is requested to $V_j = (k_j, s_j, K_j, StartDisk_{V_j}, direction)$. All interactive operations are supported simply by reconstructing the input parameters of V_j, so that the service delay is also one round T_R as similar to the re-negotiation of resolution levels.

fastforward The new input parameters are given by $V'_j = (k'_j, s_j, K'_j, StartDisk_{V_j}, direction')$, where $k'_j = k_j/m$, $K'_j = mK_j$, and $direction' = \alpha$. In this case, we can achieve $m \cdot \alpha$-times fastforward.[5] When $\alpha > 1$, the segment skipping scheme is employed. For instance, when four-times fastforward is requested to $V_j = (4, 1000, 1, 0, 1)$ that is being played back with forth-level resolution, we can obtain $V'_j = (1, 1000, 4, 0, 1)$. In addition, when $V_j = (1, 2000, 1, 0, 1)$ (i.e. normal playback with first-level resolution), the segment skipping scheme is incorporated by $V'_j = (1, 2000, 1, 0, 5)$ for five-times fastforward. As mentioned above, the segment skipping scheme leads to disk load imbalance when α and d have the least common multiple (LCM). For example, consider a video server having four disks in figure 4(a). For two-times fastforward of $V_1 = (1, 0, 1, 0, 1)$, $(1, 0, 1, 0, 2)$ is given to V'_1. Then the server retrieves a sequence of $C_0^0, C_0^2, C_0^4, C_0^6, \ldots$, so that disk 0 and disk 2 will handle all the retrievals. This problem can be solved by selecting α such that α is relatively prime to d [20].

rewind This is equivalent to fastforward with $direction' = -\alpha$.

slow playback Reducing the number of segments K_j accomplishes the slow playback. Specifically, $K'_j = K_j/m$ for m-times slow motion. When $K'_j < 1$, V_j is excluded from the input list of the scheduler until $L_j \geq 1$, where L_j (initially K'_j) is increased by K'_j in each round and decreased by one when $L_j \geq 1$.

pause and resume The scheduler excludes V_j on pause and includes V_j again on resume.

4.2. Admission control

A video server must employ an admission control algorithm to decide whether a new client can be serviced without violating the real-time requirements of clients already being serviced. Since a CBR video stream V_j produces a constant disk workload (n_j blocks) in each round, we can employ a simple admission control algorithm which checks if all the blocks ($\sum_{j=1}^{N} n_j$) for N streams can be retrieved in a round. For VBR streams, the simple algorithm may use either $n_j = \max_{0 \leq i < s}(n_j^i)$ or $n_j = \text{avg}_{0 \leq i < s}(n_j^i)$, where n_j^i is the number of blocks to be retrieved in the i-th round. However, this results in either under-utilized or over-utilized resource at the server.

Admission control algorithms for VBR streams may be classified into two categories: statistical and deterministic. The first approach exploits the bit rate statistics of video streams and the second approach does the specific knowledge of the bit traces of video streams. Vin et al. [34] propose a statistical admission control algorithm with a mechanism enforcing statistical service guarantees. They compute the overflow probability, which is the probability that the service time for a single disk access exceeds the round duration, by statistically determining the total number of blocks in a round and empirically measuring a distribution function for the service time. Chang and Zakhor [6] calculate the probability of overload by integrating the probability density function of the aggregated resource required by all clients beyond a given threshold limit. The threshold limit is computed from a single disk performance analysis on their data placement schemes. Makaroff et al. [24] propose a deterministic admission control algorithm based on the stream block schedule which contains the number of blocks to be retrieved in each round. The admission of a new stream is accomplished by merging the stream block schedule with the existing one and checking a system overflow throughout the length of the service. The deterministic admission control algorithm provides a tight and safe bound for the admission, but its complexity is relatively high.

We now describe an admission control algorithm in our multi-resolution video server. Assume that client j for $1 \leq j \leq N-1$ is being serviced with k_j-level resolution of V_j and a new client N requests V_N with k_N-level resolution service. Since each disk must retrieve all the components scheduled in figure 5 every round for the deterministic service guarantee, the following inequality must be satisfied in each disk:

$$n(\mathcal{D}_i) \times B \leq T_R \Phi, \quad 0 \leq i < d, \tag{6}$$

where $n(\mathcal{D}_i)$ denotes the number of blocks required to store $\mathcal{D}_i$. In Eq. (6), $n(\mathcal{D}_i)$ can be calculated deterministically in CBR multi-resolution video streams, but $n(\mathcal{D}_i)$ varies from round to round in VBR case. For VBR streams, we attempt to estimate an upper bound n_{upper} of $n(\mathcal{D}_i)$ statistically such that

$$P_{overflow} = P[n(\mathcal{D}_i) > n_{upper}] < \varepsilon. \tag{7}$$

A new client N is admitted if the following inequality holds.

$$n_{upper} \times B \leq T_R \Phi \tag{8}$$

For the statistical estimation of the total number of blocks retrieved in a round for all clients, Vin et al. use the central limit theorem and Chang and Zakhor compute the probability density function (pdf) by the convolution of each individual pdf for a video request. They assume that, however, all the blocks are serviced in a single disk. In a disk array environment where data blocks are serviced across multiple disks, their approach may be incorrect. We further describe how to estimate $n(\mathcal{D}_i)$ in the next section along with experiments.

The inherent feature of our multi-resolution video server enables the server to renegotiate the service resolution level with clients failed in the admission control. The server can present a lower resolution level which satisfies Eq. (8). Furthermore, if it is permissible to degrade

Table 1. The average bit rate (Mbps) of each resolution level for trace data.

Resolution level	1	2	3	4/8	5/9	6/10	7/11
MPEG-1 (30fps)	0.18	0.8	1.5				
MPEG-2 (24fps)	0.32	1.152	3.008				
3-D subband (24fps)	0.190	0.253	0.316	0.380	0.506	0.633	0.760
				0.887	1.013	1.140	1.330

the resolution level of existing clients, more clients can be serviced. Transient degradation may be required for the rounds in which the actual number of blocks to be retrieved is greater than n_{upper}. If the scheduler in each disk detects the overflow, it degrades the service level uniformly across all the clients until Eq. (8) is satisfied.

It is noteworthy that according to Eq. (8) the buffer requirement of our server is $2n_{upper}B$ per disk. This value is much smaller than that of the static policy which allocates the worst-case fixed-size buffer to each client.

5. Experimental evaluation

In this section, we evaluate our schemes through experiments with trace data generated from actual scalable video streams. As mentioned in Section 2.1, we consider three VBR scalable compression techniques: MPEG-1 with temporal scalability, MPEG-2 with spatial and SNR scalability, and 3-D subband coding scheme. Table 1 shows the average bit rate of each resolution level for three kinds of trace data. To construct our multi-resolution video stream, the round length T_R should be determined first. The round length provides trade-off between disk throughput and buffer requirement. The choice of optimal round length is beyond the scope of this paper. Chang and Zakhor [6] suggest that the total system cost is minimized at T_R of one second from cost analysis and many other works assume one second for T_R [4, 34]. We also choose one second for T_R.

5.1. Disk load balancing

First, we validate Eq. (5) which indicates that the disk workload for any resolution service for a given video stream is evenly distributed across all the disks even for VBR case. Figure 6 presents the number of blocks retrieved in each disk for 30 minutes ($l = 1800$). The value of the right-most bar in each graph is calculated from Eq. (5). As shown in figure 6, our data placement scheme guarantees the disk load balancing for a video service.

Next, to explore the actual behavior of our multi-resolution video server, we have created an event-driven simulator written in C with SMPL [23] libraries. The simulator models the server including data placement and retrieval. Along with three types of trace data for multi-resolution video streams in Table 1, the server is assumed to have eight disks and to store 24 video streams (eight for each type). The video streams are placed on disks according to our data placement scheme with different starting points ($StartDisk_V$). We assume that each client randomly chooses a video stream and resolution level.

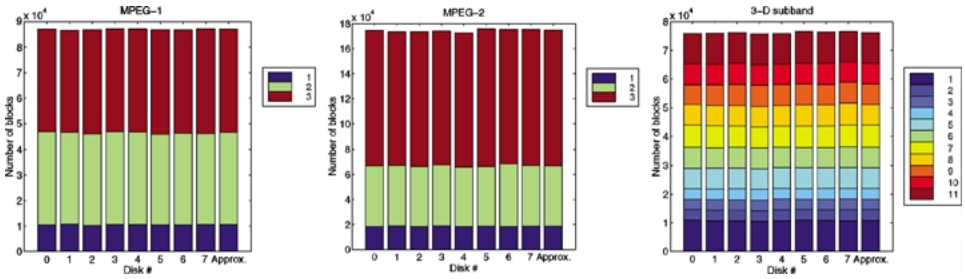
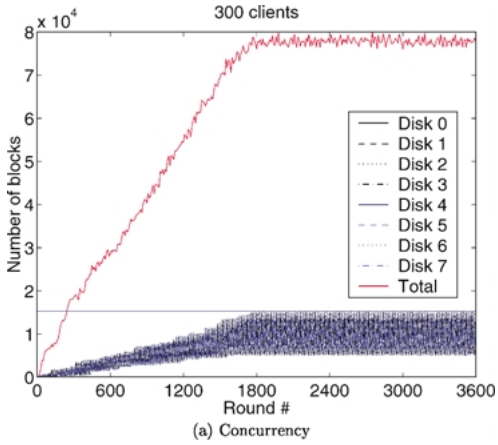


Figure 6. The distribution of disk workloads for a given resolution service.

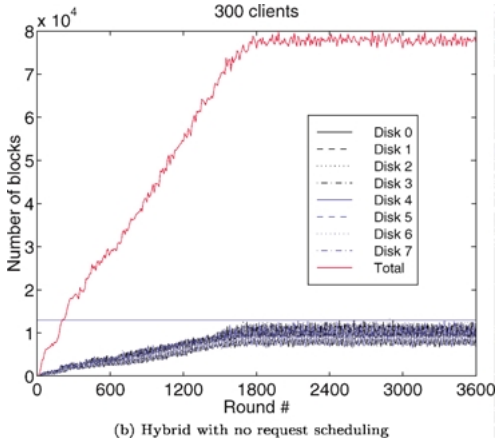
Figure 7 presents the number of disk blocks retrieved from a disk in each round for 300 concurrent clients. For the first 30 minutes (1800 rounds), clients arrive,[6] while the services continue for the next 30 minutes. First, figure 7(a) shows the distribution of disk workload when we adopt the concurrency model for the striping strategy in figure 3(a). The fluctuation of workloads in each disk is very large from round to round. This occurs because the disk workload is not evenly distributed across the disks in a given time point. Next, the result of the hybrid data placement scheme proposed in this paper is depicted in figure 7(b). The fluctuation of disk workload becomes smaller compared with figure 7(a). This indicates that the proposed hybrid data placement scheme can serve more clients since the most heavily loaded disks determine the number of concurrent clients in VOD servers. Finally, at the expense of service latency, we schedule the start point of services to minimize the maximum number of blocks retrieved in a disk (request scheduling in Section 4). By delaying the start point of service, we can evenly distribute the disk workloads in each round, so that the variation of workloads in a disk decreases significantly, as shown in figure 7(c). It should be noted that the variation of workloads in a disk is smaller than that of the total workloads. It is the most heavily loaded disk that determines the number of concurrent clients. So, small variation of workloads in a disk means increased concurrency.

5.2. *Admission control*

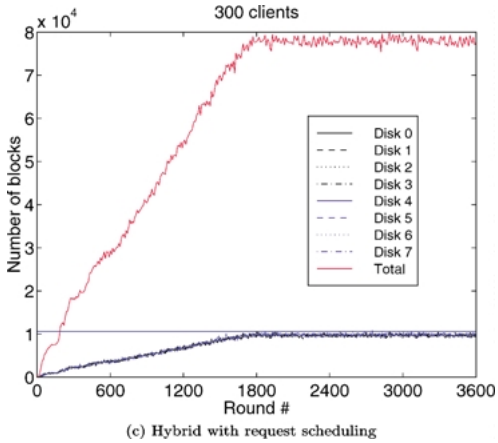
In Section 4.2, we described our admission control strategy. Since the number of blocks to be accessed at disk i in each round, or $n(\mathcal{D}_i)$, varies in VBR streams, we intend to estimate an upper bound n_{upper} from Eq. (7) for the statistical service guarantee. First, we take two existing approaches for a single disk system: central limit theorem [34] and convolution [6]. Let a random variable n_j denote the number of blocks to be accessed in each round for client j. The total number of blocks for N clients is given by $n = \sum_{j=1}^{N} n_j$. Using the central limit theorem, Vin et al. [34] estimate the distribution function of n as a normal distribution with $\mu_n = \sum_{j=1}^{N} \mu_{n_j}$ and $\sigma_n^2 = \sum_{j=1}^{N} \sigma_{n_j}^2$, where μ_n and σ_n^2 denote the mean and the variation respectively. Chang and Zakhor [6] compute the pdf $f_n(x)$ by convolving $f_{n_j}(x)$ for $1 \leq j \leq N$. When all the blocks are serviced in a single service point (i.e. disk), both of the two approaches give the exact estimation as shown in figure 8(a).



(a) Concurrency



(b) Hybrid with no request scheduling



(c) Hybrid with request scheduling

Figure 7. The distribution of disk workload for 300 clients.

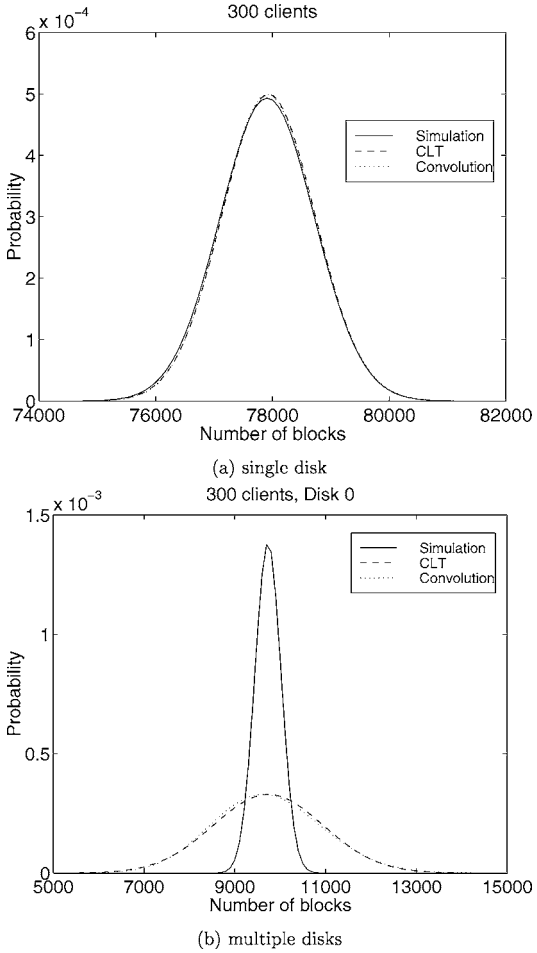


Figure 8. The result of estimation with existing schemes: The central limit theorem and convolution schemes for a single disk system should be updated in a disk array environment.

In a disk array environment where data blocks are serviced across multiple disks, however, they may not produce the proper estimation. In figure 8(b), the estimations of two approaches show a large difference from the simulation result. Observe that the disk request pattern for a video stream is repeated periodically with a period of d rounds. In figure 9, for example, the first four rounds are repeated as the service proceeds. Furthermore, from the view point of each disk, the components retrieved during d rounds contain each resolution level, although they are not in the same segment, for instance in figure 9, $\{C_0^0, C_1^3, C_2^2\}$ in disk 0 and $\{C_0^1, C_1^0, C_2^3\}$ in disk 1. This indicates that disk blocks for d rounds are perfectly distributed across all the disks throughout d rounds. In addition, the request scheduling evenly distributes the disk workloads for concurrent clients. Eventually, it is statistically true that n_j blocks are evenly distributed across d disks in each round. Thus, given a video

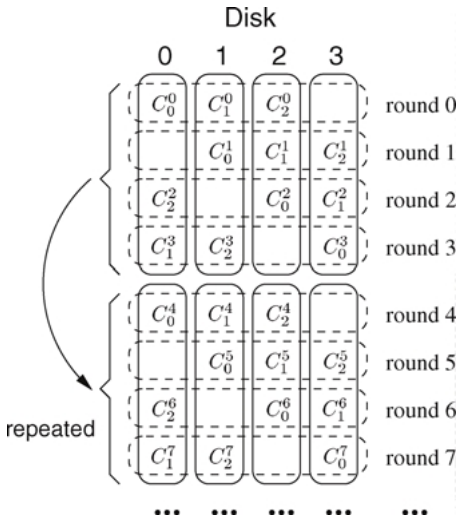


Figure 9. An example of the 3rd level resolution service: The variation of the number of blocks to be accessed at a disk during d rounds is much smaller than that of the number of blocks in each round.

stream, we can compute the mean and the variance for the number of blocks (n'_j) to be accessed at a disk in each round as follows:

$$\mu_{n'_j} = \mu_{n_j}/d, \quad \sigma^2_{n'_j} = \sigma^2_{n_j}/d. \tag{9}$$

Using the central limit theorem, $n(\mathcal{D}_i) = \sum_{j=1}^{N} n'_j$ approaches a normal distribution $N(\mu, \sigma^2)$ where $\mu = \sum_{j=1}^{N} \mu_{n'_j}$ and $\sigma^2 = \sum_{j=1}^{N} \sigma^2_{n'_j}$. From Eq. (7), then, n_{upper} is approximated as the smallest solution to the inequality,

$$\int_{n_{upper}}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \; < \; \varepsilon. \tag{10}$$

Finally, we can admit a new client N if n_{upper} satisfies Eq. (8). The statistics of the random variable n_j, which can be known a priori from the traces for the service of V with k-resolution at the time the video stream is stored, is as follows:

$$\mu_{V_k} = \frac{1}{s} \sum_{s=0}^{l-1} \sum_{c=0}^{k-1} b_c^s, \quad \sigma^2_{V_k} = \frac{1}{s} \sum_{s=0}^{l-1} \left(\left(\sum_{c=0}^{k-1} b_c^s \right) - \mu_{V_k} \right)^2 \tag{11}$$

We summarize the admission control algorithm in figure 10. The input parameters include the period of service round (T_R), the number of disks (d), the block size (B), the effective disk bandwidth (Φ), the upper bound of overflow probability (ε), and video streams which

```
1.   Procedure AdmissionControl
2.   input: T_R, d, B, Φ, ε, V_j, 1 ≤ j ≤ N
3.   output: true or false
4.   begin
5.       Get μ_{V_j}, σ²_{V_j} (Eq. (11)) from the pre-calculated table
6.       Calculate μ_{V'_j}, σ_{V'_j} from Eq. (9)
7.       Calculate μ = Σ_{j=1}^{N} μ_{V'_j} and σ² = Σ_{j=1}^{N} σ²_{V'_j}
8.       Calculate n_upper from Eq. (7) and Eq. (10)
9.       Check Eq. (8)
10.      if Eq. (8) holds
11.          return true
12.      else
13.          return false
14.  end
```

Figure 10. Admission control algorithm.

clients request ($V_j, 1 \leq j \leq N$). The algorithm determines whether the set of clients can be serviced within the given overflow probability ε.

Figure 11 exhibits that the proposed scheme gives an accurate estimation to the actual number of blocks required for the service. We compare n_{upper} calculated from Eq. (10) with the value measured from the simulation. As shown in figure 11, regardless of the number of clients and the number of disks, our admission control strategy precisely estimate the actual number of blocks to be accessed at a disk. This indicates that the server resources such as disk bandwidth and buffer memory can be fully utilized. In figure 12, the effect of the request scheduling is presented. By reducing n_{upper}, the request scheduling enables the server to efficiently provide services for more clients. The effect of the request scheduling becomes larger as the number of disks increases, since the disk load balancing is more significant on a large number of disks. The experiment result shows that the request scheduling reduces the required bandwidth by about 9.8 Mbps per disk, but the average service latency increases by 2.16 seconds. We believe that it is worthwhile to increase the concurrency at the expense of acceptable service latency.

5.3. Implementation

We have implemented a prototype of multi-resolution VOD services to realize the proposed schemes. For the quick implementation, we adopt the server architecture in figure 2(a) and MPEG-1 video with a hardware decoder. We developed the multi-resolution video manager as a server process in QNX real-time micro-kernel operating systems [27] on a Pentium PC with a AHA-1540CP SCSI adapter and four Quantum 850MB SCSI disks. As described in Section 2.1, MPEG-1 video streams are reconstructed into our multi-resolution video model in temporal dimension. In the first prototype of the server, the QoS level is provided with high, medium, or low. MPEG-1 video streams are parsed and separated by

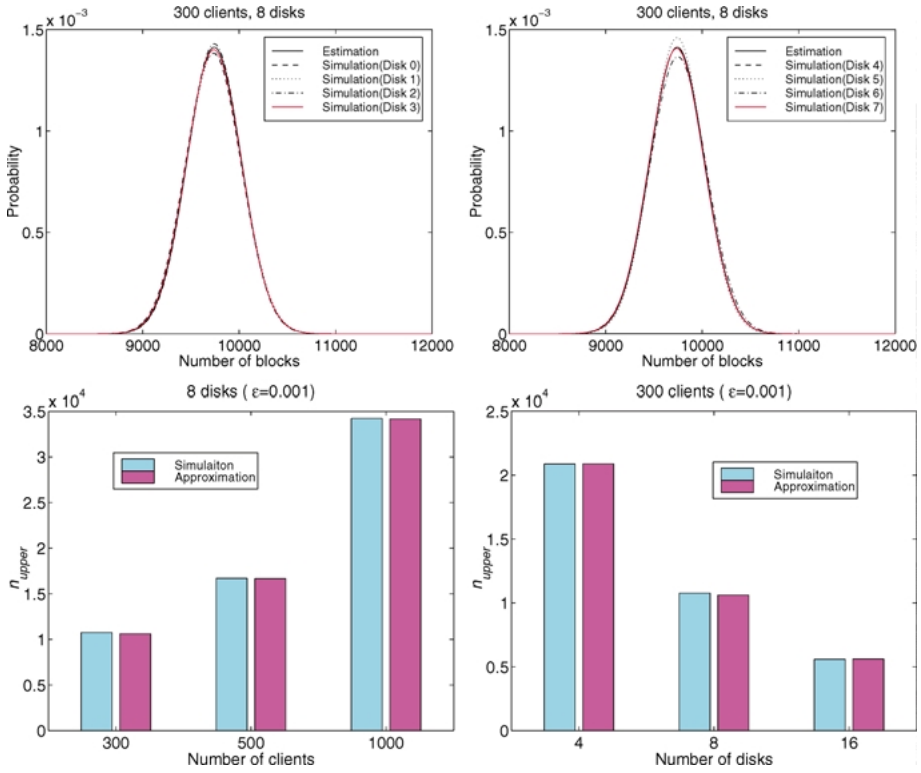


Figure 11. The result of estimation with the proposed scheme: The proposed scheme precisely estimate the number of blocks to be accessed at a disk.

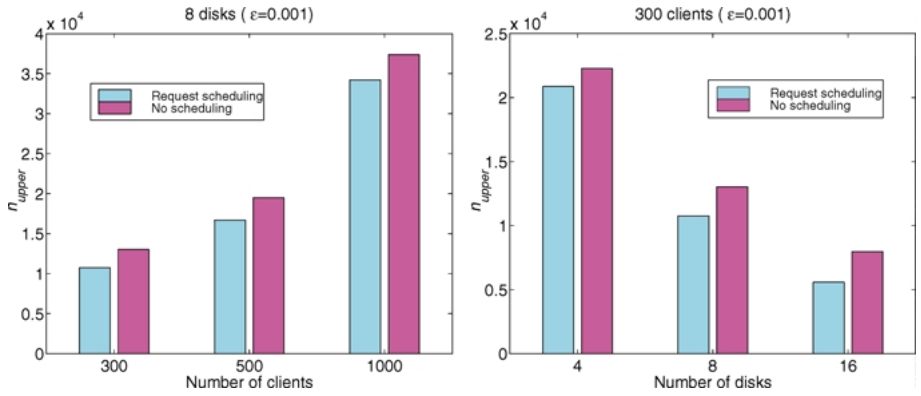


Figure 12. The effect of the request scheduling.

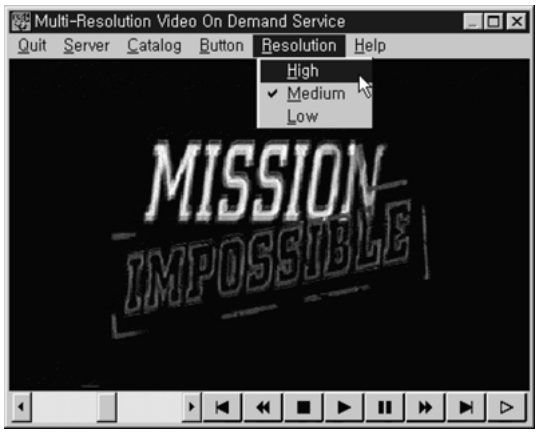


Figure 13. The client window in our prototype.

the frame type, and then, a segment is made up of a GOP. A set of the same type frames in the GOP constructs each component in a segment, for example, (I_1), (P_1, P_2, P_3, P_4), $(B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10})$. By the proposed layout scheme, the reconstructed MPEG-1 video is placed on disks. The multi-resolution video manager performs this work and provides the retrieval service of a given resolution level. During the retrieval, when necessary, the frame sequence is reorganized into the original sequence, so that the existing MPEG hardware/software decoder can work.

Based on the multi-resolution video manager, a VOD system has been developed. Figure 13 shows a window for the client program which runs on Windows95 with RealMagic hardware decoder.[7] Our prototype exhibits that the visual quality of the multi-resolution playback and fastforward playback is acceptable. This gives us more insights into our technique when extended to a practical environment.

6. Conclusions

A multi-resolution video server is able to provide multiple resolution services and has the benefits of heterogeneous client support, storage efficiency, adaptable service, and interactive operations support. In this paper, we have presented a framework for designing a multi-resolution video server. We proposed a general model for multi-resolution or scalable video and described how to build the model using the current scalable video compression techniques. Assuming a disk-array-based server model which ranges from a single computer equipped with multiple disks to a distributed or parallel server comprised of multiple nodes, we have presented the data placement and retrieval schemes.

The detailed simulation with the actual scalable video trace data have shown that the data placement and retrieval schemes achieve the disk load balancing during any resolution services and our admission control algorithm precisely estimate the actual disk workloads. This indicates that the proposed schemes enable the server to fully utilize its resources.

In addition, we have validated the proposed schemes by implementing a prototype for the multi-resolution VOD service. Extending the prototype to a practical environment remains to be done in the future.

Appendix A

Nomenclature

b	the number of disk blocks required to store a component
B	the block size of disks
C_c^s	the c-th component of s-th segment
$D(C)$	the disk which contains a component C
d	the number of disks in the server
$f_{n_j}(x)$	the probability density function of n_j
K	the number of segments retrieved in a round
k	the requested resolution level of the multi-resolution video stream
l	the length of a video stream
N	the number of clients
n_j	a random variable that represents the number of disk blocks retrieved in a round for client j
n_{upper}	an upper bound of $\sum_{i=j}^{N} n_j$
$n(\mathcal{S})$	the number of disk blocks required to store a set $\mathcal{S}$
Φ	the effective bandwidth of a disk subsystem
$R_{V,k}$	the average playback rate of V with k-level resolution
r	the look ahead parameter
$StartDisk_V$	the disk in which the starting point of V is stored
S_s	the s-th segment of a multi-resolution video stream
$size(C)$	the size of a component C
T_R	the round length
V	a multi-resolution video stream
$\mathcal{V}_{i,k}$	a set of components retrieved from disk i during k-level service of a video stream V
ε	the threshold limit for the overflow probability
z	the maximum resolution level of the multi-resolution video stream

Appendix B

Proof of Theorem 1: (i) $z = d$: Without loss of generality, we assume that $StartDisk_V = 0$. Then, for $V = \{C_c^s \mid 0 \le s < l,\ 0 \le c < z\}$, $\mathcal{V}_{i,k}$ is given from Eqs. (1) and (2) as follows:

$$\mathcal{V}_{i,k} = \left\{ C_c^s \;\middle|\; 0 \le s < l,\ 0 \le c < k,\ c = [i - s]_d \right\}. \tag{B.1}$$

Therefore, $|\mathcal{V}_{i,k}|$ is the number of s's, $0 \le s < l$, which satisfies

$$[i - s]_d < k. \tag{B.2}$$

Let $s = x \cdot d + y$, $(0 \le x < \lfloor \frac{l}{d} \rfloor, 0 \le y < d)$ and apply it to Eq. (B.2).

$$[i - x \cdot d - y]_d < k \tag{B.3}$$

If $l = m \cdot d$, for each x, $0 \le x < \lfloor \frac{l}{d} \rfloor$, the number of y's, $0 \le y < d$, which satisfies Eq. (B.3) is k. If $l \neq m \cdot d$, for each $x, 0 \le x < \lfloor \frac{l}{d} \rfloor$, the number of y's is k and for $x = \lfloor \frac{l}{d} \rfloor$, the number of y's is $\alpha, 0 < \alpha < d$. Hence, $|\mathcal{V}_{i,k}| = \lfloor \frac{l}{d} \rfloor \times k + \alpha$ $(0 \le \alpha < d)$.

(ii) $z < d$: This is equivalent to the case where $z' = d$ and $k \le z$. From the result of case i), $|\mathcal{V}_{i,k}| = \lfloor \frac{l}{d} \rfloor \times k + \alpha$ $(0 \le \alpha < d)$.

(iii) $z > d$: From Eqs. (1) and (2),

$$\mathcal{V}_{i,k} = \left\{ C_c^s \;\middle|\; 0 \le s < l,\ 0 \le c < k,\ c = [i - s]_d + a \cdot d,\ 0 \le a < \left\lceil \frac{z}{d} \right\rceil \right\}. \tag{B.4}$$

If $k \le d$, this case is equivalent to case i) because each disk retrieves one component in a segment. If $k > d$, each disk retrieves one or more components in a segment. From Eq. (B.4), $l \cdot \lfloor \frac{k}{d} \rfloor$ components $\{C_c^s \mid 0 \le s < l,\ c = [i - s]_d + a \cdot d, 0 \le a < \lfloor \frac{k}{d} \rfloor\}$ are retrieved and additional components $\{C_c^s \mid 0 \le s < l,\ c = [i - s]_d + \lfloor \frac{k}{d} \rfloor \cdot d < k\}$ are also retrieved. According to the result of case i), the number of the additional components is $\lfloor \frac{l}{d} \rfloor \times (k - \lfloor \frac{k}{d} \rfloor \cdot d) + \alpha'$ $(0 \le \alpha' < d)$. By integrating two terms, we can obtain the total number of components, $\lfloor \frac{l}{d} \rfloor \times k + \alpha$ $(0 \le \alpha < d)$. □

Notes

1. In this paper, the terms VOD server and video server are used interchangeably.
2. In Appendix A, we summarize the symbols used in this paper.
3. The adjacent disk of disk $d - 1$ is disk 0.
4. In Eq. (1), we define $y = [x]_d$ if $x = a \cdot d + y$, $0 \le y < d$, for all integer values.
5. In more detail, $\frac{R_{V_j,k_j}}{R_{V_j,k'_j}} \cdot \alpha$-times fastforward is achieved, where $R_{V,k}$ denotes the average playback rate of V with k resolution service.
6. We assume that clients arrive randomly since we focus on the stationary state where all the clients have arrived. Although the inter-arrival time may affect the efficiency of buffer cache and disk I/O algorithms, we neglect its effect because we assume the effective bandwidth for the performance of a disk subsystem.
7. Sigma Designs Inc.

References

1. D.P. Anderson, Y. Osawa, and R. Govindan, "A file system for continuous media," ACM Transactions on Computer Systems, Vol. 10, No. 4, pp. 311–337, 1992.
2. S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju, "Staggered striping: A flexible technique to display continuous media," Multimedia Tools and Applications, Vol. 1 No. 2, pp. 127–148, 1995.

3. A. Bogdan, "Multiscale (intrer/intra-frame) fractal video coding," in Proc. of IEEE International Conference on Image Processing, 1994.
4. W.J. Bolosky et. al., "The tiger video fileserver," in Proc. of International Workshop on Network and Operating System Support for Digital Audio and Video, 1996, pp. 97–104.
5. E. Chang and A. Zakhor, "Scalable video data placement on parallel disk arrays," in Proc. of IS&T/SPIE International Symposium on Electronic Imaging: Science and Technology, 1994, pp. 208–221.
6. E. Chang and A. Zakhor, "Disk-based storage for scalable video," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 7, No. 5, pp. 758–770, 1997.
7. M.-S. Chen and D.D. Kandlur, "Stream conversion to support interactive video playout," IEEE Multimedia Magazine, Vol. 3, No. 2, pp. 51–58, 1996.
8. M.-S. Chen, D.D. Kandlur, and P.S. Yu, "Optimization of the grouped sweeping scheduling (gss) with heterogeneous multimedia streams" in Proc. of ACM Multimedia'93, 1993, pp. 235–242.
9. M.-S. Chen, D.D. Kandlur, and P.S. Yu, "Support for fully interactive playout in a disk-array-based video server," in Proc. of ACM Multimedia'94, 1994, pp. 391–398.
10. M.-S. Chen, D.D. Kandlur, and P.S. Yu, "Using rate staggering to store scalable video data in a disk-array-based video server," in Proc. of IS&T/SPIE Symposium on Electronic Imaging Conference on Multimedia Computing and Networking, 1995, pp. 338–345.
11. T.-C. Chiueh and R.H. Katz, "Multi-resolution video representation for parallel disk arrays," in Proc. of ACM Multimedia'93, 1993, pp. 401–409.
12. J. Cho and H. Shin, "Scheduling video streams in a large-scale video-on-demand server," Parallel Computing, Vol. 23, No. 12, pp. 1743–1755, 1997.
13. A. Dan, D. Dias, R. Mukherjee, D. Sitaram, and R. Tewari, "Buffering and caching in large-scale video servers," in Proc. of IEEE CompCon'95, 1995, pp. 217–224.
14. A. Dan and D. Sitaram, "A generalized interval caching policy for mixed interactive and long video workloads," in Proc. of IS&T/SPIE Symposium on Electronic Imaging Conference on Multimedia Computing and Networking, 1996, pp. 344–351.
15. Y. N. Doganata and A.N. Tantawi, "A video server cost/performance estimator tool," Multimedia Tools and Applications, Vol. 1, No. 2, pp. 127–148, 1993.
16. D.J. Gemmell, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L.A. Rowe, "Multimedia storage servers: A tutorial," IEEE Computer Magazine, Vol. 28, No. 5, pp. 40–49, 1995.
17. A. Heybey, M. Sullivan, and P. England, "Calliope: A distributed, scalable multimedia server," in Proc. of USENIX 1996 Annual Technical Conference, 1996.
18. J. Hunter, V. Witana, and M. Antoniades, "A review of video streaming over the internet," White paper http://www.dstc.edu.au/RDU/staff/jane-hunter/video-streaming.html.
19. K. Keeton and R.H. Katz, "The evaluation of video layout strategies on a high-bandwidth file server," in Proc. of International Workshop on Network and Operating System Support for Digital Audio and Video, 1993, pp. 237–248.
20. T.-G. Kwon, Y. Choi, and S. Lee, "Disk placement for arbitrary-rate playback in an interactive video server," Multimedia Systems, Vol. 5, No. 4, pp. 271–281, 1997.
21. S.-W. Lau and J.C.S. Lui, "Scheduling and data layout policies for a near-line multimedia storage architecture," Multimedia Systems, Vol. 5, No. 5, pp. 310–323, 1997.
22. J.Y.B. Lee, "Parallel video server: A tutorial," IEEE Multimedia Magazine, Vol. 5, No. 2, pp. 20–28, 1998.
23. M.H. MacDougall, "Simulating Computer Systems: Techniques and Tools," MIT Press, 1987.
24. D. Makaroff, G. Neufeld, and N. Hutchinson, "An evaluation of vbr disk admission algorithms for continuous media file servers," in Proc. of ACM Multimedia'97, 1997, pp. 143–154.
25. R.T. Ng and J. Yang, "An analysis of buffer sharing and prefetching techniques for multimedia systems," Multimedia Systems, Vol. 4, No. 2, pp. 55–69, 1996.
26. S. Paek, P. Bocheck, and S.F. Chang, "Scalable mpeg2 video servers with heterogeneous qos on parallel disk arrays," in Proc. of International Workshop on Network and Operating Systems Support for Digital Audio and Video, 1995, pp. 363–374.
27. QNX, QNX Operating Systems Manual, QNX Software Systems Ltd., 1993.
28. A.L.N. Reddy and J. Wyllie, "Disk scheduling in a multimedia i/o system," in Proc. of ACM Multimedia'93, 1993, pp. 225–233.

29. J.M.D. Rosario and G. Fox, "Constant bit rate network transmission of variable bit rate continuous media in video-on-demand servers," Multimedia Tools and Applications, Vol. 2, No. 3, pp. 215–232, 1996.
30. P.J. Shenoy and H.M. Vin, "Efficient support for scan operations in video servers," in Proc. of ACM Multimedia'95, 1995, pp. 131–140.
31. P.J. Shenoy and H.M. Vin, "Efficient support for interactive operations in multi-resolution video servers," Multimedia Systems, Vol. 7, No. 3, pp. 241–253, May 1999.
32. D. Taubman and A. Zakhor, "Multirate 3-d subband coding of video," IEEE Transactions on Image Processing, Vol. 3, No. 5, pp. 572–588, 1994.
33. S.-R. Tong and Y.-F. Huang, "Study on disk zoning for video servers," in Proc. of International Conference on Multimedia Computing and Systems, 1998, pp. 86–95.
34. H.M. Vin, P. Goyal, A. Goyal, and A. Goyal, "A statistical admission control algorithm for multimedia servers," in Proc. of ACM Multimedia'94, 1994, pp. 33–40.
35. H.M. Vin and P.V. Rangan, "Designing a multiuser hdtv storage server," IEEE Journal on Selected Areas in Communications, Vol. 11, No. 1, pp. 153–164, 1993.
36. H.M. Vin, S.S. Rao, and P. Goyal, "Optimizing the placement of multimedia objects on disk arrays," in Proc. of International Conference on Multimedia Computing and Systems, 1995, pp. 158–165.
37. K.-L. Wu and P.S. Yu, "Increasing multimedia system throughput with consumption-based buffer management," Multimedia Systems, Vol. 6, No. 6, pp. 421–428, 1998.
38. M.-Y. Wu and W. Shu, "Scheduling for interactive operations in parallel video servers," in Proc. of International Conference on Multimedia Computing and Systems, 1997, pp. 178–185.

Jinsung Cho received his B.S. and Ph.D. degrees in Computer Engineering from Seoul National University, Korea, in 1992 and 2000, respectively. Currently, he is a professor of School of Electronics & Information at Kyung Hee University, Korea. His research interests include multimedia systems, real-time systems, and mobile computing & communications.

Minyoung Sung received his B.S. and M.S. degrees in Computer Engineering from Seoul National University, Korea, in 1995 and 1997, respectively. Currently, he is a Ph.D. candidate in School of Computer Science and Engineering at Seoul National University. His research interests include real-time communications, performance analysis, multimedia storage systems, and embedded Java systems.

Heonshik Shin received the B.S. degree in Applied Physics from Seoul National University, Korea, in 1973 and Ph.D. degree in Computer Engineering from the University of Texas at Austin in 1985. He is currently a professor of School of Computer Science and Engineering at Seoul National University. His research interests include real-time computing, distributed systems, and storage systems.