Video File Allocation for Wear-Leveling in Distributed Storage Systems with Heterogeneous Solid-State-Disks (SSDs)

Dayoung Lee, Joonho Lee and Minseok Song

***Abstract*—With the advent of new large-capacity solid-state disks (SSDs) such as quad-level-cells (QLC), SSD arrays can be effectively used in video storage systems that require large-capacity storage space. Typically, SSD manufacturers specify a drive-writes-per-day (DWPD) metric, which is the ratio of bytes written per day to the total capacity in bytes, to ensure an SSD's specified lifetime; it is important to limit the number of write operations by considering the DWPD for each SSD. We propose a new video file allocation technique to effectively manage the heterogeneous DWPD characteristics of SSDs in distributed storage systems. To express the degree of wear-leveling for heterogeneous SSDs, we first introduce the concept of ADWD, which is the actual number of bytes written per day compared to DWPD. We then propose two algorithms for file placement and migration. The file placement algorithm places files greedily based on the bandwidth-to-space ratio (BSR) of each file and SSD to balance the bandwidth usage and storage of the SSD. The file migration algorithm moves files from overloaded to underloaded SSDs to meet bandwidth limit requirements while minimizing the overall ADWD as a result of migration, and then migrates additional popular files to improve SSD bandwidth utilization. To use these algorithms in actual distributed file systems, we implemented a suite of tools for file placement and migration in the Hadoop distributed file system (HDFS). Experimental results show that the proposed algorithm reduces the mean of ADWD by 35.44% and its standard deviation by 69.78% compared to the benchmark methods on average.**

***Index Terms*—Storage Management, Distributed file systems, Streaming media, Multimedia systems**

I. Introduction

Recent advances in network and system technologies have made video services popular in a wide range of video applications, including user-generated content (UCC), video-on-demand (VoD), and over-the-top (OTT) streaming. These applications must handle a large number of concurrent clients, significantly increasing the amount of bandwidth. For example, nearly five billion videos are viewed on YouTube every day [1].

Manuscript received August 15, 2022; revised October 24, 2022; accepted November 6, 2022.

This paper was the result of the research project supported by SK hynix Inc. and this work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT). (NRF-2022R1A2C1007237) *(Corresponding author: Minseok Song.)*

Dayoung Lee and Minseok Song are with the Department of Computer Engineering, Inha University, Incheon, 22212, South Korea (e-mail: dayoung08@inha.edu, mssong@inha.ac.kr). Joonho Lee is with SK Hynix, Seongnam, 13558, South Korea (e-mail: joonho.lee@sk.com)

Significant storage space is needed when storing large numbers of video files. Additionally, each video file must be transcoded into multiple bitrate versions and stored, which also significantly increases the required storage space [2], [3], [4], [5], [6]. For example, on Netflix, each video can be transcoded 120 times [2]. Thus, it is important to effectively manage the limited storage space in video storage systems.

Solid-state disks (SSDs) have many technical advantages such as non-volatility, low power consumption, shock resistance, and excellent random read performance [7], [8], [9]. In addition, SSDs are desirable for use in video servers for the following reasons: low latency for read operations, good random read performance to support a large number of simultaneous video streams, and excellent sequential read/write performance [10], [11], [12], [13]. However, owing to the high cost of SSDs, video servers commonly use hard disk drives (HDDs) as their main storage devices [7], [8], [9].

With the advent of quad-level cell (QLC) SSDs that allow four bits per cell, the flash memory density has increased by 33% compared to triple-level cell (TLC) flash memory [14]. The high capacity per cell can provide a competitive cost per terabyte, making the use of SSDs in video storage systems viable. However, QLC SSDs are generally less reliable than TLC SSDs [14], [15], [16], [17]. For example, the program/erase (P/E) cycle of a QLC SSD is approximately one-tenth that of a TLC SSD [18], [19]. Therefore, to use QLC SSDs in video storage, it is essential to manage the lifespan of the SSDs.

The drive-writes-per-day (DWPD) metric is commonly used to express the endurance characteristics of SSDs. This is the ratio of bytes written per day to the total capacity of bytes to ensure a specified lifetime. For example, suppose the drive is 200 GB and the warranty is 5 years. A DWPD of one means that 200 GB of data can be written every day over five years [14], [18], [20]. To support the increasing demand for video files, SSDs can be gradually added, creating an array of SSDs with heterogeneous DWPDs, I/O bandwidths, and storage space characteristics. In particular, because QLC has a much lower DWPD value than TLC, different DWPD characteristics must be considered for data placement on an array of SSDs.

It is important to balance wear-out across SSDs constituting an SSD array for the following reasons [21], [22], [23]. If certain SSDs are heavily utilized, then the entire SSD array becomes unreliable, increasing maintenance costs in the data

center. For example, It may be necessary to take the entire array offline for maintenance, which can significantly degrade the server performance. Wear imbalance also causes frequent garbage collection, which degrades the I/O performance of the entire array and complicates the use of data redundancy techniques for fault tolerance [21], [22], [23].This non-uniform wear-out makes the SSD lifespan unpredictable, complicating both provisioning and load-balancing aspects of data center clusters [24].

Each video file has its own bandwidth and storage requirements. For example, a popular video clip requires more bandwidth than an unpopular video clip; however, a long video clip with a high bitrate requires more storage space [5], [6]. To effectively use the limited storage space and bandwidth of SSDs, it is important to strike a balance between bandwidth and storage when placing files. In addition, because the popularity of each video file changes continuously, data movement is essential [25], [26]; however, it is important to balance wear-out to guarantee the lifespan of each SSD [21], [22], [23]. To the best of our knowledge, none of the previous studies have tackled this issue.

We propose new data placement and migration methods to balance the wear-out of SSDs while effectively using the limited I/O bandwidth and storage space in distributed storage systems with heterogeneous SSDs. We first propose a file placement algorithm that aims to maximize both storage and bandwidth utilization using as few SSDs as possible. Based on this, we propose a file migration algorithm that can effectively cope with changes in popularity. For this purpose, it calculates the value of actual daily writes compared with DWPD (ADWD) for each SSD and then migrates the files to ensure the specified lifespan of each SSD while efficiently using the I/O bandwidth. To use the proposed file placement and migration schemes, a suite of tools was implemented in the Hadoop distributed file system (HDFS).

The remainder of this paper is organized as follows. Section II reviews related work, and Section III provides a system model. Sections IV and V present the file placement and migration algorithms, respectively. Section VI describes the algorithm implementation for distributed file systems. Section VII evaluates the proposed scheme and Section VIII concludes the paper.

II. Related Work

Several studies have proposed the use SSDs in video servers. For example, Zhang et al. [27] developed an online transcoding method to reduce storage overhead and an error-correction method to use low-cost flash memory in SSD-based video servers. Rahiman and Karim [28] presented a technique for reducing SSD access times by segmenting videos and storing adjacent video data pages on separate planes on different SSDs, allowing parallel access. He et al. [29] proposed a 360-degree video streaming method that determines the type of storage to be cached according to the popularity of video tiles in a storage system consisting of SSD, HDD, and DRAM to minimize storage access times.

Song [30] presented an SSD cache management method in a storage system with multispeed HDDs to minimize HDD energy consumption while limiting the number of speed transitions. Gao et al. [31] proposed a method to solve the I/O bandwidth limitation problem by migrating files from high-load to low-load nodes in an SSD-based hybrid storage system. Sun et al. [32] presented a method of using SSDs as write buffers and metadata storage in a video surveillance storage system. Al-wesabi et al. [33] proposed a VoD storage management system that can handle a large number of simultaneous streams. This system caches the next video to be played in a RAM buffer and stores popular files on a fast SSD and other files on an HDD. However, none of these studies considered SSD endurance.

Many caching and block management methods have been proposed to improve SSD longevity. For example, Salkhordeh [34] proposed a new SSD cache architecture that improves the lifespan of SSDs by not keeping dirty data pages in the cache and reconfiguring caching policies according to application classes. Liu et al. [35] proposed a method to store long-term popular data in an SSD cache while avoiding moving unpopular data from deduplication containers to the SSD to reduce the number of SSD write operations. Chai et al. [36] proposed a method to store frequently accessed data in the SSD cache for as long as possible to prevent the SSD cache from wearing out quickly. Ma et al. [37] proposed a method to extend the lifespan of an SSD by reusing bad blocks with relatively few uncorrectable bits. Yen et al. [38] proposed a bad block management method that manages adjacent blocks as clusters and determines the size of the clusters based on the similarity of the error tendency of adjacent blocks. Kim et al. [39] proposed a hybrid storage system composed of SSD, HDD, and DRAM by considering the trade-off between performance and longevity within the cost-budget. However, none of these studies considered the wear balance between the SSDs that make up the SSD array.

Several studies have been conducted to balance writes among SSDs. For example, Wang et al. [21] proposed a method for monitoring and balancing the distribution of writes on an SSD-based disk array, in which files are striped. Zhao et al. [22] proposed a write-offloading method that balances the wear and tear of multiple flash servers by periodically monitoring the popularity of all the replicated data and switching the redundancy policy for each file accordingly. Xiao et al. [23] showed that the existing hardware wear-leveling technique is ineffective in a system with multiple NVM slots, and proposed an allocation technique that can dynamically adjust the use of non-volatile memory (NVM) slots by calculating the degree of wear. However, none of these studies considered SSD endurance in video storage systems.

Several schemes have been developed to ensure the longevity of SSDs, many of which make use of throttling techniques [40], [41], [42], [43]. For example, Lee et al. [42] proposed a recovery-aware dynamic throttling scheme that exploits the self-recovery effect of floating-gate transistors in flash memory to ensure a specified SSD lifetime. They [43] improved this scheme by making throttling decisions that take into account the characteristics of the workload to ensure the required SSD lifespan with less performance degradation.

Several studies have been conducted considering SSD lifes-

pan in video storage systems. Kang et al. [44] proposed a video storage architecture that stores keyframes in a single-level cell (SLC) SSD and stores other frames in a multilevel cell (MLC) SSD. Ryu et al. [10] evaluated the effect of block replacement schemes on the lifespan of SSDs and argued that file-level least frequently used (LFU) schemes were more effective than interval caching in video storage systems. Lee et al. [45] developed a technique for maximizing the I/O bandwidth used by an SSD by selecting videos to be cached based on popularity and limiting the number of replacements to ensure SSD lifespans. However, none of these studies considered the heterogeneous endurance characteristics of SSDs.

Several authors have considered the use of distributed-file systems for video servers. Liu et al. [46] proposed a method to segment video files into data blocks by considering the relationships between frames in a distributed storage system. Al-Abbasi and Aggarwal [47] analyzed the latency of video streaming in an erasure-coding-based distributed storage system and presented a probabilistic scheduling access policy to optimize the quality of experience (QoE). SaatiAlsoruji [48] proposed a method to improve video data processing throughput in computing nodes by minimizing load imbalances among nodes in an HDFS used for video surveillance systems. Dai et al. [49] proposed an algorithm for a distributed file system that caches data to reduce storage costs and total access latency based on user requests and access patterns. However, none of these studies address SSD-related optimization issues such as SSD lifetime management, file placement, and migration.

To the best of our knowledge, this study is the first attempt to address the inter-SSD wear-leveling issue in a video storage system by considering the heterogeneous SSD endurance, I/O bandwidth, and storage space characteristics of SSDs constituting a distributed storage system.

III. System Model and Algorithm Concept

A. System Model

Fig. 1 shows the architecture of our video storage system, which consists of SSD and HDD zones. The SSD zone is used as the main storage area that handles actual video requests, whereas the HDD zone is used as the archival area to store unpopular video files. The goal of data placement and migration is to maximize the bandwidth utilization of the SSD area to fully benefit from the SSD's advantages, such as minimizing power consumption.

Each SSD j has a different bandwidth B_j^{limit} and storage capacity, S_j^{limit}, $(j = 1, ..., N^{\text{SSD}})$ where N^{SSD} is the number of SSDs. Table I summarizes the symbols used in this paper. We assume that each video file is divided into chunks of a fixed size, where S^{chunk} denotes chunk size. Data placement and migration are then managed in chunks.

Different SSDs may have different DWPD values, which are typically provided by SSD manufacturers. Let N_j^{DWPD} be the DWPD value for SSD j, $(j = 1, ..., N^{\text{SSD}})$. Then, the number of bytes that can be written to SSD j is $N_j^{\text{DWPD}} S_j^{\text{limit}}$ bytes per day to guarantee the given SSD lifetime. It is important to maintain the number of bytes written as less than those specified by these DWPD values. To express this, we define

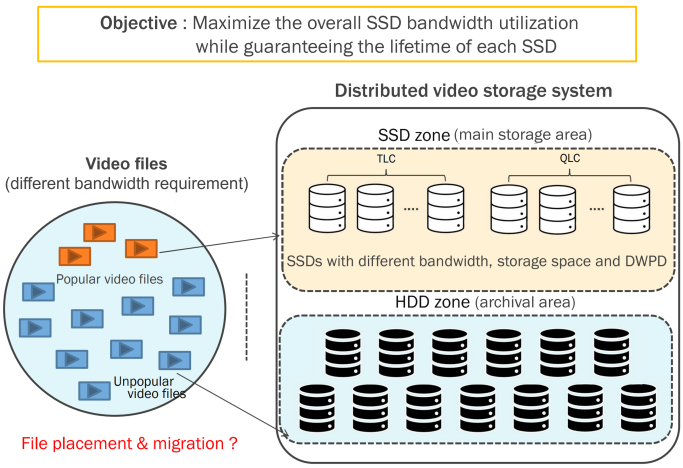

Fig. 1. Architecture of our video storage system.

TABLE I
Notations.

Symbol	Meaning
N^{SSD}	Number of SSDs
B_j^{limit}	Bandwidth limit for SSD j
S_j^{limit}	Storage capacity for SSD j
N_j^{DWPD}	DWPD value for SSD j
p_j	Period in which SSD j is actually used compared to its lifetime
N_j^{life}	Lifetime of the SSD j in days
W_j^{DWPD}	Maximum number of bytes that can be written to SSD j for $p_j N_j^{\text{life}}$ days to satisfy the lifetime constraints
W_j^{actual}	Actual number of total bytes written to SSD j for $p_j N_j^{\text{life}}$ days
$ADWD_j$	Ratio of the actual number of bytes written per day compared to that suggested by the DWPD value
N^{chunk}	Number of video chunks
B_i	Bandwidth requirement in Mbps for chunk i
X_i	SSD index where chunk i is stored
S^{chunk}	Size of chunks in which the video is divided into fixed size
$R_{i,j}^{\text{BSR}}$	Bandwidth-to-space ratio for each SSD j to allocate chunk i
A^{yet}	Array of chunk indices that have not yet been placed
X_i^{prev}	SSD index where chunk i is stored before algorithm execution
X_i^{tmp}	Variable that records the value of X_i during algorithm execution
A^{over}	Array of overloaded SSD indices
A^{under}	Array of underloaded SSD indices
F_j^{ST}	Binary variable indicating whether SSD j has free storage space to store the migrated chunk
H_j	Chunk index requiring the highest bandwidth on SSD j
L_j	Chunk index requiring the lowest bandwidth on SSD j
$R_{i,j}^{\text{FMA}}$	Ratio of SSD bandwidth to ADWD for each SSD j
I^{src}	SSD index j ($j \in A^{\text{over}}$) with the highest value of $\sum_{\forall i\ X_i^{\text{tmp}}=j} B_i - B_j^{\text{limit}}$
A^{des}	Array of SSD indices with available bandwidth for chunk $H_{I^{\text{src}}}$
I^{des}	SSD index j ($j \in A^{\text{des}}$) with the highest value of $R_{H_{I^{\text{src}}},j}^{\text{FMA}}$
F_j^{BW}	Binary variable indicating whether bandwidth of SSD j is fully used
A^{avail}	Array of SSD indices with available bandwidth for chunk $H_{I^{\text{des}}}$
I^{avail}	SSD index j ($j \in A^{\text{avail}}$) with highest value of $R_{H_{I^{\text{des}}},j}^{\text{FMA}}$

a new metric called ADWD, which represents the ratio of the actual number of bytes written per day to that suggested by the DWPD value. This indicates the amount of data written relative to the specified DWPD value so far, and it is important to reduce the ADWD value for all SSDs to extend the lifetime of SSDs.

Let p_j, $(0 < p_j \leq 1)$ be the ratio of the period in which SSD j is actually used to its lifetime. Let N_j^{life} be the lifetime of the SSD j in days. We introduce a new variable W_j^{DWPD}

to represent the maximum number of bytes that can be written to SSD j for $p_j N_j^{\text{life}}$ days to satisfy the constraints specified by the DWPD metric. W_j^{DWPD} can be calculated as follows:

$$W_j^{\text{DWPD}} = N_j^{\text{DWPD}} S_j^{\text{limit}} p_j N_j^{\text{life}}. \quad (1)$$

Let W_j^{actual} be the actual number of total bytes written to SSD j for $p_j N_j^{\text{life}}$ days. Then, the ADWD value for SSD j, $ADWD_j$ can be expressed as follows:

$$ADWD_j = \frac{W_j^{\text{actual}}}{W_j^{\text{DWPD}}}. \quad (2)$$

If $ADWD_j > 1$, then the number of bytes written to SSD j is greater than specified by DWPD, which negatively affects lifetime guarantees. The lower the $ADWD_j$ value, the lower the number of writes compared with the DWPD, thus extending the life of the SSD. It is thus essential to keep the value of $ADWD_j$ as low as possible to ensure that the SSD lasts as long as possible.

B. Algorithm Concept

The objective of our scheme is to maximize throughput, the amount of bandwidth that SSDs can provide, which requires balanced use of SSD bandwidth and storage. For example, if unpopular chunks are concentrated on some SSDs, their SSD bandwidth cannot be fully utilized, thereby reducing the number of requests to be processed by the SSD zone. To address this issue, we propose a file placement algorithm (FPA) that determines the files to be stored on each SSD.

FPA places the files requiring the highest bandwidth first on the SSD with the highest bandwidth-to-space ratio (BSR) value [50], [51]. To efficiently utilize the storage space, popular video files are first stored on an SSD, and then the SSD with the highest BSR value is selected as the location for data placement to balance storage space and bandwidth usage.

In practice, video popularity continuously changes [25], [26], [52]. Obviously, to maximize throughput, the FPA can be re-executed when a popularity change is detected, but this may involve many file migrations, which increases the number of write operations and negatively affects SSD longevity. To address this, we propose a file migration algorithm (FMA) that determines the SSD index for each file after migration, with the aim of minimizing the overall ADWD.

To achieve the two goals of minimizing the overall ADWD while maximizing throughput, the FMA is divided into two phases: elimination of overloaded SSDs (EOS) and BSR ratio enhancement (BE). It first calculates the expected bandwidth utilization of each SSD from the video popularity prediction and determines SSDs with utilization greater than 1 as overloaded SSDs. Then, the EOS phase migrates files from overloaded to underloaded SSDs, taking into account each SSD's ADWD value. For this purpose, when a chunk from the overloaded SSD is migrated, it calculates a parameter for each SSD to express the ratio of the decrease in bandwidth of an overloaded SSD to the increase in ADWD of an underloaded SSD. Then the SSD with the highest parameter value can be selected as the target SSD for migration. The BE phase is then executed to maximize the throughput of each SSD by moving popular files from the HDD zone to appropriate underloaded SSDs.

FPA can be used to determine the initial chunk placement on the SSD arrays. For example, this process is required when the SSD array is replaced or all files stored on the SSD zone are moved to the HDD zone. On the other hand, FMA can be used in most cases where the popularity of video files changes and new video files are added gradually. Because FPA only aims to maximize bandwidth utilization without considering ADWD, it can provide an upper bound on the bandwidth utilization that FMA can achieve. Therefore, FMA can be used as a practical means of allowing for changes in popularity; FPA can still be used as a guideline to provide an upper bound on SSD bandwidth utilization.

IV. File Placement Algorithm

A. Problem Formulation

We present a file placement optimization problem. Assume that chunk i, $(i = 1, ..., N^{\text{chunk}})$ is stored and served by SSD X_i, $(X_i = 0, 1, ..., N^{\text{SSD}})$. For ease of exposition, SSD 0 represents the HDD zone; therefore, if $X_i = 0$, then chunk i is stored in the HDD zone. The two constraints (**C1** and **C2**) are defined as follows:

1) **C1:** (Bandwidth limit constraint) Let B_i be the bandwidth requirement in Mbps for chunk i. The bandwidth usage of SSD j must not exceed B_j^{limit}:

$$\sum_{\forall i \ X_i = j} B_i \leq B_j^{\text{limit}}, (j = 1, ..., N^{\text{SSD}}). \quad (3)$$

2) **C2:** (Storage limit constraint) Total storage space used on SSD j must not exceed its storage capacity, S_j^{limit} as follows:

$$\sum_{\forall i \ X_i = j} S^{\text{chunk}} \leq S_j^{\text{limit}}. \quad (4)$$

We formulate the throughput maximization problem (TMP), which determines the value of X_i for each chunk i as follows:

$$\begin{aligned} &\text{Maximize} && \textstyle\sum_{j=1}^{N^{\text{SSD}}} \sum_{\forall i \ X_i = j} B_i \\ &\text{Subject to} && \textstyle\sum_{\forall i \ X_i = j} B_i \leq B_j^{\text{limit}}, (j = 1, ..., N^{\text{SSD}}) \\ & && \textstyle\sum_{\forall i \ X_i = j} S^{\text{chunk}} \leq S_j^{\text{limit}}, (j = 1, ..., N^{\text{SSD}}). \end{aligned}$$

B. File Placement Algorithm

TMP is NP-hard because it can be reduced to a multi-dimensional multiple knapsack problem (MDMKP), which is known to be NP-hard [53]. For example, MDMKP has several knapsacks with multidimensional constraints on resource usage. In addition, there are objects that require multiple resources, and there are profits that these objects return when allocated to knapsacks. The objective of MDMKP is to assign objects to knapsacks to maximize total profit while satisfying knapsack resource limits. Likewise, in TMP, each SSD (knapsack) has two constraints: storage and bandwidth

(multidimensional constraints). There are chunks (objects), each of which has capacity and bandwidth requirements and throughput values (profit). The objective of TMP is to assign chunks to one of the SSDs to maximize the overall throughput.

When there are multidimensional constraints on resource usage, it is important to balance resource usage (bandwidth and storage resources) [53]. To use storage space efficiently, FPA places popular chunks first. In addition, for a balanced use of storage and bandwidth, it calculates the remaining bandwidth and storage of each SSD as a result of file allocation during algorithm execution, and stores that chunk on the SSD with the highest BSR value.

If there are several SSDs with the same BSR value, the SSD with the highest remaining bandwidth is selected. For this purpose, before a new chunk i is allocated, it calculates a parameter $R_{i,j}^{\mathrm{BSR}}$ to express the remaining BSR for each SSD j as follows:

$$R_{i,j}^{\mathrm{BSR}} = \frac{B_j^{\mathrm{limit}} - (\sum_{\forall i \ X_i=j} B_i + B_i)}{S_j^{\mathrm{limit}} - (\sum_{\forall i \ X_i=j} S^{\mathrm{chunk}} + S^{\mathrm{chunk}})}. \quad (5)$$

Algorithm 1 presents the details of this algorithm.

Algorithm 1: File placement algorithm (FPA).

Input: B_i $(i = 1, ..., N^{\mathrm{chunk}})$, B_j^{limit} and S_j^{limit} $(j = 1, ..., N^{\mathrm{SSD}})$;
Output: X_i $(i = 1, ..., N^{\mathrm{chunk}})$;
$\forall\ i$, X_i is initialized as 0 ;
$A^{\mathrm{yet}} \leftarrow \{1, ..., N^{\mathrm{chunk}}\}$;
while $A^{\mathrm{yet}} \neq \phi$ **do**
  Find the chunk index i with the highest value of B_i in A^{yet} for which $i = v$;
  Find the SSD index j with the highest value of $R_{v,j}^{\mathrm{BSR}}$ where $\sum_{\forall i \ X_i=j} S^{\mathrm{chunk}} + S^{\mathrm{chunk}} \leq S_j^{\mathrm{SSD}}$ for which $j = h$;
  if $\sum_{\forall i \ X_i=h} B_i + B_v \leq B_h^{\mathrm{limit}}$ **then**
    $X_v = h$;
  end
  $A^{\mathrm{yet}} \leftarrow A^{\mathrm{yet}} - \{v\}$;
end

By default, the model in Fig. 1 assumes that caching is not used, but FPA can be easily extended to support SSD caching without modification. For example, FPA aims for determining the SSD index of X_i for each chunk i, $(i = 0, ..., N^{\mathrm{SSD}})$, where SSD 0 indicates the HDD zone. Thus, if $X_i = 1, ..., N^{\mathrm{SSD}}$, then chunk i is considered cached on SSD; otherwise, it is not cached.

V. File Migration Algorithm for Minimizing Overall ADWD

As video popularity changes, the SSD location where the files are stored may change to effectively use the limited SSD bandwidth. A straightforward method of addressing this file migration issue is to run the FPA when the popularity changes and move the files as specified by the algorithm. However, frequent write operations can wear SSDs and shorten their lifespans. In addition, the file migration policy must consider the different DWPD characteristics of each SSD to maximize SSD lifespan.

Let X_i^{prev} be the SSD index, where the chunk i is stored in the current placement. Then, the objective of the FMA is to find a new SSD index X_i for chunk i to satisfy two constraints, **C1** and **C2**, with the aim of minimizing the overall ADWD while maximizing the throughput. To achieve this, the FMA migrates chunks stored on overloaded SSDs to one of the underloaded SSDs to balance bandwidth usage among SSDs. It also considers storing the chunks of the newly uploaded video files. Video server applications typically contain information about each video, including its release date. This information can be used to identify newly uploaded files.

For ease of exposition, new chunks are assumed to be stored in the HDD zone (SSD 0) prior to algorithm execution; thus, $X_i^{\mathrm{prev}} = 0$ for a new chunk i. FMA is then run to determine the placement of all chunks based on video popularity. Next, chunks are placed according to the execution result of FMA. For this purpose, FMA maintains a new variable X_i^{tmp} for each chunk i to record the value of X_i during the execution of the algorithm. $\forall i$, $X_i^{\mathrm{tmp}} = X_i^{\mathrm{prev}}$.

FMA maintains two arrays, A^{over} and A^{under}, to store SSD indices of the overloaded and underloaded SSDs, respectively. Thus, if $\sum_{\forall i \ X_i^{\mathrm{prev}}=j} B_i > B_j^{\mathrm{limit}}$, $j \in A^{\mathrm{over}}$; otherwise, $j \in A^{\mathrm{under}}$. FMA is divided into EOS and BE phases. The EOS phase migrates the chunks that require high bandwidth from overloaded to underloaded SSDs until the following condition is satisfied: $A^{\mathrm{over}} = \phi$. However, if these SSDs cannot be found because of insufficiencies in the remaining bandwidth of the underloaded SSDs, then the chunks requiring low bandwidth on the overload SSDs are temporarily migrated to the HDD zone, which can make A^{over} empty. The BE phase then examines the chunks located in the HDD zone (SSD 0) to migrate them to the underutilized SSDs if this helps improve the BSR on the SSDs. Algorithm 2 shows the details of the FMA, which consists of EOS and BE phases.

If the underloaded SSD has unused storage space, then the migrated chunk i can be placed there, reducing B_i MB/s on the overloaded SSD. Otherwise, a chunk must be removed from the underloaded SSD to free up storage space for the migrated chunk. To express this, we introduce a flag parameter F_j^{ST} to indicate whether SSD j has the remaining storage space to store the migrated chunk. Thus, if $F_j^{\mathrm{ST}} = 0$, then SSD j has the remaining storage space for a chunk; otherwise, it does not. If $F_j^{\mathrm{ST}} = 0$, then a chunk of the overloaded SSD can be migrated to the underloaded SSD. Otherwise, the chunk must be evicted from the underloaded SSD. In this case, to satisfy the constraints while improving the BSR of each SSD, a chunk requiring the lowest bandwidth from the underloaded SSD must be evicted. This evicted chunk then temporarily migrates to the HDD zone.

To satisfy the two objectives of minimizing the overall ADWD while maximizing throughput, FMA introduces a new ratio parameter, $R_{i,j}^{\mathrm{FMA}}$, when chunk i is migrated to SSD j as follows:

$$R_{i,j}^{\mathrm{FMA}} = \frac{B_i - F_j^{\mathrm{ST}} B_{L_j}}{ADWD_j + \frac{S^{\mathrm{chunk}}}{W_j^{\mathrm{DWPD}}}}. \quad (6)$$

The denominator of $R_{i,j}^{\mathrm{FMA}}$ represents the ADWD value of SSD j after migration, whereas its numerator represents the reduced total bandwidth of SSD $j \in A^{\mathrm{over}}$.

Let H_j and L_j be the chunk indices that require the highest and lowest bandwidths on SSD j, respectively. To maintain the ADWD value as low as possible while minimizing the bandwidth used by SSD $j \in A^{\mathrm{over}}$, it is advantageous to migrate chunk i to SSD j with the highest value of $R_{i,j}^{\mathrm{FMA}}$. However, this must not violate condition **C2**. Based on this, the EOS phase has the following four steps, as described between lines 4 and 18 of Algorithm 2.

1) The EOS phase first finds source SSD index $j \in A^{\mathrm{over}}$ with the highest value of $\sum_{\forall i\ X_i^{\mathrm{tmp}}=j} B_i - B_j^{\mathrm{limit}}$ where $j = I^{\mathrm{src}}$. Then migrating chunk $H_{I^{\mathrm{src}}}$ can reduce the highest bandwidth on the most heavily loaded SSD I^{src}.
2) To find the target SSD index for migration, it then finds the highest value of $R_{H_{I^{\mathrm{src}}},j}^{\mathrm{FMA}}$, for which $j = I^{\mathrm{des}}$. Then, chunk $H_{I^{\mathrm{src}}}$ is moved to SSD I^{des} so that $X_{H_{I^{\mathrm{src}}}}^{\mathrm{tmp}} = I^{\mathrm{des}}$. If $F_{I^{\mathrm{des}}}^{\mathrm{ST}} = 1$ so that there is no remaining space on SSD I^{des}, then chunk $L_{I^{\mathrm{des}}}$ is temporarily moved to the HDD zone so that $X_{L_{I^{\mathrm{des}}}}^{\mathrm{tmp}} = 0$.
3) If there is no SSD $j \in A^{\mathrm{under}}$ that can accommodate the migrated chunk $H_{I^{\mathrm{src}}}$ from SSD I^{src} due to bandwidth limitation, then the chunks requiring low bandwidth on SSD I^{src} are migrated to the HDD zone one by one, until SSD I^{src} becomes underloaded.
4) Two arrays, A^{over} and A^{under}, are updated to reflect workload changes. Then, the steps above are repeated until A^{over} becomes empty.

The BE phase migrates chunks on the HDD to the SSD zones to improve the overall BSR. For this purpose, we maintain a flag variable F_j^{BW} for each SSD j, $(j = 1, ..., N^{\mathrm{SSD}})$ to express the bandwidth usage of SSD j, where all values of F_j^{BW} are initialized to 0. If all of these values are 1, then the bandwidth of all SSDs is fully used, so the BE phase ends. Based on this, the BE phase (lines 20–37 in Algorithm 2) can be described as follows:

1) To find the SSD index to which chunk H_0 will be migrated, the BE phase first examines the SSD indices j, where $F_j^{\mathrm{BW}} = 0$ and $F_j^{\mathrm{ST}} = 0$, but if $\forall F_j^{\mathrm{ST}} = 1$, then it searches for SSD indices that only satisfy the condition $F_j^{\mathrm{BW}} = 0$ to find the highest value of $R_{H_0,j}^{\mathrm{BSR}}$ for which $j = I^{\mathrm{des}}$.
2) If chunk migration from the HDD zone to SSD I^{des} would exceed $B_{I^{\mathrm{des}}}^{\mathrm{limit}}$, then it finds SSDs with available bandwidth to accommodate chunk $H_{I^{\mathrm{des}}}$. If those SSDs can be found, then chunk $H_{I^{\mathrm{des}}}$ is migrated to an SSD with the highest value of $R_{H_{I^{\mathrm{des}}},j}^{\mathrm{FMA}}$. Otherwise, the chunks requiring the lowest bandwidth from SSD I^{des} are moved to the HDD zone one by one until the **C2** condition for SSD I^{des} is satisfied, and then $F_{I^{\mathrm{des}}}^{\mathrm{BW}}$ is set to 1.
3) The steps above are repeated until all values of F_j^{BW}, $(j = 1, ..., N^{\mathrm{SSD}})$ are 1.

FMA can also be used effectively for SSD-caching-based architectures. For example, because all video chunks are already stored in the HDD zone in a caching-based architecture, there is no need to migrate chunks from SSD to the HDD zone. Therefore, all codes related to this process (lines 10-12, 15, 28-30, and 33 in Algorithm 2) should be removed. Except for that, FMA can be used as is.

VI. Implementation on Hadoop Distributed File System

HDFS is one of the most widely used file systems owing to its excellent scalability [46], [47], [54]. It provides an effective and reliable means of managing a pool of big data and analytics applications. The growing demand for video processing in various application areas is driving the use of HDFS in video storage servers [46], [47], [54], [55]. In particular, for a video storage system in which video files are continuously uploaded, it is essential to build a scalable cluster-based server that can easily add storage devices such as HDFS. Therefore, we propose a method that uses our algorithms for HDFS. Our implementations are based on the source code of Hadoop version 3.3.1 provided by the official site.

HDFS supports a master-slave architecture consisting of a single node and multiple DataNodes [56]. The NameNode plays the role of a master server to manage the file system namespace, performing tasks such as opening, closing, and renaming files and directories and controlling file access by clients. The NameNode also divides each file into chunks and randomly allocates each chunk to a DataNode. The DataNode then stores each chunk in one of the physical storage devices managed by that DataNode.

Two software modules in Hadoop, (*block placement policy* and *volume choosing policy* classes), determine which device each chunk is stored on, while two tools (*Balancer* and *Mover*) can be used in HDFS to allow file migrations across DataNodes. However, these built-in software modules in Hadoop cannot be used for both FPA and FMA for several reasons.

- The *block placement policy* class in the NameNode randomly determines the DataNode on which each chunk can be placed [48].
- The *volume choosing policy* class in the DataNode stores each chunk across the devices in that DataNode in a round-robin manner [57].
- The *Balancer* and *Disk balancer* tools move files from a device with less free space to a device with more free space [58], [59].
- The *Mover* tool classifies storage types by ARCHIVE, DISK, SSD, and RAM_DISK and only supports migrating files between them [60].

To allow file placement and migration as specified by the algorithm, we first modified the *block placement policy* and *volume choosing policy* classes in the HDFS and created two new tools, *FilePlacement* and *FileMovement*. These tools are then used to place chunks on HDFS as specified by the

Algorithm 2: File migration algorithm (FMA).

Input: B_i $(i = 1, ..., N^{\text{chunk}})$, S^{chunk}, B_j^{limit}, S_j^{limit}, W_j^{DWPD}, and W_j^{actual} $(j = 1, ..., N^{\text{SSD}})$;
Output: X_i $(i = 1, ..., N^{\text{chunk}})$;

$\forall\, i$, X_i^{tmp} is initialized as X_i^{prev} ; $\forall\, j$, F_j^{BW} is initialized to 0 ;
/* *EOS phase* */;
while $A^{\text{over}} \neq \phi$ **do**
  Find the SSD index $j \in A^{\text{over}}$ with the highest value of $\sum_{\forall i\ X_i^{\text{tmp}}=j} B_i - B_j^{\text{limit}}$ for which $j = I^{\text{src}}$;
  Find an array of indices A^{des} of SSDs with an available bandwidth for chunk $H_{I^{\text{src}}}$;
  if $A^{\text{des}} \neq \phi$ **then**
    Find the highest value of $R^{\text{FMA}}_{H_{I^{\text{src}}},j}$, $(j \in A^{\text{des}})$ for which $j = I^{\text{des}}$;
    $X^{\text{tmp}}_{H_{I^{\text{src}}}} \leftarrow I^{\text{des}}$;
    if $F^{\text{ST}}_{I^{\text{des}}} = 1$ **then**
      $X^{\text{tmp}}_{L_{I^{\text{des}}}} \leftarrow 0$;
    end
  end
  else
    Migrate chunk $L_{I^{\text{src}}}$ to the HDD zone and update $X^{\text{tmp}}_{L_{I^{\text{src}}}}$ to 0 one by one until **C2** for SSD I^{des} is satisfied;
  end
  Update A^{over} and A^{under};
end
/* *BE phase* */;
while $\exists j, F_j^{\text{BW}} = 0$ *and* $\exists i, X_i^{\text{tmp}} = 0$ **do**
  Find the highest value of $R^{\text{BSR}}_{H_0,j}$ $(F_j^{\text{BW}} = 0)$ for which $j = I^{\text{des}}$;
  $X^{\text{tmp}}_{H_0} \leftarrow I^{\text{des}}$;
  if $\sum_{\forall i\ X_i^{\text{tmp}}=I^{\text{des}}} B_i + B_{H_0} - F^{\text{ST}}_{I^{\text{des}}} B_{L_{I^{\text{des}}}} > B^{\text{limit}}_{I^{\text{des}}}$ **then**
    Find an array of indices of SSDs (A^{avail}) with available bandwidth for chunk $H_{I^{\text{des}}}$;
    if $A^{\text{avail}} \neq \phi$ **then**
      Find the highest value of $R^{\text{FMA}}_{H_{I^{\text{des}}},j}$, $(j \in A^{\text{avail}})$ for which $j = I^{\text{avail}}$;
      $X^{\text{tmp}}_{H_{I^{\text{des}}}} \leftarrow I^{\text{avail}}$;
      if $F^{\text{ST}}_{I^{\text{avail}}} = 1$ **then**
        $X^{\text{tmp}}_{L_{I^{\text{avail}}}} \leftarrow 0$;
      end
    end
    else
      Migrate chunk $L_{I^{\text{des}}}$ to the HDD zone and update $X^{\text{tmp}}_{L_{I^{\text{des}}}}$ to 0 one by one until **C2** for SSD I^{des} is satisfied;
      $F^{\text{BW}}_{I^{\text{des}}} = 1$;
    end
  end
end
/* *Finalization* */;
$\forall\, i$, X_i finalized to X_i^{tmp} ;

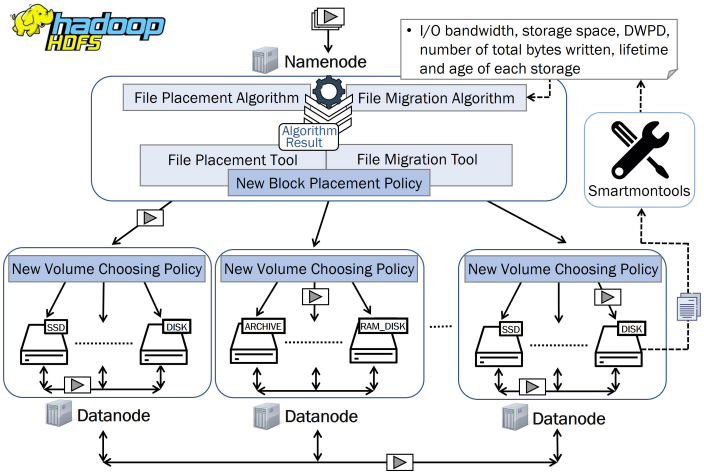

Fig. 2. Process of data placement and migration on the HDFS.

algorithms, as shown in Fig. 2. For this purpose, the algorithm results are written to two files, *placementInfo.in* file for FPA and *movementInfo.in* file for FMA; the *'hdfs FilePlacement'* and *'hdfs FileMovement'* commands are then executed to place and migrate chunks as specified by the FPA and FMA, respectively.

Two files, *placementInfo.in* and *movementInfo.in*, are located in a directory *'/shared'* in the NameNode through which all DataNodes can obtain chunk placement information. The *placementInfo.in* file contains the hostname of the DataNode and storage index within the DataNode for each chunk, and the *movementInfo.in* file contains a pair of source and destination DataNodes and storage indices for the chunks being migrated.

We modified several execution-related modules, including *hdfs, hdfs.cmd, hadoop* and *hadoop.cmd* in the *HADOOP_HOME/bin* directory, to provide a command-line interface to run new tools (*'hdfs FilePlacement'* and *'hdfs FileMovement'*). After executing these commands, the *block placement policy* class directs the chunks to the appropriate DataNodes, and the *volume choosing policy* class stores the chunks on the device determined by the algorithm.

Smartmontools, a set of utility programs built into most modern disks, provides SSD age, total writes, SSD brand name, etc [61]. We integrated this tool into HDFS to derive ADWD values for each SSD.

VII. Experimental Results

Simulations were conducted on an SSD array with commercial SSDs with the specifications shown in Table II, where terabytes written (TBW) is the total number of terabytes that can be written, and DWPD is calculated assuming that the SSD has a lifespan of 5 years. Each SSD has a storage capacity of 0.5TB, 1TB, 2TB, or 4TB at random. The chunk size, S^{chunk}, is set to 15MB. The bandwidth requirements for each file are calculated based on the bit-rate of 2.5MB/s video data, which is HD video quality.

The default parameters are as follows:

1) N^{SSD} is set to 30, so three of each SSD in Table II are used to build an SSD array.

TABLE II
SSD SPECIFICATION USED FOR SIMULATIONS.

Name	Bandwidth	TBW	Memory type
SK Hynix gold P31	3500MB/s	750TB	TLC
Samsung 980	3100MB/s	600TB	TLC
Seagate Barracuda Q5	2400MB/s	274TB	QLC
Micron Crucial P1	1800MB/s	200TB	QLC
Transcend SSD370S	560MB/s	1180TB	MLC
SK Hynix Gold S31	560MB/s	600TB	TLC
Transcend SSD230S	560MB/s	560TB	TLC
Samsung 870 QVO	560MB/s	360TB	QLC
Transcend SSD220Q	550MB/s	200TB	QLC
Micron Crucial BX500	540MB/s	195TB	QLC

2) The total bandwidth requirement of all video chunks ($\sum_{i=1}^{N^{\text{chunk}}} B_i$) is set to 50 GB/s.
3) N^{chunk} is set to 3 million.
4) Since most VoD systems migrate video files in consideration of popularity during off-peak hours, such as early in the morning [52], it is assumed that the FMA algorithm runs daily once a day during off-peak hours.
5) The access probability for each video follows a Zipf distribution, where θ was set to 0.271 observed from the real VoD system [62].
6) Users can end video playback before completion, so even in a single video, earlier chunks are more popular than later ones [63], [64]. This inter-chunk popularity can also be modeled as a Zipf distribution where θ was set to 0.2 [63].
7) Video life-cycle consists of three phases; a hot phase for newly uploaded files, a warm phase, and a cold phase where videos are rarely viewed [52]. To reflect this, newly uploaded videos in the last 7 days were included in the hot phase, assigned the highest popularity, and the popularity among the videos in the hot phase was randomly selected. Likewise, it is assumed that videos uploaded during the previous 3 weeks belong to the warm phase, and the rest belong to the cold phase.

A. *Efficacy of FPA*

We compare FPA with five other benchmarks bandwidth-aware placement (BWAP) [45], storage-aware placement (SWAP) [65], lifetime-aware placement (LWAP) [21], [22], random placement (RDP) [65], and round-robin placement (RRP) [57]), each of which places chunks on SSDs with different allocation priorities until both **C1** and **C2** conditions are met. For example, all three methods (BWAP, SWAP, and LWAP) are allocated individually, starting with the chunk with the highest bandwidth, but this chunk is placed first on the SSD with the highest remaining bandwidth in BWAP, the largest remaining storage in SWAP, and the lowest ADWD in LWAP. The RDP selects a chunk and allocates it to an SSD at random, whereas the RRP selects a chunk and allocates it to each SSD in turn.

1) Effect of Number of SSDs: We examine the effect of the number of SSDs on SSD bandwidth usage, calculated as $\sum_{j=1}^{N^{\text{SSD}}} \sum_{\forall i\ X_i=j} B_i$. Fig. 3 shows how the SSD bandwidth usage varies with the number of SSDs, which demonstrates that FPA always provides the best performance. For example, it improves SSD bandwidth usage by up to 8.35x and by an average of 3.18x compared with other methods.

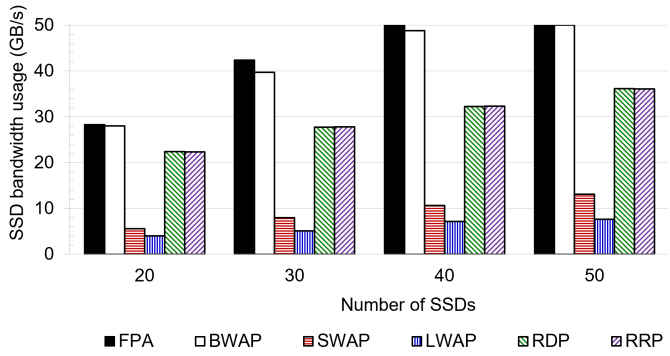

Fig. 3. SSD bandwidth usage (GB/s) against the number of SSDs.

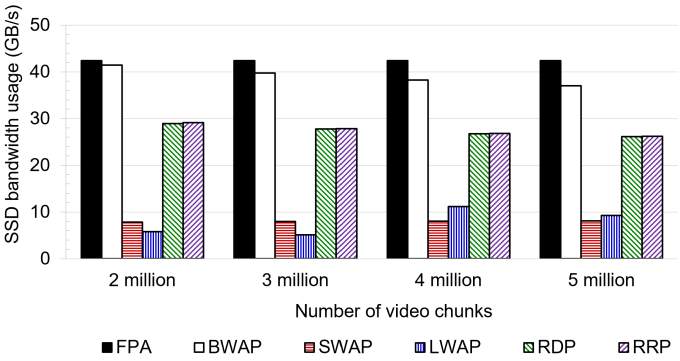

Fig. 4. SSD bandwidth usage (GB/s) against the number of video chunks.

Among other schemes, BWAP performs the best, indicating the need to allocate chunks based on bandwidth rather than the remaining capacity. However, when $N^{\text{SSD}} = 30$, its bandwidth usage is 6.74% lower than FPA owing to the unbalanced use of storage space and bandwidth, FPA shows between 26.04% and 54.92% higher SSD bandwidth usage than RDP and RRP. This implies that HDFS default allocations (i.e. random allocation by NameNode and round-robin placement by DataNode [48], [57]) are inefficient for video allocation.

2) Effect of Number of Video Chunks: Fig. 4 shows SSD bandwidth usage for various numbers of video chunks. Again, FPA shows the best performance, providing an average of 3.1x higher SSD bandwidth usage than other schemes. Among other schemes, BWAP performs the best, although its SSD bandwidth usage is 2.31% to 14.43% lower than FPA. SWAP and LWAP show the worst performance because they do not consider bandwidth requirements. The difference in bandwidth utilization between FPA and BWAP increases with the number of chunks, indicating that it is important to balance bandwidth with storage as the number of chunks increases.

3) Effect of Aggregate Bandwidth Demand of Video Chunks: Fig. 5 illustrates how SSD bandwidth usage depends on the aggregate bandwidth demand of all video chunks ($\sum_{i=1}^{N^{\text{chunk}}} B_i$). It shows results similar to those in the previous experiments, indicating an average of 2.92x higher SSD bandwidth usage compared with the other schemes. Again, BWAP performs the best among the other schemes. However,

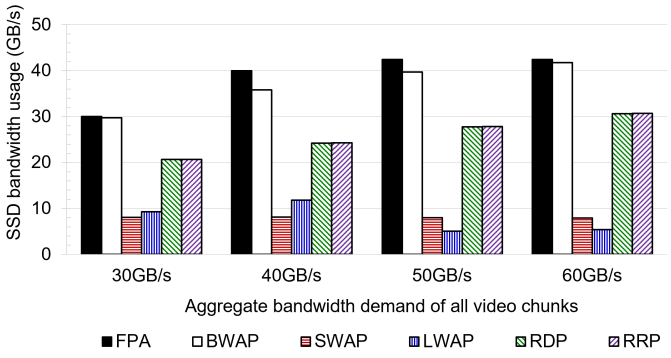

Fig. 5. SSD bandwidth usage (GB/s) against aggregate bandwidth demand of all video chunks.

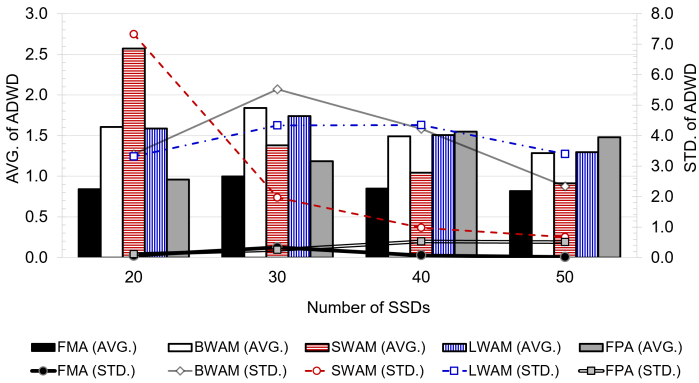

Fig. 6. Average and standard deviation values of ADWD against the number of SSDs.

when the total bandwidth demand exceeded 40 GB/s, its SSD bandwidth usage is 11.71% lower than FPA. In FPA, when the total bandwidth demand exceeded 42.4 GB/s, the SSD bandwidth usage no longer increases because of the total limit of SSD bandwidth.

B. Efficacy of FMA

To account for dynamic changes in video popularity, consider a video storage system where the hot popular video chunks of 1% of the number of existing videos are added periodically. We assume that this popularity change of video chunks occurs once per day. For a fair comparison, simulations were performed under identical initial conditions. Video chunks are initially stored as a result of the FPA algorithm, assuming that all SSDs are 30 days old, with an ADWD value of 1.

We compare FMA with four other schemes including FPA. The three schemes bandwidth-aware migration (BWAM) [31], storage-aware migration (SWAM) [58], [59], and lifetime-aware migration (LWAM) [22]) all migrate chunks with the highest bandwidth from SSD j, ($j = 0$ or $j \in A^{\text{over}}$) one by one to SSD k, ($k \in A^{\text{under}}$) until both **C1** and **C2** conditions are satisfied. These chunks are then placed on the SSD with the highest remaining bandwidth in BWAM, the largest remaining storage in SWAM, and the lowest ADWD in LWAM. In SWAM, if there is no SSD with remaining storage capacity, then the overloaded SSD can be unburdened by file swapping with the SSD with the remaining bandwidth. When migration is required, FPA can be run again to find an upper bound on SSD bandwidth usage. We then examine the effects of SSD, video chunk numbers, and total bandwidth demand using two performance metrics: SSD bandwidth usage and ADWD. All results are obtained one week after each scheme is executed.

1) Effect of Number of SSDs: Fig. 6 shows how the average and standard deviation of ADWD values vary with the number of SSDs, and Table III tabulates the normalized SSD bandwidth usage compared with that of FPA. From there, we make the following observations.

- FMA yields the same SSD bandwidth usage as FPA as shown in Table III, because the BE phase is further executed to fully utilize the remaining SSD bandwidth.
- FMA produces the lowest average ADWD, ranging from 10.12% to 67.19% (mean 36.42%) lower than the other schemes. This is because the remaining bandwidth, storage, and ADWD are all taken into account when determining which SSD the migrated chunk will be placed on as expressed in Equation (6).
- The standard deviation of ADWD in FMA is up to 99.37% (mean 83.54%) lower than the other schemes, indicating that FMA can effectively balance wear-leveling across the SSDs.
- Among other schemes, BWAM has the worst performance when $N^{\text{SSD}} = 30$. If the number of SSDs is small, then BWAM mainly needs file swapping owing to storage limitations, so each file migration may require write operations on two SSDs. However, SWAM can avoid this swapping by migrating chunks to SSDs with ample remaining space.
- Among other schemes, BWAM yields the lowest ADWD when $N^{\text{SSD}} = 50$, suggesting that bandwidth can be a major factor in determining ADWD when there are many SSDs with free space in an array.
- For FPA, ADWD increases with the number of SSDs because the number of migrations increases with the number of SSDs.
- Among other schemes, SWAM performs best when $N^{\text{SSD}} = 50$. This is because file swapping due to insufficient storage space occurs less compared to other schemes.
- LWAM yields the lowest SSD bandwidth usage as shown in Table III because SSD j, $j \in A^{\text{under}}$ cannot accommodate the chunk requiring high bandwidth.

TABLE III
NORMALIZED SSD BANDWIDTH USAGE COMPARED WITH THAT OF FPA.

	Number of SSDs			
Scheme	20	30	40	50
FMA	1	1	1	1
BWAM	1	0.76	0.71	0.77
SWAM	0.99	0.83	0.81	0.85
LWAM	1	0.81	0.67	0.74

2) Effect of Number of Video Chunks: Fig. 7 shows the average and standard deviation of the ADWD values for different numbers of video chunks, and Table IV tabulates the normalized SSD bandwidth usage compared with that of FPA.

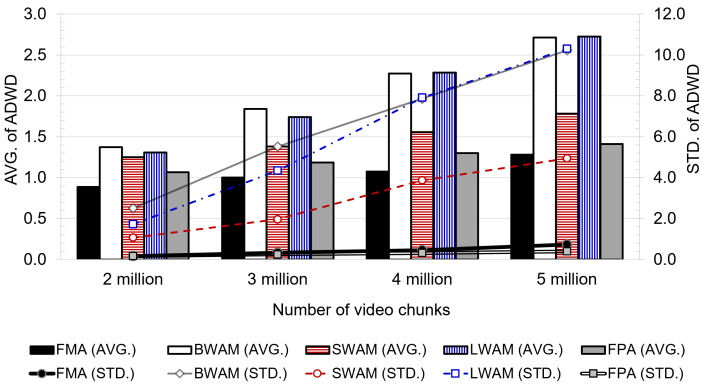

Fig. 7. Average and standard deviation values of ADWD against the number of video chunks.

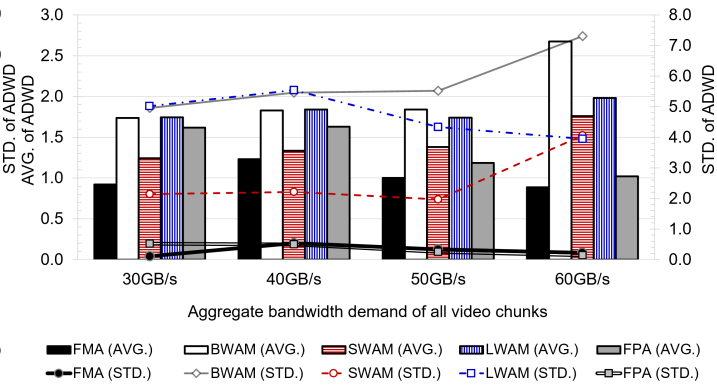

Fig. 8. Average and standard deviation values of ADWD against total bandwidth demand.

TABLE IV
NORMALIZED SSD BANDWIDTH USAGE COMPARED WITH THAT OF FPA.

Number of video chunks				
Scheme	2 million	3 million	4 million	5 million
FMA	1	1	1	1
BWAM	0.9	0.76	0.72	0.7
SWAM	0.92	0.83	0.73	0.71
LWAM	0.9	0.81	0.7	0.68

Again, FMA yields the lowest ADWD but shows the same SSD bandwidth utilization as FPA. For example, the average is 9.28% to 52.95% (mean 33.85%), and the standard deviation is up to 94.35% (mean 58.09%) lower than those of the other schemes.

Obviously, as the number of video chunks increases, more migrations are required, and thus, ADWD increases. This increase is relatively small in FMA; therefore, when the number of video chunks is large, the performance difference increases compared with the other schemes. As the number of chunks increases, the ADWD of BWAM, which requires more swapping, increases significantly. Among the other schemes, SWAM shows the best performance, suggesting that storage space should be used efficiently.

3) Effect of Aggregate Bandwidth Demand of Video Chunks: Fig. 8 shows how the average and standard deviation of the ADWD values vary with the total bandwidth demand, and Table V tabulates the normalized SSD bandwidth usage compared with that of FPA. FMA yielded the lowest ADWD. For example, its average is 7.7% to 66.89% (mean 36.05%), and its standard deviation is up to 97.9% (mean 67.71%) lower than that of the other schemes. The ADWD value of FMA is hardly affected by the total bandwidth demands of the video chunks, unlike in the other schemes.

TABLE V
NORMALIZED SSD BANDWIDTH USAGE COMPARED WITH THAT OF FPA.

Aggregate bandwidth demand of video chunks				
Scheme	30GB/s	40GB/s	50GB/s	60GB/s
FMA	1	1	1	1
BWAM	0.71	0.66	0.76	0.94
SWAM	0.83	0.71	0.83	0.93
LWAM	0.7	0.63	0.81	1

TABLE VI
AVERAGE ALGORITHM EXECUTION TIME (SECONDS).

Number of SSDs				
Scheme	20	30	40	50
FMA	0.7	1.165	1.001	0.812
FPA	4.925	6.505	7.563	6.28
Number of video chunks				
Scheme	2 million	3 million	4 million	5 million
FMA	0.798	1.165	1.75	2.737
FPA	4.146	6.505	9.012	11.157
Aggregate bandwidth demand of video chunks				
Scheme	30GB/s	40GB/s	50GB/s	60GB/s
FMA	1.135	1.952	1.165	1.113
FPA	4.431	5.94	6.505	6.064

C. Algorithm Execution Time

To evaluate the overhead of the algorithms, we measured the average execution times on a Xeon W-2235 CPU. Table VI shows how the execution time depends on the number of SSDs, the number of video chunks, and the total bandwidth demand of the video chunks. Since it only takes 1.29 seconds on average to run FMA, the overhead is not high. Because file migration occurs during off-peak hours, this overhead is almost negligible. Algorithms are also scalable because they are hardly affected by the number of SSDs and total bandwidth demand.

D. Evaluating the Feasibility of the Proposed Scheme on Real HDFS

Using 6 TLC SSDs (SK Hynix Gold S31) and 18 QLC SSDs (Samsung 870 QVO), we built an actual HDFS-based storage system as shown in Fig. 9. The specifications of the two SSD models are listed in Table II. A Hadoop cluster was then configured using Docker, a container-based virtualization technique used to implement multiple services in a single system. The cluster contained a NameNode, a secondary NameNode, and 6 DataNodes, each of which communicates with each other using the virtual IPs shown in Table VII. Each of the 6 DataNodes uses 3 QLC SSDs and 1 TLC SSD.

To verify that the new tools (*FilePlacement* and *FileMovement*) were working properly, a feasibility test was performed on a modified HDFS system. We first created 1000 video chunks to run FPA. The bandwidth requirement for each chunk

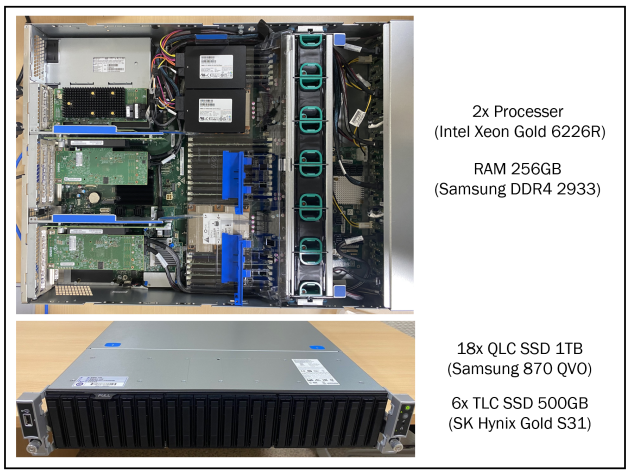

Fig. 9. Actual system configuration.

TABLE VII
LIST OF VIRTUAL IP ADDRESSES GIVEN TO EACH NODE IN HDFS.

Node	Virtual IP address
NameNode	172.20.0.2
Secondary NameNode	172.20.0.3
DataNode1	172.20.0.4
DataNode2	172.20.0.5
DataNode3	172.20.0.6
DataNode4	172.20.0.7
DataNode5	172.20.0.8
DataNode6	172.20.0.9

was calculated using a Zipf probability distribution, where $\theta = 0.271$.

As a result of FPA execution, a file, *placementInfo.in*, was created to record the chunk placement results. Subsequently, when the *'hdfs FilePlacement'* command was executed, all chunks were placed as recorded in the *placementInfo.in* file. Fig. 10 shows an example of the file placement results for a real HDFS. Each chunk corresponds to a block in HDFS and is assigned a block ID separately. Within HDFS, chunks can only be identified by these block IDs. Each block ID could be found as a result of the command execution (*hdfs fsck [FILE PATH] -files -blocks -locations*), as shown in Fig. 10 (a), where block ID 1073741825 was assigned to a file *alpha-1.mp4*. This file was then placed on QLC 2 on DataNode 4, as specified in the *placementInfo.in* file shown in Fig. 10 (b).

To check the result of chunk migration by FMA in HDFS, 76 new popular chunks were added; therefore, FMA determined the SSD location for a total of 1076 chunks. To model the change in I/O bandwidth, the popularity of existing video chunks was changed randomly. As a result of FMA execution, a new file, *movementInfo.in*, was created to record the chunk migration results. The *'hdfs FileMovement'* command was then executed to reflect this chunk migration to the actual HDFS. An example of the file migration results is shown in Fig. 11, which indicates that the chunks were migrated as determined by FMA.

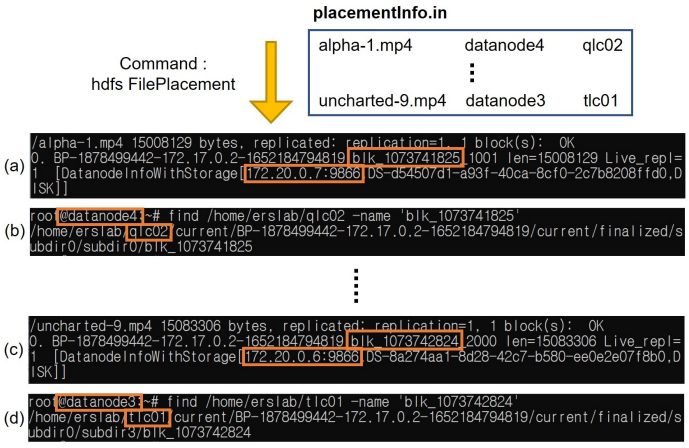

Fig. 10. Example that confirms the result of file placement in actual HDFS.

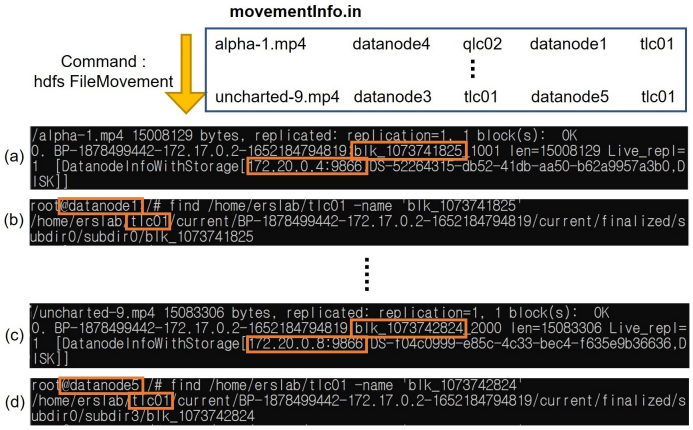

Fig. 11. Example that confirms the result of file migration in actual HDFS.

E. ADWD Results on Real HDFS

To validate wear-leveling among SSDs, FMA was executed, and ADWD was derived daily for 7 days on real HDFS in Fig. 9. The video and chunk popularity model is assumed to follow a Zipf distribution with the same parameters as the simulation, based on which the bandwidth requirement for each chunk can be calculated when the total bandwidth requirement of all video chunks is 5GB/s.

Initially, 200,000 video chunks of 15MB are placed by the FPA and the ADWD value of each SSD is initialized to 1. A total of 2000 video chunks are added every day, and the popularity for each chunk is recalculated in the same way as the simulation. Based on the changed popularity, FMA runs daily and chunks are migrated among SSDs if necessary. For accurate calculation of ADWD, a utility program called Smartmontools was used to check the actual number of writes on each SSD.

We compared the ADWD values based on the number of writes provided by Smartmontools with those values calculated by FMA. Fig. 12 shows the average ADWD values over 7 days against the number of SSDs in an array with a TLC to QLC ratio of 1:3. Actual measured ADWD values are almost identical to those calculated by FMA, where the difference ranges between 0.72% and 0.94%. This small difference can

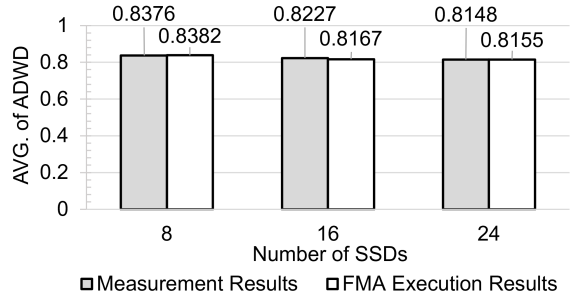

Fig. 12. Values of ADWD against the number of SSDs in actual HDFS.

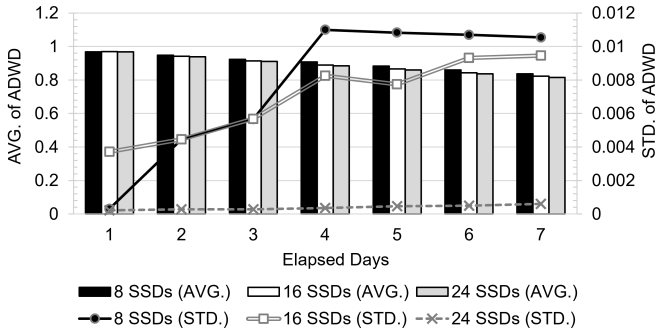

Fig. 13. Daily actual ADWD trend for 7 Days in actual HDFS.

be attributed to the additional metadata writing overhead or the write size granularity provided Smartmontools. For example, some SSD vendors only provide information on the write size in gigabytes, so writes less than 1 GB may not be ascertainable. This suggests that FMA can be effectively used in real HDFS.

Fig. 13 shows the average and standard deviation values of ADWD over all SSDs after daily FMA execution. The standard deviation of the ADWD values for each SSD is very low, up to 0.11, indicating that wear-leveling is effectively achieved. For example, Table VIII tabulates the ADWD value for each SSD after 7 days, confirming these wear-leveling characteristics.

VIII. Conclusions

We proposed new data placement and migration techniques for wear-leveling of heterogeneous SSDs in a storage array while effectively using limited I/O bandwidth and storage space. We first presented a file placement algorithm that maximizes bandwidth usage while effectively utilizing the limited storage space of SSDs. Based on this, we proposed a new file migration algorithm to control the number of write operations on SSDs by considering the DWPD and lifetime of heterogeneous SSDs. For this purpose, it determines the SSD index for each file after migration with the aim of minimizing the overall ADWD, and divides the process into two phases: elimination of overloaded SSDs and BSR enhancement. To apply these algorithms to an actual HDFS, we implemented new file allocation modules inside HDFS and a set of tools that utilize them.

Simulations were performed to examine how ADWD and SSD bandwidth utilization depend on the number of SSDs, the number of video chunks, and the bandwidth requirements.

TABLE VIII
ADWD value for each SSD after 7 days.

Configuration		8 SSDs	16 SSDs	24 SSDs
DataNode1	TLC01	0.8301	0.8108	0.8144
	QLC01	0.8442	0.8172	0.8158
	QLC02	0.8438	0.8141	0.8158
	QLC03	0.8523	0.8130	0.8157
DataNode2	TLC01	0.8249	0.8334	0.8140
	QLC01	0.8459	0.8326	0.8157
	QLC02	0.8202	0.8109	0.8157
	QLC03	0.8395	0.8109	0.8151
DataNode3	TLC01	-	0.8249	0.8144
	QLC01	-	0.8109	0.8160
	QLC02	-	0.8109	0.8157
	QLC03	-	0.8109	0.8156
DataNode4	TLC01	-	0.8354	0.8148
	QLC01	-	0.8110	0.8156
	QLC02	-	0.8109	0.8158
	QLC03	-	0.8110	0.8156
DataNode5	TLC01	-	-	0.8144
	QLC01	-	-	0.8151
	QLC02	-	-	0.8155
	QLC03	-	-	0.8158
DataNode6	TLC01	-	-	0.8144
	QLC01	-	-	0.8160
	QLC02	-	-	0.8156
	QLC03	-	-	0.8157

In all cases, the proposed algorithms achieve the highest SSD bandwidth utilization. In particular, the file migration algorithm can reduce the mean of ADWD by 7.7% to 67.19% (mean 35.44%) and its standard deviation up to 99.37% (mean 69.78%) compared to other methods, which implies that it is very effective at balancing the wear-out of different SSDs. The effectiveness of the proposed algorithms in an actual HDFS was also verified using a Docker-based cluster system.

We believe that our algorithms can be effectively used to cost-effectively build SSD-based VoD storage systems, because they are developed and evaluated for VoD workloads such as Netflix. In addition, the proposed method can be used to improve SSD wear-leveling in various storage servers, thereby contributing to reducing data center maintenance costs. As our future work, we are going to develop redundancy optimization techniques that take into account SSD endurance and fault tolerance in storage arrays.

Acknowledgements

The authors would like to thank Chanyoung Park (Hanyang University, Ansan) for helping build an actual HDFS-based storage system.

References

[1] *Hours of video uploaded to YouTube every minute as of February 2020*. Accessed: July 2022. [Online] Available: https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/.

[2] D. Krishnappa, M. Zink, and R. Sitaraman, "Optimizing the video transcoding workflow in content delivery Networks," In *Proc. ACM Multimed. Syst. Conf.*, Mar. 2015, pp. 37–48.

[3] D. Vergados, A. Michalas, A. Sgora, D. Vergados, and P. Chatzimisios, "FDASH: A fuzzy-based MPEG/DASH adaptation algorithm," *IEEE Syst. J.*, vol. 10, no. 2, pp. 859–868, Jun. 2016.

[4] R. Aparicio-Pardo, K. Pires, A. Blanc, and G. Simon, "Transcoding live adaptive video streams at a massive scale in the cloud," In *Proc. ACM Multimed. Syst. Conf.*, Mar. 2015, pp. 49–60.

[5] H. Zhao, Q. Wang, J. Wang, B. Wan, and Z. Wu, "Popularity-based and version-aware caching scheme at edge servers for multi-version VoD systems," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 3, pp. 1234–1248, Mar. 2021.

[6] C. Zhou, C. Lin, X. Zhang, and Z. Guo, "A control-theoretic approach to rate adaption for DASH over multiple content distribution servers," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 4, 681–694, Apr. 2014.

[7] J. Niu, J. Xu, and L. Xie, "Hybrid storage systems: A survey of architectures and algorithms," *IEEE Access*, vol. 6, pp. 13385–13406, Feb. 2018.

[8] W. Li, G. Jean-Baptise, J. Riveros, G. Narasimhan, T. Zhang, and M. Zhao, "Cachededup: In-line deduplication for flash caching," In *Proc. USENIX Conf. on File and Storage Tech.*, Feb. 2016, pp. 301–314.

[9] D. Arteaga, J. Cabrera, J. Xu, S. Sundararaman, and M. Zhao, "Cloudcache: On-demand flash cache management for cloud computing," In *Proc. USENIX Conf. on File and Storage Tech.*, Feb. 2016, pp. 301–314.

[10] M. Ryu, H. Kim, and U. Ramachandran, "Impact of flash memory on video-on-demand storage: Analysis and Tradeoffs," In *Proc. ACM Multimed. Syst.*, Feb. 2011, pp. 175–186.

[11] M. Ryu and U. Ramachandran, "Flashstream: A multi-tiered storage architecture for adaptive HTTP Streaming," In *Proc. ACM Multimed. Conf.*, Oct. 2013, pp. 313–322.

[12] M. Lin, R. Chen, L. Lin, X. Li, and J. Huang, "Buffer-aware data migration scheme for hybrid storage systems," *IEEE Access*, vol. 6, pp. 47646–47656, Aug. 2018.

[13] S. Ahmadian, R. Salkhordeh, O. Mutlu, and H. Asadi, "Etica: Efficient two-level I/O caching architecture for virtualized platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 10, pp. 2415–2433, Oct. 2021.

[14] *Micron technical brief*. Accessed: July 2022. [Online] Available: https://www.micron.com/-/media/client/global/documents/products/technical-marketing-brief/qlc_technology_high-level_tech_brief.pdf.

[15] N. Papandreou et al., "Open block characterization and read voltage calibration of 3D QLC NAND flash," In *Proc. IEEE Int. Reliab. Phys. Symp.*, Apr. 2020, pp. 1–6.

[16] Y. Takai, M. Fukuchi, R. Kinoshita, C. Matsui, and K. Takeuchi, "Analysis on heterogeneous SSD configuration with quadruple-level cell (QLC) NAND flash memory," In *Proc. IEEE Int. Mem. Workshop*, May 2019, pp. 1–6.

[17] S. Hwang, S. Moon, J. Jung, D. Kim, I. Park, J. Ha, and Y. Lee, "Energy-efficient symmetric BC-BCH decoder architecture for mobile storages," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 4, pp. 1632–1647, Apr. 2021.

[18] S. Maneas, K. Mahdaviani, T. Emami, and B. Schroeder, "Operational characteristics of SSDs in enterprise storage systems: A large-scale field study," In *Proc. USENIX Conf. on File and Storage Tech.*, Feb. 2022, pp. 165–180.

[19] *WD and Tosh talk up penta-level cell flashes*. Accessed: July 2022. [Online] Available: https://blocksandfiles.com/2019/08/07/penta-level-cell-flash/.

[20] Q. Li, H. Li, and K. Zhang, "A survey of SSD lifecycle prediction," In *Proc. IEEE Int. Conf. Softw. Eng. Serv. Sci.*, Oct. 2019, pp. 195–198.

[21] W. Wang, T. Xie, and A. Sharma, "Swans: An interdisk wear-leveling strategy for RAID-0 structured SSD arrays," *ACM Trans. Stor.*, vol. 12, no. 3, pp. 1–21, Apr. 2016.

[22] N. Zhao et al., "Chameleon: An adaptive wear balancer for flash clusters," In *Proc. IEEE Int. Parallel Dist. Process. Symp.*, May 2018, pp. 1163–1172.

[23] C. Xiao, L. Cheng, L. Zhang, D. Liu, and W. Liu, "Wear-aware memory management scheme for balancing lifetime and performance of multiple NVM slots," In *Proc. Symp. Mass Stor. Syst. Technol.*, May 2019, pp. 148–160.

[24] J. Huang, A. Badam, and L. Caulfield, "FlashBlox: achieving both performance isolation and uniform lifetime for virtualized SSDs," In *Proc. USENIX Conf. on File and Storage Tech.*, Feb. 2017, pp. 375–390.

[25] G. Xie, Z. Li, M. Kaafar, and Q. Wu, "Access types effect on Internet video services and its implications on CDN caching," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 5, pp. 1183–1196, May. 2018.

[26] J. He, D. Wu, Y. Zeng, X. Hei, and Y. Wen, "Toward optimal deployment of cloud-assisted video distribution services," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 10, pp. 1717–1728, Oct. 2013.

[27] X. Zhang, D. Xiong, K. Zhao, C. Chen, and T. Zhang, "Realizing low-cost flash memory based video caching in content delivery systems," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 4, pp. 984–996, Apr. 2018.

[28] A. Rahiman and B. Karim, "Continuous media (CM) data stream in flash-based solid state disk (SSD) Storage Server," In *Proc. Int. Conf. Commun. Eng. Technol.*, Apr. 2019, pp. 11–16.

[29] D. He, J. Jiang, T. Ma, G. Yang, C. Westphal, J. Garcia-Luna-Aceves, and S. Xia, "CUBIST: High-quality 360-degree video streaming services via tile-based edge caching and FoV-adaptive prefetching," In *Proc. IEEE Int. Conf. on Web Services*, Sep. 2021, pp. 208–218.

[30] M. Song, "Minimizing power consumption in video servers by the combined use of solid-state disks and multispeed disks," *IEEE Access*, vol. 6, pp. 25737–25746, May 2018.

[31] Y. Gao, H. Zhang, Y. Zhu, B. Tang, and H. Ma, "A load-aware data migration scheme for distributed surveillance video processing using hybrid storage architecture," In *Proc. IEEE Int. Conf. High Perform. Comput. Commun.*, Dec. 2017, pp. 563–570.

[32] Z. Sun, Q. Zhang, Y. Li, and Y. Tan, "DPPDL: A dynamic partial-parallel data layout for green video surveillance storage," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 1, pp. 193–205, Jan. 2018.

[33] O. Al-wesabi, P. Sumari, and N.Abdullah, "Data stream management system for video on demand hybrid storage server," *Int. J. Intell. Syst. Technol. Appl.*, vol. 18, no. 5, pp. 470–493, Jul. 2019.

[34] R. Salkhordeh, S. Ebrahimi, and H. Asadi, "ReCA: An efficient reconfigurable cache architecture for storage systems with online workload characterization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 7, pp. 1605–1620, Jul. 2018.

[35] J. Liu, Y. Chai, X. Qin, and Y. Liu, "Endurable SSD-based read cache for improving the performance of selective restoration of deduplication systems," *J. Comput. Sci. Technol.*, vol. 33, pp. 58–78, Jan. 2018.

[36] Y. Chai, Z. Du, X. Qin, and D. Bader, "WEC: Improving durability of SSD cache drives by caching write-efficient data," *IEEE Trans. Comput.*, vol. 64, no. 11, pp. 3304–3316, Nov. 2015.

[37] C. Ma, Z. Zhou, L. Han, Z. Shen, Y. Wang, R. Chen and Z. Shao, "Rebirth-FTL: Lifetime optimization via approximate storage for NAND flash memory (early access)," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, pp. 1–16, Oct. 2021.

[38] J. Yen et al., "Efficient bad block management with cluster similarity," In *Proc. IEEE Int. Symp. High-perf. Comput. Arch.*, Apr. 2022, pp. 503–513.

[39] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "Hybridstore: a cost-efficient, high-performance storage system combining SSDs and HDDs," *Proc. IEEE Int. Symp. Model., Anal., and Simul. of Comput. and Telecomm. Syst.*, Jul. 2011, pp. 227–236.

[40] H. Yoon, Y. Woo, D. Lee, Y. Moon, and H. Kwon, "Semiconductor storage device and method of throttling performance of the same," In *US patent*, Feb. 2012, US20120047317.

[41] G. Tressler and A. Walls, "Enabling throttling on average write throughput for solid state storage devices," In *US patent*, Apr. 2013, US20130086302.

[42] S. Lee, T. Kim, K. Kim, and J. Kim, "Lifetime management of flash-based SSDs using recovery-aware dynamic throttling," In *Proc. USENIX Conf. on File and Storage Tech.*, Feb. 2012, pp. 537—540.

[43] S. Lee and J. Kim, "Effective lifetime-aware dynamic throttling for NAND flash-based SSDs," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1331–1334, Apr. 2016.

[44] Y. Kang, X. Zhang, Z. Shao, R. Chen, and Y. Wang, "A reliability enhanced video storage architecture in hybrid SLC/MLC NAND flash memory," *J. Syst. Arch.*, vol. 88, pp. 33–42, Aug. 2018.

[45] J. Lee, H. Han, S. Lee, and M. Song, "Lifetime-aware solid-state disk (SSD) cache management for video servers," *Multimed. Syst.*, vol. 25, pp. 695—-708, May 2019.

[46] C. Liu, K. Fan, Z. Yang, and J. Xiu, "A distributed video share system based on Hadoop," In *Proc. IEEE Int. Conf. on Cloud Comput. Intell. Syst.*, Nov. 2014, pp. 587–590.

[47] A. Al-Abbasi and V. Aggarwal, "Video streaming in distributed erasure-coded storage systems: Stall duration analysis," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp.1921–1932, Aug. 2018.

[48] E. SaatiAlsoruji, "A novel snake-like data placement policy for improving video processing in the Hadoop clusters," In *Proc. IEEE Conf. Big Data Anal.*, Nov. 2020, pp. 12—19.

[49] X. Dai, X. Wang, and N. Liu, "Optimal scheduling of data-intensive applications in cloud-based video distribution services," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 1, pp. 73–83, Jan. 2017.

[50] A. Dan and D. Sitaram. "An online video placement policy based on bandwidth to space ratio (BSR)," In *Proc. ACM SIGMOD Int. Conf. Manag. Data*, May 1995, pp. 376–385.

[51] Y. Guo, Z. Ge, B. Urgaonkar, P. Shenoy, and D. Towsley, "Dynamic cache reconfiguration strategies for cluster-based streaming proxy," *Comput. Commun.*, vol. 29, no. 10, pp. 1710–1721, Aug. 2006.

[52] Y. Zhou, L. Chen, C. Yang, and D. Chiu, "Video popularity dynamics and its implication for replication," *IEEE Trans. Multimed.*, vol. 17, no. 8, pp. 1273–1285, Aug. 2015.

[53] Y. Akçay, H. Li, and S. Xu, "Greedy algorithm for the general multidimensional knapsack problem," *Ann. Oper. Res.*, vol. 150, no. 17, pp. 17–29, Mar. 2007.

[54] K. Qu, L. Meng, and Y. Yang, "A dynamic replica strategy based on Markov model for Hadoop distributed file system (HDFS)," In *Proc. Int. Conf. on Cloud Comput. Intell. Syst.*, Aug. 2016, pp. 337–342.

[55] H. Wang, B. Xiao, L. Wang, F. Zhu, Y. Jiang and J. Wu, "CHCF: A cloud-based heterogeneous computing framework for large-scale image retrieval," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 12, pp. 1900–1913, Dec. 2015.

[56] *HDFS Architecture*. Accessed: July 2022. [Online] Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/hdfsdesign.html.

[57] *HDFS Configuration*. Accessed: July 2022. [Online] Available: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml.

[58] *HDFS Users Guide*. Accessed: July 2022. [Online] Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/hdfsuserguide.html.

[59] *HDFS Disk Balancer*. Accessed: July 2022. [Online] Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/hdfsdiskbalancer.html.

[60] *Archival Storage, SSD & Memory*. Accessed: July 2022. [Online] Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/archivalstorage.html.

[61] *Smartmontools official site* . Accessed: October 2022. [Online] Available: https://www.smartmontools.org/.

[62] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *ACM/Springer Multimed. Syst. J.*, vol. 4, no. 3, pp. 112–121, Jun. 1996.

[63] S. Lim and Y. Ko and G. Jung and J. Kim and P. Suei and M. Yeh and T. Kuo, "Inter-chunk popularity-based edge-first caching in content-centric networking," *IEEE Commun. Lett.*, vol. 18, no. 8, pp. 1331–1334, Feb. 2014.

[64] H. Yu and D. Zheng and B. Zhao and W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," In *Proc. ACM Eurosys Conf.*, Apr. 2006, pp. 333–344.

[65] *BlockPlacementPolicies*. Accessed: October 2022. [Online] Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsBlockPlacementPolicies.html.

Dayoung Lee received the B.S. and M.S. degrees in computer engineering from Inha University, Korea, in 2016 and 2018, respectively. She is currently pursuing the Ph.D. degree from the Department of Computer Engineering at Inha University. Her current research interests include distributed-file systems and multimedia systems.

Joonho Lee received the B.E degree from the University of New South Wales, Australia, and the M.E degree from Seoul National University, Korea. He is currently working for SK Hynix as Principle Engineer.

Minseok Song (M'07) received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Korea, in 1996, 1998, and 2004, respectively. Since September 2005, he has been with the Department of Computer Engineering at Inha University, Korea, where he is currently a professor. His research interests include embedded systems, multimedia and IoT systems.