A New RAID-Disk placement method for Interactive Media Server with an Accurate Bit Count Control

Yo-Won Jeong, Seung-Ho Lim, and Kyu-Ho Park

Computer Engineering Research Laboratory,
Korea Advanced Institute of Science and Technology,
Daejon 373-1, Republic of Korea,
{ywjeong, shlim, kpark}@core.kaist.ac.kr

Abstract. In this paper, we propose a RAID-disk placement algorithm of coded video data and an efficient disk prefetching method to increase the number of clients who can be serviced interactive operations in the media server. Our placement policy is incorporated with a special bit count control method that is based on repeated tuning of quantization parameters to adjust the actual bit count to the target bit count. The encoder using this method can generate coded frames whose sizes are synchronized with the RAID *stripe size*, so that when various fast-forward levels are accessed we can reduce the seek and rotational latency and enhance the disk throughput.

1 Introduction

On-demand interactivity means that users can freely interact with the media server because video streams have extremely large data size, the high data retrieval bandwidth is required to support the interactivity to many users.

Generally, disk array technology is employed in multimedia server to provide the high disk bandwidth and satisfy real-time IO requirements [3][4][6]. In the disk array, disk striping is done by dividing the video data into blocks and storing these blocks into different disks. While storing these blocks into different disks, the proper placement algorithm should be considered in disk array to efficiently support the retrieval of such streams at different interactivity. X. Huang [3] studied the rate staggering method for scalable video in a disk array based video server. This method can reduce the buffer space and achieve better load balancing, but their allocation method did not consider the precise disk stirpe management and scalable encoding technique so that rate staggering method hardly apply to the real disk array. Shenoy [6] used the disk array to support the interactive operations in multi-resolution video. They present an encoding technique combined with placement algorithm to efficiently support interactive scan operation. Their variable-size block placement can reduce additional disk requests, but its management is very difficult in disk array.

In this paper, we propose an efficient placement algorithm to support the interactivity in media server, and develope the adaptive prefetching algorithm considering the

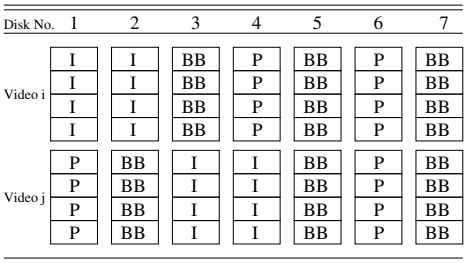


Fig. 1. Proposed Placement Algorithm for Interactive Media Server on Disk Array. Let the GOP structure be {IBBPBBPBB}

interactive operation. We have set up real interactive media server using SCSI disk array and linux operating system. Our placement policy is incorporated with an special bitcount control method, called $Fine\ Tuning\ of\ Tail\ Amount$, that repeatedly tunes quantization parameters to adjust the actual bit counts of video frames to the given target bit counts. The encoder using this method can generate coded frames whose sizes are synchronized with the RAID $stripe\ size$, so that when various fast-forward levels are accessed we can reduce the seek and rotational latency and enhance the disk throughput of each disk in the RAID system.

The rest of the paper is organized as follows. In Section 2, we present the efficient placement algorithm and the adaptive prefetching algorithm. The proposed encoding technique is presented in Section 3. In Section 4, we present performance results of our placement algorithm and encoding technique.

2 Efficient Placement for Interactive Operation

2.1 Placement Policy on Disk Array

In general, MPEG video stream consists of GOP (Group of Picture)s, and each GOP is represented as a sequence of I-, P- and B-frames. For example, if a GOP structure is {IBBPBBPBB}, the next-level fast-forward scan could be {IPPIPP..} which is not include any B-frames, and the next one is {II..} without any P-frames, and so on. Each sub-sequence for each fast-forward level accessed during a round is required to retrieve together from disks so that more client's real-time playbacks are guaranteed.

When server employs disk array to store the video streams, the server interleaves the storage of each video stream among disks in the array. The amount of data interleaved on a single disk, denoted as $stripe\ size$, is fixed when the disk array is configured. In that environment, to minimize the seek and rotational latency incurred by the requests, the same types of frame accessed during a round are in the same disks, and the different types of frame are stored in adjacent disks. However, the video streams made from conventional encoder do not have fixed frame size which is opposite to the fixed $stripe\ size$. It causes the additional disk requests at different fast-forward levels because frames are spread over more disks. Therefore, the special encoding technique is required to apply our placement policy. We will describe it in next section. Using this

special encoder, we can make the size of coded each I-, P- and B-frame is twice, same and half as the $stripe\ size$. Then, the GOP is stored as each I-, P- and B-frame consumes two, one, half stripes on disk array, and the next GOP is stored in next stripe level on disk array, and so on, as shown in Figure 1. At normal playback, the server should be retrieved from all disk array with evenly distributed number of frames. For K-level fast-forward, the server can skip every K-th disk to play out the video streams because the required frames to play fast-forward are separated beyond the disk boundaries. Notice that we can change the starting disk of the next video content for the load balancing of disk requests as shown in Figure 1.

2.2 Stream Classification

If streams stored by the above placement policy like Figure 1 have same frame rate, they have $same\ bitrate$ because all stripe sizes of the RAID system are fixed. However, because many tpyes of streams, having big picture size, small picture size, high quality or low quality, can be stored in one RAID system, the limitation of same bitrate of all steams is a serious weak point in our placement policy. To relieve this limitation, we purpose the stream classification as follows;

- Class A, the class of streams having high bitrate: Each I-, P- and B-frame consumes four, two and one consecutive stripes respectively. We set the ratio of bit count of I-, P- and B-frame is 4:2:1 because this ratio generally obtains best video quality [2].
- Class B, the class of streams having middle bitrate: Each I-, P- and B-frame consumes two, one and half consecutive stripes respectively. The ratio of bit count of I-, P- and B-frame is also 4:2:1. Therefore, the bitrate of this stream is $half$ of the stream of Class A.
- Class C, the class of streams having low bitrate: One I-frame consumes one stripe, but P-frames and B-frames cannot be synchronized with the stripe size. The ratio of bit count of I-, P- and B-frame cannot be same as other class. In this case, we cannot have gain for the fast-forward operation that I- and P-frames are scaned, but we still have gain for the fast-forward that only I-frames are accessed.

The placement of each frame in the RAID system is shown in Figure 2. In the stream of Class C, all I-frames consumes one stripe. For this, the sum total of sizes of P- and B-frames in one GOP is a multiple of the stripe size as shown in Figure 2-(c). the ratio of bit count of P- and B-frame has to be closest to 2:1. Therefore, To get the target bit counts of P- and B-frame, first, solve Equation (1) and (2), and select integer values closet to above solution satisfying Equation (2).

$$N_P C_P + N_B C_B = nS \qquad , n \text{ is apositive integer} \tag{1}$$

$$C_P / C_B = 2 \tag{2}$$

Where N_P and N_B are the numbers of P- and B-frames in one GOP, and C_P and C_B are the target bit counts of P- and B-frames. S is the stripe size.

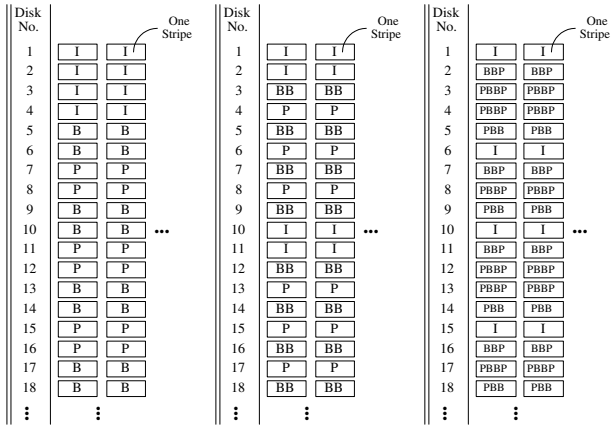


Fig. 2. The position of each frame of a stream: (a) Class A; the GOP structure is {IBBPBBPBBPBB}. (b) Class B; the GOP structure is {IBBPBBPBBPBB}. (c) Class C; the GOP structure is {IBBPBBPBBPBB} or {IBBPBBPBB}.

In Figure 2-(c), The GOP structure alternates {IBBPBBPBBPBB} and {IBBPBBPBB}. In the {IBBPBBPBBPBB} case, Equation (1) can be expressed as

$$3C_P + 8C_B = 4S\,. \tag{3}$$

If we set stripe size S to be 32KBytes, the solution of Equation (3) and Equation (2) is

$$C'_P = 128/7 \text{ KBytes } , C'_B = 64/7 \text{ KBytes}\,. \tag{4}$$

Therefore, The target bit counts of P- and B-frame, which are the integer values closest to Equation (4) and satisfying Equation (3) are

$$C_P = 18728 \text{ Bytes } , C_B = 9361 \text{ Bytes}\,. \tag{5}$$

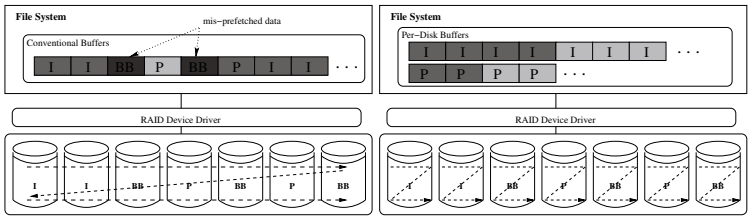
Note that the ratio of bit count of I-, P- and B-frame is 4:2.29:1.14. By the same way, in the {IBBPBBPBB} case, The target bit counts of P- and B-frame are

$$C_P = 19662 \text{ Bytes } , C_B = 9830 \text{ Bytes}\,. \tag{6}$$

The ratio of bit count of I-, P- and B-frame is 4:2.4:1.2. In both cases, The bit count ratios of frames do not large deviate from the ratio of Class A or B.

2.3 Per-Disk Prefetching Method

In general system, when the server retrieves data from disks, consecutive frames are retrieved ahead with the currently requested frames to increase disk throughput. We call these frames are prefetch frames or requests. Because the prefetching requests incur



(a) (b)

Fig. 3. Per-Disk Prefetching and Buffer Management; An example of Class B stream and X2 fast-forward operation. The light gray and dark gray represents well-prefetched data mis-prefetched data, respectively.: (a) Conventional prefetching requests, (b) Proposed per-disk prefetching requests

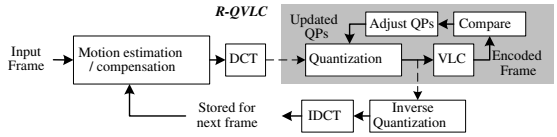


Fig. 4. The conceptual procedure of the R-QVLC scheme.

more data transfer and buffer space, it is important that proper amount of frames are retrieved. The conventional prefetching requests are generated across disk array, as shown in Figure 3-(a) because file system only know about the logically continuous allocation of video files. It causes prefetching requests make unnecessary data (B-frames) retrieval for fast-forward plays and would be overhead.

We propose the generation of prefetching requests for per disk, as shown in Figure 3-(b). When current request are retreived from one disk, our file system generates the prefetching requests to retrieve more data from the $same\ disk\ not\ other\ disks$. We call this method $per\text{-}disk\ prefetching\ method$. Because our placement policy separates the other frame types to other disks, the per-disk prefetching requests do not generate any unnecessary requests. Note that the per-disk prefetching method can be appliable to other class streams.

3 Accurate Bit count Control

In order to establish our placement policy, bit counts of all frames can be accurately controlled. However, conventional bit count control schemes cannot satisfy this requirement because of rate-distortion modeling errors and buffer controls for enhancing the subjective video quality [5]. We propose a method that exactly fixes each bit count of coded frames into given target bit count.

3.1 Fine Tuning of Tail Amount

Our bit count control method does not modify the process of the conventional encoding but work as a post-processing process after every one frame is encoded. If the actual

Test sequence	Rate Control Method	Avg. PSNR(dB)	Encoding Time(s)
Mobile	Conventional	27.50	116.6
	FTTA	27.13	140.1
Susie	Conventional	42.41	127.6
	FTTA	42.38	158.9

Table 1. The Average PSNR and Encoding Time

bit count of the coded frame is not equal to the target bit count, we pause the encoding process and start to adjust quantization parameter (QP)s of macroblock (MB)s. Figure 4 shows the conceptual procedure of proposed bit count control scheme. We will call this scheme repeated-quantization and variable length coding (R-QVLC). If the actual bit count is lower (or higher) than the target bit count, we increase (or decrease) QPs of appropriate MBs, and carry out QVLC in these MBs, and repeat this process. Detailed algorithm is called $fine\ tuning\ of\ tail\ amount$ ($FTTA$). There are three stages, rate-decreasing stage, rate-increasing stage and fine-tuning stage. If actual bit count or AB is bigger than the target bit count or TB, we start the rate-decreasing stage, otherwise, the rate-increasing stage.

In the rate-decreasing stage, we increase some QPs by 1 and perform the QVLC. After that, if AB is still bigger, we increase the number of adjusted MBs or BN by $twice$ for making AB approach to TB more fast. We repeat this process until AB becomes equal to or small than TB.

In the rate-increasing stage, the process is similar to the rate-decreasing stage except that QPs decrease by 1 instead of increase. In this stage, if AB becomes bigger than TB, we translate the fine-tuning stage.

In the fine-tuning stage, we restore QPs to previous values and, at this time, decrease BN by $half$ for fixing AB into TB. After that, we goto the rate-increasing stage.

Note that the direction of adjusting MB is the reverse of encoding direction.

4 Performance Evaluation

To evaluate our proposed methods we have developed the prototype interactive media server with real disk array storage system and prototype MPEG-2 encoder in Linux operating system. We implement a system for only Class B streams, but these results can be applied to other class.

First, we describe about the FTTA method. We use the MPEG-2 codec provided by the MPEG Simulation Group [2]. Encoding system consists of Pentium 4 3GHz CPU and 1GB main memory. We use 'Mobile' and 'Susie' sequences with 100 frames. The frame size and rate are 720x480 pixels and 30 frames/s respectively. When we encode a source stream, we set the GOP structure to $\{$IBBPBBPBB$\}$ and bitrate to 6Mbps. The default rate control method provided by the MPEG-2 codec is used as the conventional rate control. When we apply the FTTA, we set the target bitcounts of I-, P-, and B-frame to 512, 256 and 128Kbits for making constant bitrate of 6Mbps.

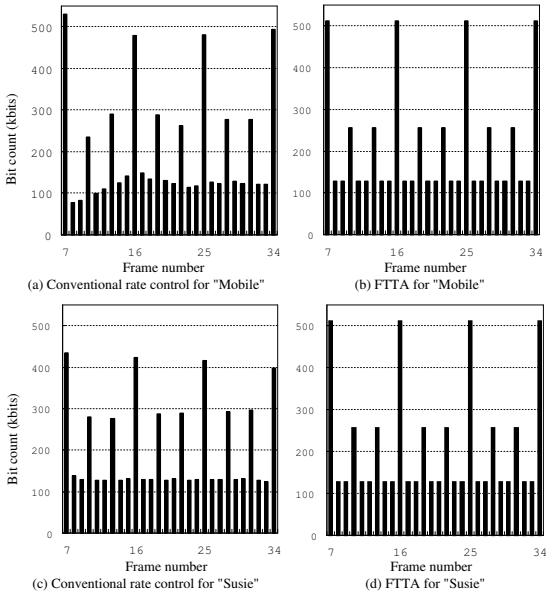


Fig. 5. The generated bit count of each frame

Figure 5 shows the encoding results for the two test sequences. *All* the generated bit counts by the proposed method are *equal* to the target bit counts. Average PSNRs and encoding time are summarized in Table 1. We can see that average PSNR is degraded by the FTTA (0.37 dB for 'Mobile' and 0.03 dB for 'Susie') because the FTTA tries to fix bit count into target bit count without considering the rate-distortion characteristics. However, this PSNR degradation shown in Table 1 is acceptable. In Table 1, encoding time increases by the FTTA because the FTTA is post-processing process after a conventional encoding process. These additional processing times are cost for accurate bit count control. Next, we have evaluated the placement policy and per-disk prefetching method. The evaluation environment is as follows. The server system consists of 2.4GHz intel Pentium 4 CPU, 512MB main memory and disk array with seven SCSI disks, model ST318304FC. The stripe size is set to be 32KB. Each client accessing the randomly selected video stream retrieves the frame-sequence with at a normal playback of 30 frames/s. The performance metric is average service time for one round playback duration as the number of clients increases. We have experimented with mixed fast-forward levels by varying the ratio between fast-forward levels. As shown in Figure 6-(a) and 6-(b), as the number of users increases, our placement policy, denoted as MSR (Media Synchronized RAID), gives better performance because of reducing the disk requests. Moreover, the average service time retrieved from disk in our placement policy with the per-disk prefetching method is much smaller than others which use conventional prefetching method. This is because per-disk prefetching method retrieves

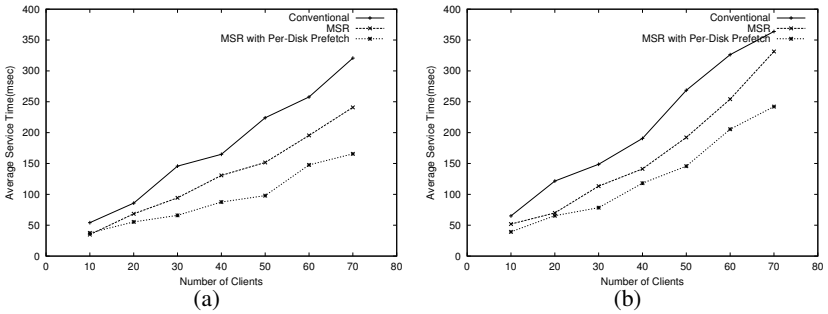


Fig. 6. Average Service Time for mixed fast-forward level: (a) X1 : X2 : X4 = 50 : 30 : 20, (b) X1 : X2 : X4 = 70 : 20 : 10

the current frame-sequence together with the near future frame-sequence in same disk request.

5 Conclusion

In this paper, we have presented an interactive media sever with media synchronized RAID storage system. We have proposed a placement algorithm and per-disk prefetching method to effectively support the interactive operation in media server. By doing this, when various fast-forward levels are accessed, we can reduce the seek and rotational latency and enhance the disk throughput of disks. We also propose a stream classification scheme for applying our placement algorithm into various types of streams. Our placement policy can be implemented with the proposed FTTA encoder. Though this encoder spends more time on encoding and yields small quality degradation, it can generate the coded video stream which is $synchronized$ with the RAID stripe size, so that we can significantly enhance the disk throughput and the average service time for each client connection as shown in our experimental results.

References

1. S. Lim, Y. Jeong, K. Park. Interactive Media Server with Media Synchronized RAID Storage System. *In Proceedings of ACM NOSSDAV 2005*, June 2005
2. Mpeg software simulation group: encoder/decoder. Version 1.1a, 1996.
3. X. Huang, C. Lin, and M. Chen. Design and performance study of rate staggering storage for scalable video in a disk-array-based video server. *In IEEE Transaction on Consumer Electronics*, 50(4):1119–1129, Nov 2004.
4. R. Katz, G. Gibson, and D. Patterson. Disk system architectures for high performance computing. *In Proceedings of the IEEE*, 77:1842–1858, Feb 1989.
5. J. Kwon and J. Kim. Adaptive video coding rate control for better perceived picture quality. *Proc. of APCC2003*, Sep 2003.
6. P. Shenoy and H. M. Vin. Efficient support for interactive operations in multi-resolution video servers. *ACM Multimedia Systems*, 7(3), 1999.