An SSD-Based Storage System for an Interactive Media Server Using Video Frame Grouping

Yo-Won Jeong, Youngwoo Park, Kwang-deok Seo, Jeong Ju Yoo, and Kyu Ho Park

For real-time interactive multimedia operations, such as video uploading, video play, fast-forward, and fast-rewind, solid state disk (SSD)-based storage systems for video streaming servers are becoming more important. Random access rates in storage systems increase significantly with the number of users; it is thus difficult to simultaneously serve many users with HDD-based storage systems, which have low random access performance. Because there is no mechanical operation in NAND flash-based SSDs, they outperform HDDs in terms of flexible random access operation. In addition, due to the multichannel architecture of SSDs, they perform similarly to HDDs in terms of sequential access. In this paper, we propose a new SSD-based storage system for interactive media servers. Based on the proposed method, it is possible to maximize the channel utilization of the SSD's multichannel architecture. Accordingly, we can improve the performance of SSD-based storage systems for interactive media operations.

Keywords: Video streaming service, interactive media service, media server, media storage system.

Manuscript received Nov. 18, 2011; revised Aug. 13, 2012; accepted Aug. 22, 2012.

This work was supported by the ETRI R&D Program of KCC (Korea Communications Commission), Korea [11921-03001, "Development of Beyond Smart TV Technology"].

Yo-Won Jeong (kaiforce@naver.com) was with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Rep. of Korea, and is currently with the Samsung Electronics Corporation, Suwon, Rep. of Korea.

Youngwoo Park (lucky.park@gmail.com) is with the Hyundai Motor Company, Seoul, Rep. of Korea.

Kwang-deok Seo (corresponding author, kdseo@yonsei.ac.kr) is with the Computer and Telecommunications Engineering Division, Yonsei University, Gangwon, Rep. of Korea.

Jeong Ju Yoo (jjyoo@etri.re.kr) is with the Broadcasting and Telecommunications Media Research Laboratory, ETRI, Daejeon, Rep. of Korea.

Kyu Ho Park (kpark@ee.kaist.ac.kr) is with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Rep. of Korea.

http://dx.doi.org/10.4218/etrij.13.0111.0715

I. Introduction

The recent growth of communication and multimedia computing technologies has led to a huge demand on servers, which must support an increasingly wide range of interactive media services for user created content (UCC), education, and entertainment.

Conventional interactive operations include *play*, *pause*, and *stop* functions. However, modern interactive media servers must include additional operations, as described below. First, the media server must have high performance for the many user-based write requests to upload UCC. Second, the media server must support such VCR-like services as fast-forward and fast-rewind at *multiple* speeds [1]. These VCR-like services are the most important features because the fast-forward and fast-rewind are the operations that are most frequently used aside from the play and stop operations. Third, the variety of user devices (such as a high performance PC, notebook, or mobile phone) demands a video scalability service. These operations could be efficiently realized through such multilayer video coding as scalable video coding (SVC) [2]. However, because of high encoding overhead and the implementation complexity of SVC, SVC technologies have not yet been widely deployed in related industries. It is thus difficult to fully support all the scalability modes of SVC, such as spatial, quality, and temporal scalability. For this reason, this paper focuses on the interactive media servers that store such single-layer video streams as H.264 video consisting of I-, P-, and B-frames.

Single-layer video streams can support such interactive services as multilevel fast-forward, fast-rewind, and scalability by alternately skipping and delivering video frames [3]. Figure 1 shows an example of the approach. The play level indicates

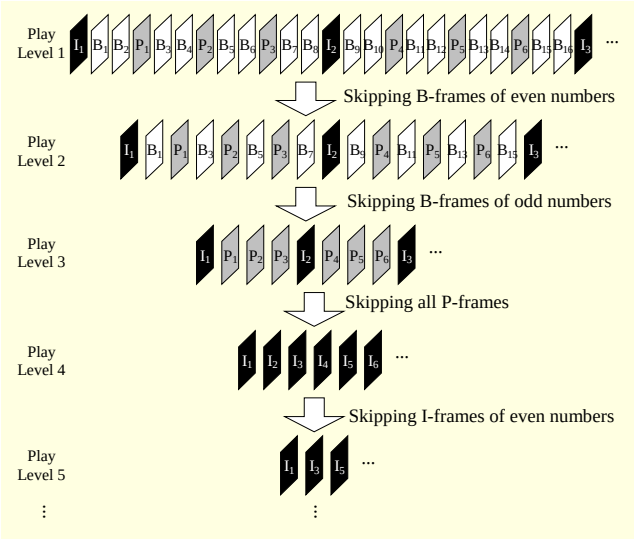


Fig. 1. Example of diverse access patterns resulting from various play levels.

the level of fast-forward, fast-rewind, or temporal scalability. Specifically, Play Level 1 means the operation of playing all video frames. If a client requests Play Level 2 or above, the media server can support the requested play level by transferring frames with a frame-skipping operation, as shown in Fig. 1. Even with this skipping, the client can still play the received video stream because of the interdependence characteristics that exist among I-, P-, and B-frames [3]. This method is not applicable to the spatial and quality scalabilities but applicable to the temporal scalability and fast-forward/fast-rewind services. Therefore, the target of this paper is to increase the efficiency of the method of adjusting play levels to nicely support the temporal scalability and fast-forward/fast-rewind services.

Note that the spatial and quality scalabilities using single-layer video streams can also be supported by storing multiple single-layer video streams encoded at different bitrates and resolutions for the same video title, which is a widely adopted approach in the industry.

For video streaming services, real-time video delivery with minimal delay is one of the most important considerations. If the network is fast and stable, delay and real-time performance are limited by server performance. However, the diverse media access patterns mentioned in Fig. 1 for interactive media operations lead to random access of a huge disk in the storage devices of the server [3]. Particularly, as the level of play increases, the random access rate also increases. The reason is that the distance between physical regions on the disk, where the frames to be accessed are stored, increases. It is thus difficult to concurrently ensure minimal delay and fulfill real-time requirements while serving many clients using HDD-based servers, which have poor random access performance.

Meanwhile, flash memory has come to the forefront in recent years, and flash memory-based solid state disks (SSDs) are rapidly broadening their share of the storage market due to the short random access time [4], [5]. While HDDs have low performance for seek times and rotational latency due to their mechanical limitations, SSDs have no mechanical components; thus, the SSD has only low seek latency due to the electronic components. Therefore, SSD-based video streaming servers supporting interactive media operations outperform their HDD-based counterparts.

In this paper, we propose a new SSD-based storage system for interactive media servers. We first analyze the throughput characteristics of random access for constant sizes of read requests in SSDs. Based on these throughput characteristics, we determine the request size that maximizes disk throughput and guarantees that read and write requests always have that size for all interactive operations based on the scheme of separated frame buffers.

The remainder of this paper is organized as follows. In section II, we review the backgrounds of interactive media operations and the characteristics of flash-based SSDs. Based on this review, we discuss critical design considerations of the storage system for media streaming servers. In section III, we describe the proposed method, which exploits the separated frame buffers. In section IV, we show extensive experiment results to validate the efficiency of the proposed method. We then discuss the results and conclude our work in section V.

II. Related Works

1. Interactive Media Operations

The write operations among interactive media operations always sequentially store entire video files to the server's disks. Thus, we do not need multiple levels for write operations, unlike what is required for read operations.

Stored video files are typically compressed by video compression standards, such as MPEG-x and H.26x. Generally, these compression standards use three types of video frames: intra-coded frame (I-frame), predictive-coded frame (P-frame), and bidirectionally predictive-coded frame (B-frame). I-frames are encoded independently and focus on removing redundancies existing within a picture. P-frames are encoded using predictions from the preceding I- or P-frame in the video sequence. B-frames are encoded using predictions from the preceding and succeeding I- or P-frames in the video sequence. Encoded video typically repeats the pattern of I-, P-, and B-frames, the so-called group of pictures (GOP), for the duration of an individual video stream [6]. Figure 2 shows an example of a GOP. The arrows indicate frame dependency relationships.

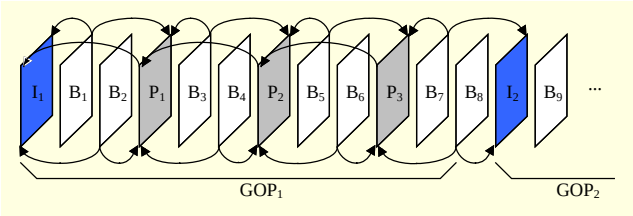


Fig. 2. Dependency structure of frames in GOP.

According to these dependency relationships, even if B-frames are skipped, we can decode all other frames. If we skip frames to decode a P-frame, we cannot decode upcoming P- or B-frames within the GOP. If we skip frames to decode an I-frame, we cannot decode the entire GOP.

Because the media server does not need to read whole video frames as its play level to minimize the data bandwidth of the server, it can serve various play levels requested by skipping (or jumping) the transfer of video frames. For example, in Fig. 1, Play Level 2 can be achieved by transferring frames while skipping even-numbered B-frames. Even with this skipping, the client can still play the stream because the skipped B-frames do not affect any other frames according to the dependency relationships among I-, P-, and B-frames. Play Level 3 can be achieved by transferring frames while skipping all even- and odd-numbered B-frames, and Play Level 4 can be achieved by additionally skipping all P-frames. Next, we can increase the play level by periodically skipping I-frames. By using these jumping read operations, we can minimize the data bandwidth of server disks and reduce the load on the server disks. However, if video files are written to disks in encoding order (which is a conventional way to store video files), the jumping read operations result in huge amounts of random access in the server disk [3]. It is thus difficult to simultaneously provide video data in real-time, with minimal delay, to many users using HDD-based servers.

To rectify this situation, some previous works proposed efficient placement of video data to HDD-based storage systems that support interactive media operations. We describe two technically feasible approaches related to the proposed media server design. Rangaswami and others [1] developed an interactive media proxy that transforms noninteractive broadcast or multicast streams into interactive ones. They carefully manage the disk device by considering disk geometry for allocation and by creating several stream files according to the various play levels. However, this method demands high storage capacity, and they did not consider additional write overhead.

Media synchronized RAID (MSR) [3] increases RAID disk performance for media servers by supporting interactive operations using the following two schemes. First, the authors synchronize sizes of all encoded frames to a RAID stripe size.

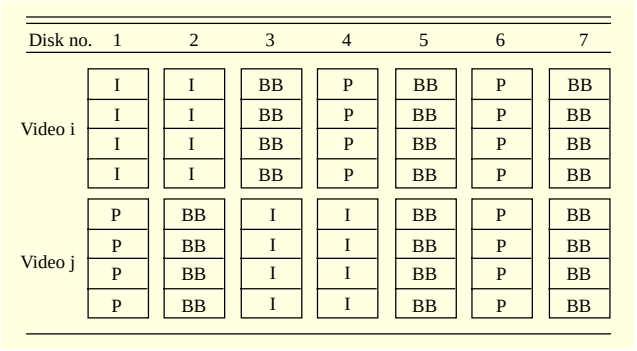


Fig. 3. Placement algorithm on disk array in media-synchronized RAID.

If the video streams that are composed of the stripe-sized synchronized frames are written to the RAID disks, I-, P-, and B-frames can be organized into the individual disks, as shown in Fig. 3. To synchronize the frames to the stripe size, an accurate bitrate control scheme was proposed [3]. Second, the authors proposed a per-disk prefetching scheme to increase disk performance. With these two schemes, MSR can increase RAID disk performance. However, the MSR method yields video quality degradation caused by fitting the number of bits of each frame to a predetermined size. Another limitation is that various types of video streams encoded with various bitrates cannot be supported.

To overcome the limitations of the previous efforts, we propose a scheme to group video frames by type to improve the performance of SSD-based storage systems.

2. NAND-Flash-Based Solid State Disk

Unlike HDDs, because flash memory can quickly access data regardless of physical location, SSDs can provide uniform and fast random access speeds—this is a key advantage of SSDs [7]. However, sustained read and write speeds for single flash memory are much slower than those of HDD disks. To compensate for that drawback, most SSDs employ a multichannel and multiway architecture, as shown in Fig. 4 [8], [9].

In this architecture, sequential write operations stripe the buffered data into *m* channels to maximize the channel utilization, as shown in Fig. 5. In each channel, because the flash memory uses one data bus, data must be transferred over the bus by time division. A pipeline scheme of the *n*-way architecture is adopted to increase the parallelism [8].

According to the striping and pipelining scheme, in a sequential write, data is spread over all flash memory with high probability. We define the *request size* as the data size requested from the host to the disk for a one-time read or write operation; as the read request size increases, the probability that spread

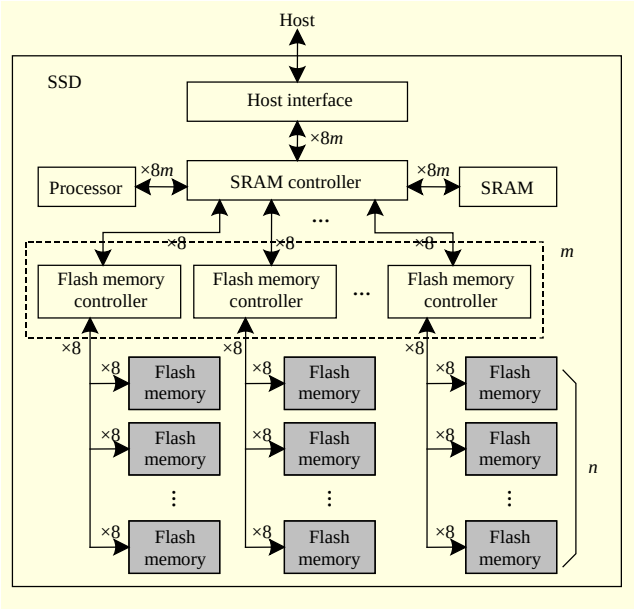


Fig. 4. *m*-channel and *n*-way architecture of SSD.

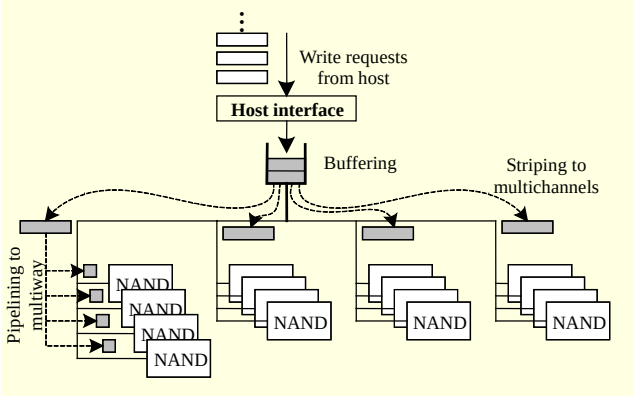


Fig. 5. General write operations in SSD.

data is simultaneously read from all flash memory increases. Therefore, in read operations of SSD, the request size is the most important factor affecting the read performance. To analyze the characteristic difference between SSD and HDD, we use the IOMeter benchmark program [10]. The benchmark program is employed in an Intel Pentium D 3.2-GHz processor with 2 GB of main memory. The operating system used in the test is Windows 7. The benchmark results display read/write throughput as a function of request size, as shown in Fig. 6. We measure throughput for both sequential and random access. We use two SSDs and one HDD storage device. SSD1 corresponds to Mtron MSD-SATA 3025 [11], SSD2 corresponds to Samsung MCBQE32G5MPPOVA [12], and HDD corresponds to Seagate 7200.10 [13]. Their basic specifications are summarized in Table 1.

As shown in Fig. 6, for data writing, *sequential* write

Table 1. Summary of specifications of sample SSDs and HDD.

Specifications	SSD1	SSD2	HDD
Model name	Mtron MSD-SATA 3025	Samsung MCBQE32G 5MPP	Segate 7200.10
Interface	S-ATA 2	S-ATA 2	S-ATA 2
Capacity	32 GB	32 GB	32 GB
Structure/type	4-ch., 4-way SLC*	4-ch., 4-way SLC*	7,200 rpm
Sustained write speed	80 MB/s	80 MB/s	NA
Sustained read speed	100 MB/s	100 MB/s	NA

* SLC: Single level cell.

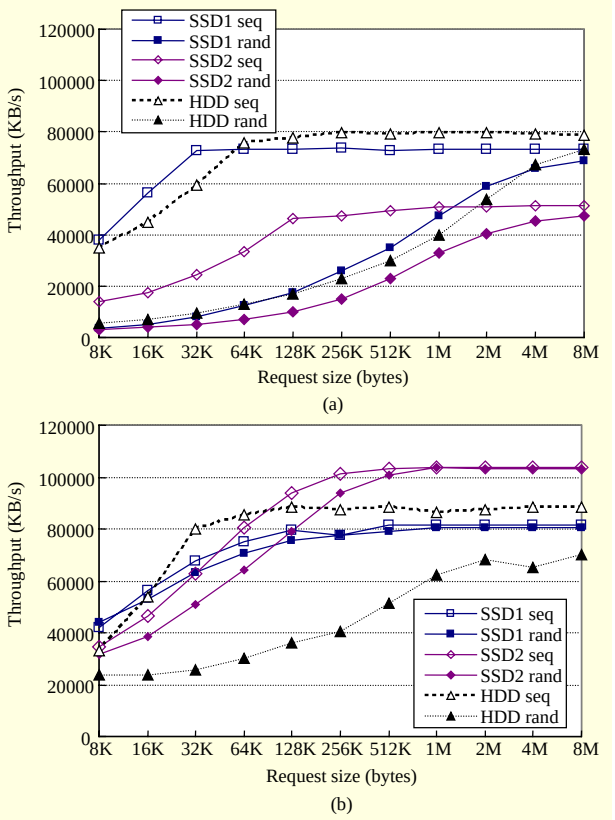


Fig. 6. Throughput measured by IOMeter benchmark program: (a) write test and (b) read test.

throughput is higher than that of *random* write for all storage devices. For both SSD and HDD, sequential write throughput becomes saturated for data request sizes in excess of 32 KB for SSD1, 64 KB for HDD, and 128 KB in SSD2. These saturation points mean that the system bus is fully operated with its maximum bandwidth. For SSDs, the full bandwidth is achieved by writing data to all flash memory in a fully parallel

fashion to maximize channel utilization. For reading data, throughput characteristics of SSD are markedly different from those of HDD. The throughput gap between sequential and random reads in SSD is smaller than in HDD, and only the request size affects the read throughput. Random read throughput becomes saturated for data request sizes in excess of 256 KB for SSD1 and 512 KB for SSD2. At these request sizes, we can maximize SSD channel utilization even if the access patterns are all random.

With due consideration of the above results, we summarize the following design strategies to increase the performance of storage systems in terms of throughput [14], [15].

i) For HDD write and read operations, one must favor sequential over random access as much as possible. If random access is required, the data request size must be large (up to several MB). However, the request size must be limited by the data size being read.

ii) For SSD write operations, similar to HDDs, one must favor sequential over random access as much as possible. If random access is required, the data request size must be made large.

iii) For SSD read operations, unlike for HDDs, differences in throughput between random and sequential operations are negligible when the throughput is saturated. An important consideration in terms of maximum throughput is the read request size to obtain saturated throughput.

III. Proposed Placement Method

1. Proposed Method with Separated Buffers

If we increase the request size to the throughput saturation point, we can maximize read throughput performance. To maintain that request size for all play levels, we must gather video frames according to type by replacing the video sequence data in video write operations.

Figure 7 outlines the basic concept of the proposed storing method. When encoding video streams, we separate encoded video frames based on frame type. For B-frames, we further separate frames according to their frame numbers (odd/even) because the play level can be distinguished by skipping either even B-frames or both even and odd B-frames. For I-frames, we also further separate frames according to their frame numbers (odd/even). Then, the play level using both even and odd I-frames and the play level using only odd I-frames can be distinguished. Therefore, five separated buffers are needed as shown in Fig. 7. All buffers have the same size (*S*). Whenever the size of data in one of the five buffers reaches *S*, the data of the occupied buffer is sequentially written to the SSD, and the sequential write pattern is thus preserved. Because the write request size is always the same as the buffer size, we can

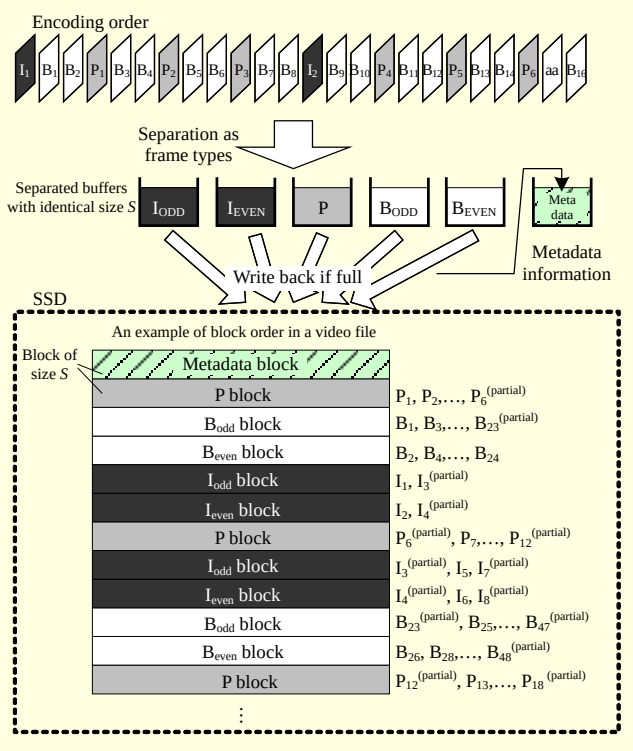


Fig. 7. Basic concept of proposed placement method.

achieve the maximum disk bandwidth if we select a buffer size larger than the request size of the saturation point in Fig. 6. We distinguish data blocks to be written according to frame type as follows: I_{odd} block, I_{even} block, P-block, B_{odd} block, and B_{even} block. Because the frame sizes can change, the last frame of each block might be fragmented. In Fig. 7, the written order of blocks and frames included in each block are examples of video content. The first P-block has full P-frames from P_1 to P_5 and the start of a fragmented P_6-frame. The end of the P_6-frame is placed in the next P-block. If the video ends before a buffer is full, we pad the buffer with zeros and write the data to the SSD. The proposed method guarantees that the read and write request sizes are always the same as the buffer size. For example, for Play Level 2, we need all the I- and P-frames and odd-numbered B-frames. Because the proposed method writes frames of the same type together to a block, as shown in Fig. 7, we can obtain necessary frames by reading I_{odd} blocks, I_{even} blocks, P-blocks, and B_{odd} blocks. For Play Level 5, we need only odd-numbered I-frames. We can obtain these frames by reading only I_{odd} blocks. For all play levels, because the request unit is always a block, all the request sizes are *S*. We next determine the buffer size *S* that maximizes the throughput.

2. Determining Sizes of Separated Buffers

There are two considerations for determining buffer size.

First, we must determine a buffer size that maximizes SSD throughput. Recall that, with the proposed method, write access patterns are always sequential, and read access patterns are always random position requests of the same size. With these access patterns, throughput increases as the buffer size increases up to a saturation point at which channel utilization is maximized (as described in subsection II.2). Therefore, we should set the buffer size such that channel utilization is maximized for both read and write operations. However, because the channel utilization algorithms of commercially available SSDs are generally undisclosed, we must determine the buffer size through experiments. Second, we must consider drawbacks associated with a large buffer size. As buffer size increases, play start time delay also increases. For example, if a user requests to start Play Level 1, the server must read all types of blocks (I_{odd} block, I_{even} block, P-block, B_{odd} block, and B_{even} block) at least once. In addition, with the increased buffer size, the frequency of reading irrelevant data also increases. For example, assume that I_1, I_3, and I_5 frames are written together into an I_{odd} block. Even if a user requests only the I_1 frame, we must read the entire I_{odd} block, which includes I_1, I_3, and I_5 frames. In this case, if the user stops playback before the I_3 frame is reached, the I_3 and I_5 frames that have already been read may not be needed. This problem can be applied to other types of frames. We must therefore determine an efficient buffer size that considers both SSD throughput and avoids potentially useless data reads.

To determine the efficient buffer size, we experimentally measure the write throughput and the read throughput as a function of buffer size. The experiment is set up as follows. For the write test, we assume that a number of users write their own video streams to the SSD using the proposed method. The total play times of written videos are randomly set from 5 to 20 minutes. For the read test, we assume that a number of users requested Play Level 4 (read all I-frames only) from their own video streams stored in SSD from the start to end frames. The read request sizes are the same as the buffer size used for write operations.

The experiment results are shown in Figs. 8 and 9. Figure 8 shows write throughput as a function of the size of separated buffers ranging from 8 KB to 4 MB for two types of SSDs and an HDD. For the large enough buffer sizes (approximately 128 KB) shown in that figure, SSD throughput saturates. Even if the buffer size exceeds this saturation point, throughput hardly increases because the maximum channel utilization is achieved. Due to the poor random access performance of the HDD, its throughput approaches the saturated throughput of SDD2 at a buffer size of several MB.

Figure 9 shows read throughput as a function of buffer size ranging from 8 KB to 4 MB for two types of SSDs and an HDD. As buffer size increases, the throughput increases and saturates for large buffer sizes. Even if the SSD1 and SSD2 specifications are similar, their read throughput characteristics are quite different. SSD2 throughput for small read requests is lower than that of SSD1. However, as the buffer size becomes larger, the increment of the throughput of SSD2 is greater than that of SSD1, and the saturated throughput of SSD2 is also higher. Throughput saturation starts at buffer sizes of around 128 KB for SSD1 and 256 KB for SSD2. Similar to the write requests case, channel utilization is maximized for read requests in excess of these saturation points. Thus, the throughput hardly increases. Due to the poor random access performance of the HDD, its throughput approaches the saturated throughput of SSD2 at a buffer size of several MB. The above analysis focuses on the case of Play Level 4. Similar read and write throughput characteristics could also be observed for other play levels.

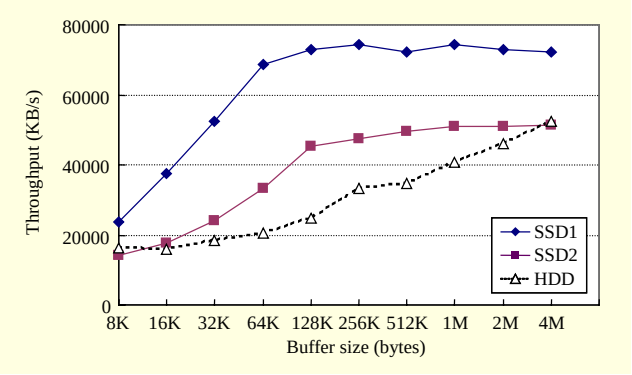


Fig. 8. Write throughput as function of buffer size for two types of SSDs and HDD.

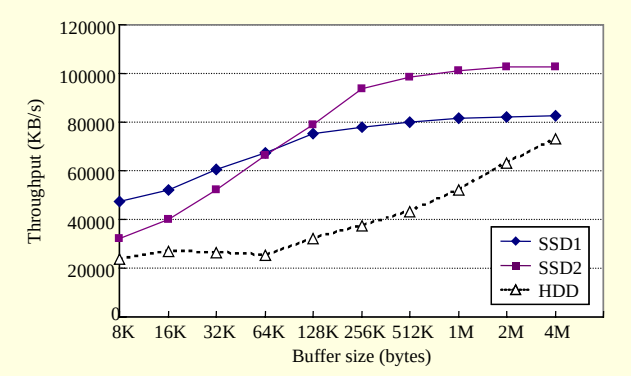


Fig. 9. Read throughput as function of buffer size for two types of SSDs and HD for Play Level 4.

Based on the above results, we set the separated buffer size to be 128 KB and 256 KB for SSD1 and SSD2, respectively. Even if the buffer size increases beyond 128 KB (for SSD1) or 256 KB (for SSD2), there is hardly an increase in the read throughput and the write throughput. The performance is rather

degraded by unnecessary reads. If we apply the proposed method to the HDD and want to obtain the same performance as that of the SSDs, we must set the size of separated buffers to be at least 4 MB, which results in that much more memory space being required than in SSD cases, and unnecessary reads increase.

3. Proposed Metadata Structure

The proposed method does not write video data in its encoding order. We must tailor the metadata structure to allow for fast searching of requested frames in the streaming file. We propose the following method for efficient metadata writing: whenever a frame block is written to the SSD, its metadata information is stored in another buffer whose size is also *S*, as shown in Fig. 7. If the metadata buffer is full, the metadata in the buffer is flushed to the SSD. If the video data ends before the metadata buffer is full, we pad the metadata with zeroes to the full buffer size and flush it to the SSD. The reason to write with a constant buffer size unit (instead of writing the complete metadata at one time) is that the total size of the proposed metadata can increase with the number of frames of video. For example, if a video file contains two-hour, 30-fps content, its total metadata size is about 1.2 MB (=30 fps × 2 hours × 3,600 seconds × 6 bytes per frame). If many users request several hundred video files, the metadata is read frequently from the disk because it is too large to hold all metadata in memory. By making the buffer size of the metadata the same as the frame buffers, we can always maintain the read and write request sizes of the disk access at a constant value.

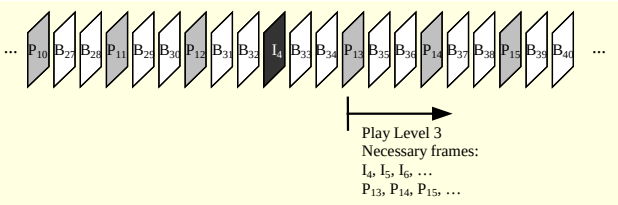
Figure 10 shows the proposed metadata structure. Each value in the table corresponds to the placement shown in Fig. 7. In Fig. 10, the "block number" denotes the *written order*. The position of a block in the stream file can be found by multiplying its block number by *S*. Content information is split (and written) into five categories: frame type, start frame index, number of frames, fragment, and link offset. Each frame has its own frame index that indicates the encoding order. We count the frame index for each individual frame type. For example, the frame indices of the first I-, P-, and B-frames are all 1, and the frame indices of the second I-, P-, and B-frames are all 2, and so on. The start frame index in the metadata indicates the frame index of the start frame of the current block. The number of frames is the number of frames included in the block, and the fragment bit identifies whether or not the last frame is fragmented. For example, according to Fig. 7, the first P-block (whose block number is 0) has full P-frames from P_1 to P_5 and the head of the fragmented P_6-frame. Therefore, frame type, start frame index, number of frames, and fragment value are set to P, 1, 6, and 1, respectively, as shown in Fig. 10. The link

Block no.	Frame type	Start frame index	No. of frames	Fragment	Link offset
0	P	1	6	1	5
1	B_{ODD}	1	11	1	7
2	B_{EVEN}	2	12	0	8
3	I_{ODD}	1	2	1	3
4	I_{EVEN}	2	2	1	3
5	P	6	7	1	4
6	I_{ODD}	3	3	1	5
7	I_{EVEN}	4	3	1	5
⋮	⋮	⋮	⋮	⋮	⋮

Block no.	Frame type	Start frame index	No. of frames	Frag-ment	Link offset
16 bits	3 bits	18 bits	5 bits	1 bit	5 bits

Total 6 bytes

Fig. 10. Proposed metadata structure.



Block no.	Frame type	Start frame index	No. of frames	Fragment	Link offset
0	P	1	6	1	5
1	B_{ODD}	1	11	1	7
2	B_{EVEN}	2	12	0	8
3	I_{ODD}	1	2	1	3
4	I_{EVEN}	2	2	1	3
5	P	6	7	1	4
6	I_{ODD}	3	3	1	5
7	I_{EVEN}	4	3	1	5
8	B_{ODD}	23	11	1	6
9	P	12	6	1	4
10	B_{EVEN}	26	11	1	6
⋮	⋮	⋮	⋮	⋮	⋮

Jump by link offset

Fig. 11. Separated buffer size variance for two types of mixed media operation services.

offset in the metadata is employed for fast searching. It records the distance between the number (or index) of the current block and the number of the next block that has frames of the same type as the current block. From Fig. 10, because the block number of the first P-block is 0 and the block number of the second P-block is 5, the link offset of the metadata of the first P-block is 5 (= 5–0). Similarly, because the block number of the first I_{odd} block is 3 and the block number of the second I_{odd} block is 6, the link offset of the metadata of the first I_{odd} block is 3 (= 6–3). Because there is a high probability (for any play level) that the same type of blocks are read within a short time period, the link offset information helps to search the necessary blocks quickly.

As shown in Fig. 11, if we assume that a user requests Play Level 3 from P_{13}, then, to satisfy the user's request, we need all

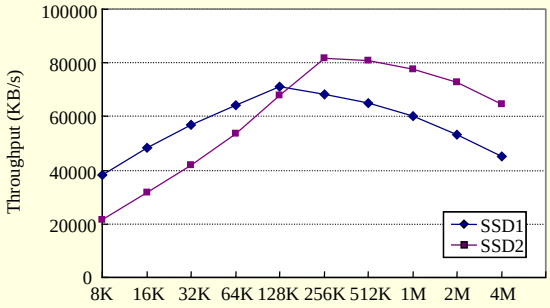
I-frames from I_4 and all P-frames from P_{13} because the decoding of P_{13} is dependent on I_4. In the metadata of Fig. 11, the 0-th block is the first P-block. This block does not include P_{13}, so we must find the next P-block by using the link offset. According to the link offset of the first P-block, we can determine that the block number of the second P block is 5. This block includes P_{13}, so we must read it. Subsequent P-blocks can be found by the link offset of their previous P-blocks. In the same way, we can find I_{even} and I_{odd} blocks as needed. We set the size of the metadata of each block to 6 bytes and allocate bits to each field, as shown in Fig. 10. This allocation strategy allows us to represent video streams that have 2^{16} blocks and 2^{18} B-frames because the bit lengths of the block number and the start frame index are 16 bits and 18 bits, respectively. For example, if the block size is 128 KB, the frame rate is 30 fps, and the GOP structure is IBBPBBP…, 2^{16} blocks implies about 8 GB (= 2^{16} × 128 KB) of video data, and 2^{18} B-frames implies 3.5 hours (= 2^{18} × 3/2 frames / 30 fps / 3,600 seconds) of running time. If we want to support a video with a longer running time, we can increase the length of the start frame index 18 bits to 26 bits (then the metadata size becomes 7 bytes). This one-byte increment is negligible relative to the video data size.

IV. Experiment Results

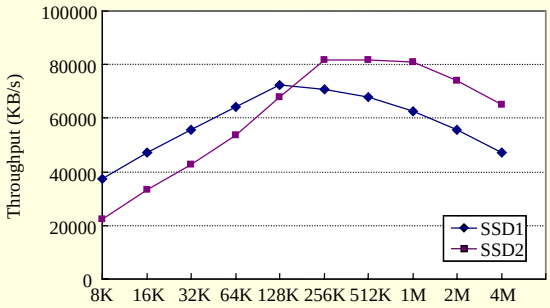
We perform various experiments to evaluate the performance of the proposed placement method. The number of simultaneous users is set to 20, 50, and 80. Each user is allowed to write or to read from an individual file. The pseudo code for the thread of data writing is as follows:

```
fd_ssdwr =
    open(video_file_name,O_RDWR|O_CREAT|O_DIRECT);
while(1) {
    if(buffer_fill>=PROP_ORWR_SIZE)
        wr_res =
        write(fd_ssdwr, buffer_data, PROP_ORWR_SIZE);
}
```

We use the O_DIRECT option of Linux to open a video file. Generally, for an application's write request, the Linux file system does not write the data immediately but gathers several requests and writes one time, which leads to changing the request size transferred to the disk. By using the O_DIRECT option, we can prevent the file system from changing the request size at write operations. If a buffer is filled with more than a predefined size (PROP_ORWR_SIZE), we flush the buffer's data of that size. Video files are encoded using H.264/AVC [16], [17] with a 2-Mbps bitrate and a 30-fps frame rate. The GOP pattern is set as IBBPBBPBBPBB...PBB..., and the number of frames in one GOP is 90 (corresponding to three seconds).



(a) Proportion of write, Play Levels 1, 2, 3, 4, and 5 is 0.1:0.5:0.1:0.1:0.1:0.1.



(b) Proportion of write, Play Levels 1, 2, 3, 4, and 5 is 0.1:0.3:0.15:0.15:0.15:0.15.

Fig. 12. Throughput as function of separated buffer size for two types of mixed media operation services.

In the experiment, we compare the proposed method as described in the previous section with a conventional method that places video stream data in its play order of Play Level 1. For the comparison, we measure the total throughput of the disk, which is defined as the total read and write data rate of a disk to serve all user requests. When many clients try to access video data, it is difficult for the server to guarantee real-time service for all clients because of the disk bottleneck. The higher the total throughput of disks, the greater the number of clients for whom real-time service can be guaranteed.

We analyze the obtained results by the following three steps:

i) First, we examine total throughput as a function of separated buffer size for mixed media operation services; this allows us to determine the optimal buffer size for maximum throughput;

ii) Second, we examine the total throughput as a function of the number of users for some individual media operations; we compare the proposed method with a conventional one; in this experiment, we can clearly identify the strong and weak points of the proposed method for different play levels;

iii) Third, we examine the total throughput as a function of the number of users for mixed media operation services.

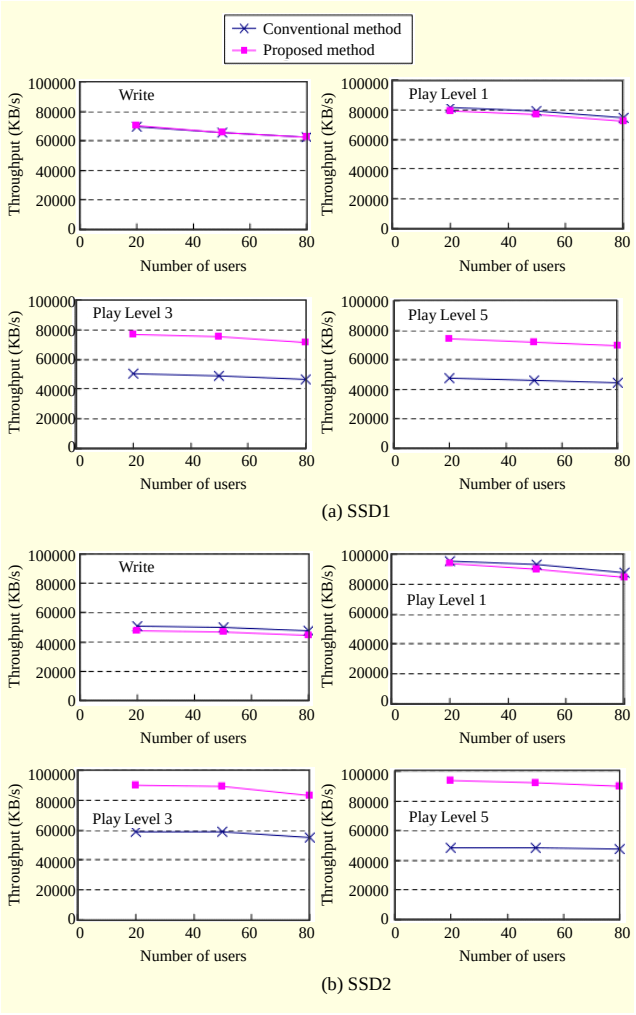


Fig. 13. Throughput for various media operation services: write and Play Levels 1, 3, and 5.

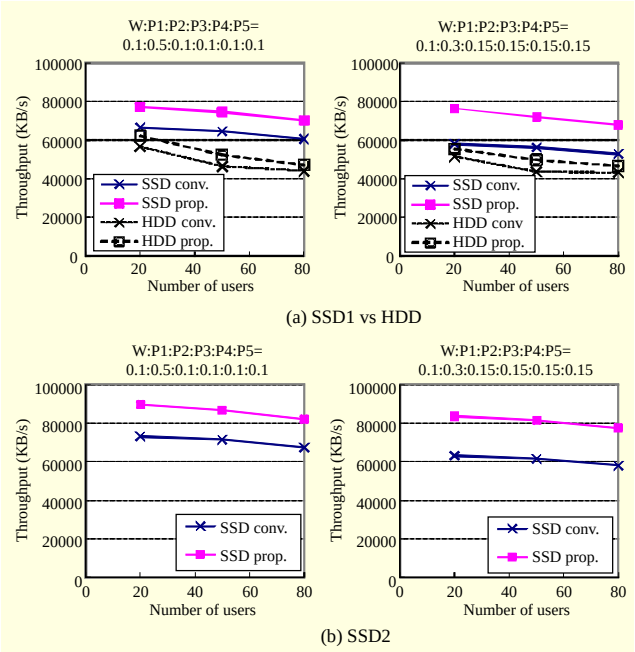


Fig. 14. Throughput for mixed media operation services: write and Play Levels 1, 2, 3, 4, and 5.

Figure 12 shows total throughput as a function of separated buffer size for two types of mixed media operations. The types of mixed media operations are determined by the proportions of media operations requested by clients. In the first type, we assume the request ratio for write and Play Levels 1, 2, 3, 4, and 5 to be 0.1 : 0.5 : 0.1 : 0.1 : 0.1 : 0.1. In the second type, we assume the request ratio for write and Play Levels 1, 2, 3, 4, and 5 to be 0.1 : 0.3 : 0.15 : 0.15 : 0.15 : 0.15. In each case, we increase the buffer size from 8 KB to 4 MB. Throughput increases with buffer size as channel utilization increases. However, throughput suffers somewhat in the case of a large buffer size as the frequency of irrelevant data read increases. Maximum performance is obtained for buffer sizes of 128 KB and 256 KB for SSD1 and SSD2, respectively, as shown in Fig. 12.

Figure 13 shows the measurement results of the total throughput for individual media operation services. For write operations, the size of the five separated buffers in the proposed method is set to 128 KB and 256 KB for SSD1 and SSD2, respectively. The size of the single buffer in the conventional method is 640 KB and 1,280 KB for SSD1 and SSD2, respectively.

As the examination in this work shows, the write throughput of the proposed method is similar to that of the conventional method because the write throughput does not significantly increase with the requested data size if this size exceeds 64 KB for SSD1 or 128 KB for SSD2, as shown in Fig. 8.

For Play Levels 1, 3, and 5, the read request sizes for both the proposed and conventional methods are the same as the size of the separated buffers. The service time of each media operation is five minutes. For Play Level 1, the throughput of the proposed method is slightly lower than that of the conventional method because of prefetching of unneeded data when the user stops play. The proposed method results in a greater amount of this data than the conventional method. However, as the service time of each media operation increases, the throughput gap decreases. For increased play levels, the proposed method significantly outperforms the conventional method because, in the conventional method, the number of dropped frames caused by skipping increases as the level of play increases.

Figure 14 shows the total throughput for mixed media operations. In this experiment, we show results for two cases. For the first case, we set the request ratio for write and Play Levels 1, 2, 3, 4, and 5 to be 0.1 : 0.5 : 0.1 : 0.1 : 0.1 : 0.1. For the second case, we set the request ratio for write and Play

Levels 1, 2, 3, 4, and 5 to be 0.1 : 0.3 : 0.15 : 0.15 : 0.15 : 0.15. Because the performance of the proposed method is slightly degraded in Play Level 1 and increases as the level of play increases, the proposed method shows much better performance in the second case. For the second case with SSD2, the throughput gain of the proposed method is more than 34%. The proposed method therefore exhibits better performance as the proportion of Play Levels 2 through 5 increases in the mixed media operations because the random access rate of the disk increases. SSD2 outperforms SSD1 with the proposed method because the saturated throughput of SSD2 is higher than that of SSD1.

In addition, Fig. 14(a) shows the total throughput of HDD. For HDD, we set the buffer size to 512 KB. The throughput of the proposed method is higher than that of the conventional method by 13% (case 1) and 11% (case 2). The proposed method is more powerful in SSD than in HDD because SSD achieves its maximum throughput with a much lower request size than HDD.

V. Conclusion

In this paper, we proposed a new storage system to increase the number of users who can be simultaneously served in real-time interactive operations using SSD-based video streaming storage systems. Support of real-time interactive media operations, such as video uploading, video play, fast-forward, and fast-rewind, for as many users as possible, is an important issue. Considering that random access rates in storage systems are increasing dramatically with the number of users, serving multiple users simultaneously is difficult with conventional HDD-based storage systems since they suffer from poor random access performance. NAND flash-based SSD outperforms HDD in terms of random access. Based on throughput analysis, we determined a constant buffer size that maximizes throughput. We then generated requests with that size for all play levels. This constant request size was possible because we gathered and wrote video frames as their types by using the proposed placement method with separated buffers. Extensive simulations showed that the proposed method can significantly improve disk storage system throughput for interactive media operations.

References

[1] R. Rangaswami et al., "Fine-Grained Device Management in an Interactive Media Server," *IEEE Trans. Multimedia*, vol. 5, no. 4, Dec. 2003, pp. 558-569.

[2] J. Ohm, "Advances in Scalable Video Coding," *Proc. IEEE*, vol. 93, no. 1, Jan. 2005, pp. 42-56.

[3] S. Lim, Y. Jeong, and K. Park, "Interactive Media Server with Media Synchronized RAID Storage System," *Proc. NOSSDAV*, June 2005, pp. 177-182.

[4] E. Seo, S.Y. Park, and B. Urgaonkar, "Empirical Analysis on Energy Efficiency of Flash-Based SSDs," *USENIX HotPower*, Dec. 2008.

[5] J. Yoon et al., "Chameleon: A High Performance Flash/FRAM Hybrid Solid State Disk Architecture," *IEEE Computer Architecture Lett.*, vol. 7, no. 1, Jan. 2008, pp. 17-20.

[6] H. Wu, M. Claypool, and R. Kinicki, "A Model for MPEG with Forward Error Correction and TCP-Friendly Bandwidth," *Proc. NOSSDAV*, June 2003, pp. 122-130.

[7] S. Lee et al., "A Case for Flash Memory SSD in Enterprise Database Applications," *Proc. ACM SIGMOD*, June 2008, pp. 1075-1086.

[8] J. Kang et al., "A Multi-channel Architecture for High-Performance NAND Flash-Based Storage System," *J. Syst. Architecture*, Sept. 2007, vol. 53, no. 9, pp. 644-658.

[9] S. Park, *A Buffer Management Scheme for NAND Flash-Based Storage System Using Multi-Channel Architecture*, master's thesis, Korea Advanced Institute of Science and Technology, Daejeon, Rep. of Korea, Dec. 2008.

[10] Iometer Project. Available: http://www.iometer.org/

[11] MTRON "Product Specification," MSD-SATA3035, rev. 0.1, Nov. 2007.

[12] Samsung SSD MCBQE32G5MPP Specification. Available: http://discountechnology.com/Samsung-32GB-2-5-SSD-ATA-Hard-Drive-MCBQE32G5MPP#specs

[13] Seagate Data Sheet, "Barracuda 7200.10," Apr. 2007.

[14] N. Agrawal et al., "Design Tradeoffs for SSD Performance," *Proc. USENIX Tech. Conf.*, June 2009.

[15] H. Kim et al., "Development Platforms for Flash Memory Solid State Disks," *Proc. ISORC*, 2008.

[16] ITU-T Recommendation H.264 and ISO/IEC 14496-10 AVC, "Advanced Video Coding for Generic Audiovisual Services," May 2003.

[17] Joint Video Team (JVT), H.264/AVC Reference Software, version JM 13.2. Available: http://iphome.hhi.de/suehring/tml/download/

Yo-Won Jeong received his BS in electronics engineering in 2000 and his MS and PhD in electrical engineering in 2002 and 2012, respectively, all from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Rep. of Korea. He is currently working at Samsung Electronics as a senior engineer. His research interests include video storage systems, video compression, and transmission.

Youngwoo Park received his BE, MS, and PhD in the Division of Electrical Engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Rep. of Korea, in 2004, 2006, and 2010, respectively. He is currently working for Hyundai Motor Company. His research interests include storage systems, automotive software platforms, and embedded virtualization.

Kwang-deok Seo received his BS, MS, and PhD in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Rep. of Korea, in 1996, 1998, and 2002, respectively. From Aug. 2002 to Feb. 2005, he was with LG Electronics. Since March 2005, he has been a faculty member in the Computer and Telecommunications Engineering Division, Yonsei University, Gangwon, Rep. of Korea, where he is an associate professor. Since Sept. 2012, he has been a courtesy professor in the School of Electrical and Computer Engineering at the University of Florida, Gainesville, FL, USA. His current research interests include digital video broadcasting, mobile IPTV, scalable video coding, and protocol design for scalable video transport. He is a member of KICS, KSBE, IEEE, and IEICE.

Jeong Ju Yoo received his BS and MS in telecommunications in 1982 and 1984, respectively, from Kwangwoon University, Seoul, Rep. of Korea. He received his PhD in computing science from Lancaster University, Lancaster, England, UK, in 2001. Since 1984, he has been a principal member of the technical staff in the Next Generation Smart TV Research Department of ETRI, Daejeon, Rep. of Korea. He was the head of the MPEG Korea delegates from 2007 to 2009. He is currently the director of the Smart TV Media Research Team at ETRI. His research interests are in the areas of QoS, video coding, media streaming, and multiscreen service technology of smart TVs.

Kyu Ho Park received his BS in electronics engineering from Seoul National University, Seoul, Rep. of Korea in 1973, his MS in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 1975, and his Dr-Ing in electrical engineering from the Université de Paris XI, Orsay, France, in 1983. He has been a professor in the Department of EECS, KAIST, since 1983. He was the president of the Korea Institute of Next Generation Computing from 2005 to 2006. His research interests include computer architecture, file systems, storage systems, ubiquitous computing, and parallel processing. Dr. Park is a member of KISS, KITE, the Korea Institute of Next Generation Computing, IEEE, and ACM.