IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 5, 2018, accepted August 17, 2018, date of publication August 22, 2018, date of current version September 21, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2866573

Buffer-Aware Data Migration Scheme for Hybrid Storage Systems

MINGWEI LIN[1], RIQING CHEN[2], LI LIN[1], XUAN LI[1], AND JINGCHANG HUANG[3]

[1]College of Mathematics and Informatics, Fujian Normal University, Fuzhou 350117, China
[2]Institute of Big Data for Agriculture and Forestry, Fujian Agriculture and Forestry University, Fuzhou 350002, China
[3]IBM-Research China Lab, Shanghai 201203, China

Corresponding author: Riqing Chen (riqing.chen@fafu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61502102 and 61502103, in part by the Natural Science Foundation of Fujian province, China, under Grant 2016J05149 and 2017J01737, and in part by the Distinguished Young Scientific Research Talent Training Plan in Universities of Fujian province (2017).

ABSTRACT Since solid-state drives (SSD) have high read speed, they are integrated into a traditional hard disk drive (HDD)-based storage systems for improving the overall performance. The data migration, which is responsible for moving the data between the HDD and SSD so as to maximize the I/O performance, may result in the overhead in terms of unnecessary page migrations. To deal with it, we develop a novel buffer-aware data migration scheme to boost the performance for hybrid storage systems consisting of HDD and SSD by exploiting the content information in the buffer cache. It first introduces two new data states to reclassify the data, which are placed on the hybrid storage systems equipped with HDD and SSD. Then, it tries to improve the efficiency of data migration by copying the newly updated data in the buffer cache instead of the old version in the HDD or SSD into the other device. We also model the process of data migration operation and assess the effectiveness of the buffer-aware data migration scheme theoretically. Experimental results reveal that the buffer-aware data migration scheme can reduce the runtime time by up to 18% and the write count by up to 14% over the buffer-unaware data migration scheme under three benchmarks and a real workload.

INDEX TERMS Buffer management, data migration, SSD, HDD.

I. INTRODUCTION

Hard disk drives (HDDs) have been broadly employed as the primary storage medium for computer systems in the past decades due to their large storage capacities and relatively low price per gigabyte. The storage capacity of a single HDD has been increasing to 60TB, but the access performance has been improving slowly and the performance gap between HDD and CPU has been becoming bigger and bigger [1]–[3]. The HDD has become the performance bottleneck of the computer systems. Moreover, with the explosive growth of data, modern computer systems impose higher requirements on their storage performances. Hence, ideal storage systems in current time should not only own big storage capacities, but also have high access performance.

To meet the above-mentioned requirements, the solid-state drives (SSDs) are installed into the traditional HDD-based storage systems so that the HDD/SSD hybrid storage systems can be built to boost the computer performance [4]–[6]. The SSDs are produced using the NAND flash memory (NFM) chip [7]–[9], so they can inherit both the attractive advantages and unique physical characteristics from the NFM chip [10]–[12]. It owns non-volatility, fast random read speed, small size, strong shock resistance, and low power consumption [13].

However, it also shows some physical characteristics. First, it has an erase-before-write limit, which forces all the blocks in the NFM to be erased prior to being written. It often exploits an out-of-update scheme for solving the limit. This scheme writes the modified data into the free area of NFM and then invalidates the original version by marking them as invalid. Next, it can perform three basic operations, which are read, write, and erase. Read operation is issued to read the data from the target page, while write operation is executed to write data to a free page. Therefore, the basic access unit of read and write operations is a page. Erase operation is employed to erase a block after its valid pages are copied to the free space. Hence, the basic access unit of erase operation is a block [14]. Finally, it has asymmetric costs for three basic

I/O operations. Its write operation cost is higher than the read operation cost and its erase operation cost is higher than the write operation cost [15].

To exploit the high random read performance owned by SSD and hide the difference between the HDD and SSD, a number of hybrid storage systems that are composed of the HDD and SSD have been devised to manage the data, which are placed at the HDD/SSD hybrid storage media. However, these systems focus on identifying hot data from cold data accurately and perform data migration operation to copy hot ones from HDD to SSD and cold ones from SSD into HDD for improving the access performance. The data migration may result in the performance overhead since it generates unnecessary page migrations.

To raise the access performance of hybrid storage systems, a buffer cache is usually used to buffer partial data, which may be hit for being accessed in the near future. However, these storage systems do not take full advantage of using a buffer cache. It can be observed that hybrid storage systems usually result in the overhead in terms of unnecessary page migrations during data migration. These unnecessary page migrations occur when the live pages in the storage device, which have the newly updated version in the buffer cache, are migrated between HDD and SSD and then their newly update versions are written back to the storage media.

To eliminate these unnecessary page migrations, we put forward an efficient buffer-aware data migration scheme for HDD/SSD hybrid storage systems. Our contributions can be summarized as follows.

(1) We first show a typical target system architecture to illustrate the HDD/SSD hybrid storage systems that manage both HDD and SSD in the same level of storage hierarchy.

(2) We introduce two new states to reclassify the data stored in the HDD/SSD hybrid storage systems and then describe the transitions among four data states, which are the live, duplicate, obsolete, and dead.

(3) We devise a novel buffer-aware data migration scheme based on the new data states. It transfers the newly updated versions in the buffer instead of moving the original data in the storage device into the other storage device.

(4) We model the buffer-aware and buffer-unaware data migration processes and testify that the buffer-aware data migration scheme performs better than the buffer-unaware one theoretically.

We have conducted a series of experiments to compare the buffer-aware data migration scheme with the buffer-unaware data migration scheme with several different kinds of traces. Our experiment results reveal that the proposed buffer-aware data migration scheme decreases the runtime by up to 18% and eliminates the write count by up to 14% over the buffer-unaware data migration scheme under three benchmarks and a real workload.

The rest of this paper is organized as follows. Related works are briefly reviewed and the motivation of our work is given in Section II. Section III presents our target system architecture. Section IV shows the new data states that are introduced to assort all the data, which are placed at hybrid storage systems. The proposed buffer-aware data migration scheme is described in Section V. Performance evaluation are shown in Section VI. Finally, our work is summarized in Section VII.

II. RELATED WORK AND MOTIVATION

A. EXISTING WORKS ON HYBRID STORAGE SYSTEMS

Aiming to enhance the access performance of hybrid storage systems, a large number of research results have been given.

Based on the placement position of SSD deployed in the storage hierarchy, previous hybrid storage systems consisting of HDD and SSD are divided into three types: 1) SSD as a buffer for HDD, 2) HDD as a buffer for SSD, and 3) HDD and SSD as the same level of the storage hierarchy.

1) SSD AS A BUFFER FOR HDD

In [16], a hybrid hard drive, which introduces a NFM chip to be a buffer of HDD, is developed. Liu *et al.* [17] proposed a hybrid data storage architecture called RAF. It consists of a SSD-based cache and a hard disk. The SSD-based cache is logically divided into a read cache and a write cache. The read cache is dedicated to serve random-access data, while the write cache serves as a circular write-through log. Sehgal *et al.* [18] integrated SSD as a read-write cache for HDD and then devised a SLO-based resource management algorithm, which adjusts the amount of SSD dynamically according to the workload. Lin *et al.* [19] showed a self-optimizing hybrid storage system, named HRO, which utilizes SSD as a by-passable cache for hard disk. The SSD-based cache is used to serve a majority of random I/O accesses. He *et al.* [20] and [43] studied a hybrid architecture called the Smart Selective SSD Cache that uses a small set of SSD-based file servers as a cache for selectively serving the performance critical data for conventional HDD-based file servers. In [21], the SSD is designed as a cache for slow HDD, which migrates partial data from HDD to SSD so as to reduce the application launch time.

2) HDD AS A BUFFER FOR SSD

Yang *et al.* [22] combined SSD and HDD to build a hybrid storage system, named HB-Storage, which manages HDD as a write buffer to optimize the SSD write request. In [23], a hybrid storage device, called Griffin, is put forward to utilize a HDD as a write cache for the SSD. However, the SSD is expensive and cannot be afforded by many companies.

3) HDD AND SSD AS THE SAME LEVEL OF STORAGE HIERARCHY

Jacobi [24] used flash memory-based SSD to supplement or replace hard disks. Applications and files are mapped into SSD for extending battery life and speeding up boot-ups of laptops. Since SSDs have different physical characteristics from HDDs, in [25], a hybrid file system, referred to as the HybridFS, is put forward to manage two kinds of storage

media, which are HDD and SSD. The HybridFS is designed to place data blocks of files at the HDD and then map their metadata into the SSD [26]. In [27], a hybrid copy-on-write (CoW) storage device is devised to combine the SSDs and HDDs for consolidated environments. In the hybrid system, read-only template images are placed at the SSDs, while the write operations are mapped into the HDDs. Hsu and Bai [28] devised a SSD, which was connected to a PC system using the SATA bus. In this work, SSD is used to place frequently accessed files.

No [29] presented a hybrid file system, named N-hybrid, for combining the strengths of the HDD and SSD. Kim *et al.* [30] designed and evaluated a hybrid storage system, called Hybird Store, to obtain the trade-offs between the HDD and SSD in terms of cost, performance, and lifetime. Hui *et al.* [31] proposed a new hybrid storage system, referred to as the E-HASH, which is composed of a SSD and multiple HDDs. In E-HASH, the most frequently read data is stored in the SSD, while all the writes are served by the HDDs. Fisher *et al.* [32] put forward a hybrid SSD-HDD storage system to make full use of the high random access performance of the SSD, as well as the high sequential access performance and the large storage capacity of HDD. In [33], a hybrid storage system, referred to as the PASS, is put forward to obtain the tradeoff between I/O performance and data discrepancy of HDD and SSD. In the PASS, all of the I/O requests are remapped into SSD first and then the updated data blocks are migrated into HDD. Welch and Noer [8] exploited the file size distributions to optimize the data layout of hybrid storage system by storing all of block-level and file-level metadata, and all of small files into SSD and holding large file extents in HDD. In [34], an improved capacity planning technique named the HybridPlan is studied for hybrid storage systems to obtain the most cost-effective hybrid storage configuration with HDD and SSD.

Xu *et al.* [35] proposed a storage architecture called SOHO for write-once-read-once scientific applications to process the raw data in the SSD and move the processed data to the HDD. In [36], a measurement-driven data migration method named MDDM is given for hybrid storage systems to move the data blocks between the HDD and SSD according to their access patterns. Xie and Madathil [37] and Xie and Sun [38] have put forward a file layout strategy, named SAIL, for hybrid disk arrays, which combine the hard disks (HDDs) with flash disks (SSDs) as the data storage devices. In [3], the dying state is utilized to reclassify the data and a victim chunk selection method is put forward for the hybrid storage systems. However, this work does not move the data, which has copies in the buffer cache, between the HDD and SSD. It may lose partial data and incur data inconsistency. Moreover, it does not offer theoretical analysis to support the proposed method. He *et al.* [44]–[47] studied the layout scheme for parallel file systems equipped with HDD and SSD. In [44] and [45], a selection and distribution algorithm is put forward to develop an efficient data layout scheme named HAS to select the optimal data layout solution. In [46], a data layout scheme called HARL is put forward to split a file into several fine-grained regions with varying sizes and then distribute them into different servers based on their performance. In [47], to boost the performance of parallel file systems under heterogeneous access patterns, a data layout scheme called MHA is proposed to move the file data with similar access patterns into different regions and determine the appropriate stripe sizes on the storage servers based on their performance.

It can be seen that existing works on the HDD/SSD hybrid storage systems have not exploited the content information of a buffer cache to optimize the data migration efficiency.

B. OUR MOTIVATION

In this section, a simple sample is introduced to illustrate the benefit of exploiting the content information in the buffer cache to improve the data migration efficiency in the hybrid storage systems, which is considered as the motivation of our work.

When the data migration operation is performed to move the user data between the HDD and SSD, a number of page migrations will be incurred, which not only block the normal I/O operations of hybrid storage systems, but also incur lots of read and write operations. It is observed that most of them would be unnecessary if hybrid storage systems are aware of the content information of a buffer cache.

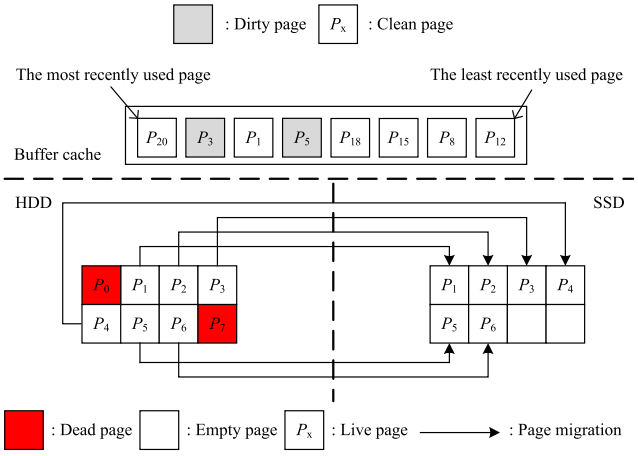


FIGURE 1. An example of unnecessary page migrations.

Figure 1 gives an example of the data migration operation in the hybrid storage systems. Assume that the granularity of the data migration operation is a chunk that has eight pages and the buffer cache can hold eight pages. It can be seen that two pages, p_3 and p_5, in the buffer cache are dirty pages.

To perform the data migration operation as demonstrated in Figure 1, there will be six page migrations in the existing hybrid storage systems. However, it is useless to migrate the pages, p_3 and p_5, from HDD to SSD because p_3 and p_5 will be dead shortly when their newly updated versions p_3 and p_5 in the buffer cache are written back into the hybrid storage medium. How soon the dirty pages will be written back to

the hybrid storage device is determined by the buffer manager of operating system. The time interval for flushing the pages in the buffer is set to 30 seconds by Linux kernel. If the newly updated versions p_3 and p_5 in the buffer cache are migrated to SSD instead of p_3 and p_5 within the HDD during the data migration, two unnecessary page migrations will be eliminated. This is the main motivation of our work.

Exploiting the content information of a buffer cache has two positive impacts on the performance of data migration scheme in the hybrid storage systems. First, it can reduce the number of write operations to the hybrid storage device due to the eviction of dirty pages in the buffer cache. Since the dirty pages p_3 and p_5 in the buffer cache have been written to the SSD during the data migration operation, these pages in the buffer cache have changed to be clean and there is no need to flush them into the hybrid storage device when they are evicted from the buffer cache by the operating systems. Second, it can delay the data migration operation that will be performed from the SSD to HDD in the near future. In the traditional hybrid data storage system, two pages p_3 and p_5 within the HDD are migrated into the SSD during the data migration operation and their newly updated versions must be flushed into the hybrid storage medium when they are evicted from the buffer cache. In this case, eight free pages in the SSD will be occupied. However, if the dirty pages p_3 and p_5 in the buffer cache are directly copied back into the SSD when the data migration operation is performed, two free pages will be saved and the data migration operation from SSD to HDD in the near future will be delayed since the SSD uses an out-of-place update scheme, which could use up its free pages and incur the data migration operation to migrate the data from the SSD to HDD and then make the corresponding data chunk be free. These positive impacts of exploiting the content information in a buffer cache to boost the performance of data migration operation for the hybrid storage systems are called potential benefits since they will be achieved at a later time.

If p_3 and p_5 in the buffer cache are updated after they are copied to SSD during the data migration operation, they should be flushed into the hybrid storage device when they are evicted from the buffer cache in the near future. In this case, these two page migrations for migrating the pages p_3 and p_5 during the data migration operation are useless. This case could impose restriction on its positive effects on the performance of the data migration operation in the hybrid storage systems.

III. TARGET SYSTEM ARCHITECTURE

Figure 2 presents a real architectural overview of our target hybrid storage systems, which are equipped with the HDD and SSD as the storage media at the same level of memory hierarchy. Both of them are connected to the host systems through a standard block device interface such as SATA. The buffer management layer is responsible for managing the buffer cache, which is used to temporarily hold partial data that have high probabilities to be accessed in a short

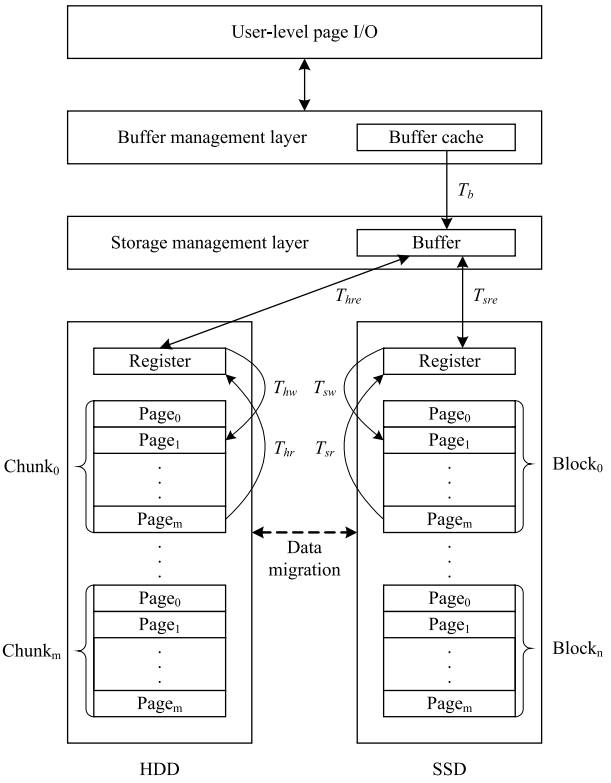


FIGURE 2. Our target system architecture.

time [39], [40]. The storage management layer works as a pseudo block device. Then the upper-level file system can view it as a single block device simply. The storage space of HDD is logically split into a fixed number of chunks, whose size is equal to that of the block in SSD. The storage management layer consists of three major components, namely the address remapper, data mover, and I/O monitor. To reduce the query time, the address remapper should store an address mapping table in the SSD to track the locations of pages in the hybrid storage medium. When the computer system is booted, this address mapping table is loaded into the main memory. To achieve the data consistency during the system failure, the address remapper updates the address mapping table and its in-memory version at the same time. The I/O monitor is in charge of collecting I/O requests and profiling the workload access pattern. The data mover can perform data migrations operation according to the workload access pattern. In this paper, we focus on how to exploit the content information of a buffer cache to boost the performance instead of how to identify the hot degree of each page or each file in the hybrid storage device accurately.

When a page is written from the buffer cache to one of HDD or SSD, it requires data transmissions for three times. First, a page within the buffer cache managed by the buffer management layer is sent into the internal buffer of storage management layer via a system bus. Then, it is transmitted into the register of HDD or SSD. Finally, the page is moved into the target page in the HDD or SSD. Therefore, the time

that is taken to move a page from the buffer cache owned by the buffer management layer to one of the HDD and SSD is $(T_b + T_{re} + T_w)$. The term T_b is the time that is consumed to migrate a page between a buffer cache and the internal buffer of storage management layer. The term T_{re} is the time that is consumed to transfer a page between the internal buffer of storage management layer and the register of the HDD or SSD. The term T_w is the time that is consumed to perform a write operation on the page from the register to the target page. Their values are defined as follows.

$$T_{re} = \begin{cases} T_{hre} & \text{if the target device is a HDD} \\ T_{sre} & \text{if it is a SSD} \end{cases} \quad (1)$$

$$T_w = \begin{cases} T_{hw} & \text{if the target device is a HDD} \\ T_{sw} & \text{if it is a SSD} \end{cases} \quad (2)$$

Reading a target page from HDD or SSD to the internal buffer of storage management layer needs data transmissions for two times. First, a page is moved into the register of HDD or SSD. Then, the page is transferred into the internal buffer of storage management layer. Hence, the time taken to read a page from the HDD or SSD to the internal buffer of storage management layer is $(T_r + T_{re})$, where T_r is the time that is consumed to read a page from the physical page into the register in the HDD or SSD. The value of T_r is defined as follows.

$$T_r = \begin{cases} T_{hr} & \text{if the target device is a HDD} \\ T_{sr} & \text{if it is a SSD} \end{cases} \quad (3)$$

A total data migration operation between HDD and SSD requires several data transmissions. Suppose that a page is moved from HDD to SSD. First, a page in the chunk is read into the register of HDD and then it is transferred into the internal buffer of the storage management layer. Next, it is moved into the register of SSD. Finally, it is written to the target page in the SSD. Hence, the total time that is taken to migrate a page from HDD to SSD is $(T_{hr} + T_{hre}) + (T_{sre} + T_{sw})$ and the total time required to move a page from SSD to HDD is $(T_{sr} + T_{sre}) + (T_{hre} + T_{hw})$.

IV. DATA STATES

In this section, we introduce two new liveness states, called obsolete and duplicate, to redefine the states of data, which was placed at the hybrid storage device.

As illustrated in Figure 3, the data stored in the existing storage systems has two states, which are the live and dead. When the new data is written to the storage device by the host, the data owns a live state in the storage device. If the data is modified and its new version is written to the storage device, the state of the previous live data within the storage device will changed to be dead. If a file that contains the live data has been deleted by the upper-level file system, then the state of the live data will also become dead.

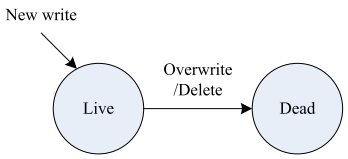


FIGURE 3. Transition between live and dead states.

As mentioned above, a buffer cache is usually designed as a supplement of main memory to temporarily hold a portion of data, which has a high probability to be hit in a short time, for improving the I/O performance. When files are read, their data are loaded from the data storage device into the main memory and a portion of the data may be updated soon. We have analyzed that exploiting the content information in the buffer cache can eliminate the unnecessary page migrations. Hence, we introduce another two new states, called obsolete and duplicate, to reclassify all the data that are placed at the hybrid storage device.

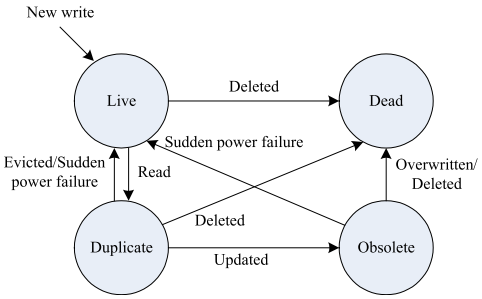


FIGURE 4. Transitions between live, duplicate, obsolete, and dead states.

As presented in Figure 4, the liveness state of each page within the hybrid storage device, which is composed of HDD and SSD, is divided into four types: live, duplicate, obsolete, and dead. The live data is the data, which is not dead and does not have any copy in the buffer cache. If the live data is loaded into the main memory, its state becomes duplicate since it has a copy in the buffer cache. When the copy is modified, then its state becomes obsolete. When the newly updated version in the buffer cache is written back into the hybrid storage device, its state becomes dead. The state of duplicate data will become live if the host system suffers from the sudden power failure or its copy within the buffer cache is flushed out. The state of obsolete data will become live when the host system suffers from the sudden power failure, because its newly updated version in the buffer cache will be lost.

In this paper, we use two reserved bits of the data structure representing the page to implement its state. For example, 00 stands for the live page, 01 for the duplicate page, 10 for the obsolete page, and 11 for the dead page.

V. BUFFER-AWARE DATA MIGRATION SCHEME

The buffer-aware data migration scheme aims at optimizing the data migration process. In order to decrease the useless page migrations, the buffer-aware data migration process is designed to write the data within the buffer cache back into

the target data storage device directly only if the states of the migrated data are duplicate or obsolete.

A. BUFFER-AWARE DATA MIGRATION ALGORITHM

The process of data migration from HDD to SSD is the same as that of data migration from SSD to HDD, so we only show the buffer-aware data migration algorithm that migrates the data from HDD to SSD in this subsection.

```
1: Buffer_Aware_Data_Migration (Ci) {
2:     obtain a new block Bj from a free block list in the SSD;
3:     for pk ∈ S(Ci) {
4:         if pk is duplicate or obsolete {
5:             write its copy in the buffer cache to Bj;
6:             if its copy is dirty
7:                 make it clean;
8:         }
9:         else if pk is live{
10:            migrate pk from Ci to Bj;
11:            update the address mapping table;
12:        }
13:        else if pk is dead{
14:        }
15:    }
16: }
```

FIGURE 5. The algorithm for buffer-aware data migration from HDD to SSD.

Figure 5 gives the buffer-aware data migration algorithm to show that how a data migration operation works in detail. Suppose that a chunk C_i in the HDD is selected as a victim chunk. The term S (C_i) denotes a set of pages in the chunk C_i. For each page p_k within the chunk C_i, the algorithm determines if p_k is a duplicate or obsolete page. If it is obsolete, it means that p_k in the chunk C_i is out-of-date and it has the newly updated version in the buffer cache. Hence, copying p_k in the chunk C_i to a free block B_j in the SSD is useless. To eliminate unnecessary page migrations and also prolong the lifetime of SSD, the algorithm writes the newly updated version in the buffer cache to the free block B_j with the lowest erase count in the SSD and makes it clean. If the newly updated version of page p_k in the buffer cache is not modified again before being removed from the buffer cache, then a write operation that flushes a dirty page into the SSD will be eliminated.

If p_k is a duplicate page, it means that this page owns a equal copy within the buffer cache. The algorithm migrates its copy in the buffer cache into the SSD instead of p_k in the chunk C_i. That is because the time that is consumed to write a page from the buffer cache into SSD is less than the time that is consumed to migrate a page from the HDD to the SSD. It can be proved as follows.

As discussed in Section III, the time, which is consumed to write a page from a buffer cache into SSD, is $(T_b + T_{sre} + T_{sw})$, while the time, which is taken to migrate a page from HDD into SSD, is calculated as $(T_{hr} + T_{hre}) + (T_{sre} + T_{sw})$. Because the data transmission between the buffer cache and storage management layer is performed via a high-speed system bus, the value of T_b is too small so that it can be negligible. In our work, T_b is assumed to be 0. Hence,

$$(T_b + T_{sre} + T_{sw}) < (T_{hr} + T_{hre}) + (T_{sre} + T_{sw}) \quad (4)$$

B. EFFECT OF BUFFER-AWARE DATA MIGRATION

The buffer-aware data migration scheme leads to lower cost than existing buffer-unaware data migration schemes since the buffer-aware one fully exploits the content information of a buffer cache to eliminate useless page migrations between the HDD and SSD. To help analyze the performance effect of our proposed buffer-aware data migration scheme on the performance of the hybrid storage systems, the buffer-aware data migration cost is first compared with the buffer-unaware data migration cost.

In the buffer-unaware data migration scheme, all the data that are placed within the hybrid storage device consisting of the HDD and SSD are simply divided into live and dead data. When the data migration operation is performed, all the live data in the victim chunk are migrated between the HDD and SSD. Therefore, it can be derived that the data migration cost mainly depends on the number of pages that are transferred between HDD and SSD.

Definition 1: Let IA (C_i) be a set of all live pages, which are migrated from HDD into SSD when a chunk C_i is selected for data migration and let $|\text{IA}\ (C_i)|$ be the number of pages in IA (C_i). If the time taken to migrate a single page from the HDD into SSD is $T_{h\to s}$, then the buffer-unaware data migration cost to migrate all the live pages in the chunk C_i, which is denoted by $T_{dm}^{BU}\ (C_i)$, is defined as follows:

$$T_{dm}^{BU}\ (C_i) = |\text{IA}\ (C_i)| * T_{h\to s} \quad (5)$$

For the example that is presented in Figure 1, IA (C_i) is $\{p_1, p_2, p_3, p_4, p_5, p_6\}$ and $|\text{IA}\ (C_i)|$ equals 6. As analyzed in Section III, $T_{h\to s}$ is equal to $(T_{hr} + T_{hre}) + (T_{sre} + T_{sw})$.

To eliminate useless page migrations, if the selected victim chunk has duplicate and/or obsolete data, the buffer-aware data migration scheme directly writes their newly updated copies in the buffer cache into the SSD instead of duplicate and/or obsolete pages in the HDD. It decreases unnecessary page migrations and the number of read operations, which are performed to read the target data from the HDD. Hence, the buffer-aware data migration cost is define as follows:

Definition 2: Let IO (C_i) be a set of obsolete pages that are in the chunk C_i, ID (C_i) be a set of duplicate pages, and IL (C_i) be a set of live pages. Then $|\text{IO}\ (C_i)|$, $|\text{ID}\ (C_i)|$, and $|\text{IL}\ (C_i)|$ denote the number of pages in the corresponding sets. If the time consumed to write a page from the buffer cache to the SSD is denoted by $T_{b\to s}$, then the buffer-aware data migration cost for migrating the pages in the chunk C_i, which is denoted by $T_{dm}^{BA}\ (C_i)$, is defined as

$$T_{dm}^{BA}(C_i) = (|\text{IO}(C_i)| + |\text{ID}(C_i)|) * T_{b\to s} + |\text{IL}(C_i)| * T_{h\to s} \quad (6)$$

$T_{b\to s}$ is also denoted as $(T_b + T_{sre} + T_{sw})$. For the example of Figure 1, IO (C_i), ID (C_i), and IL (C_i) are $\{p_3, p_5\}$, $\{p_1\}$, and $\{p_2, p_4, p_6\}$. As discussed in Section IV, the live state introduced in the traditional storage systems is further classified into three states: live, duplicate, and obsolete, so $|\text{IO}\,(C_i)| + |\text{ID}\,(C_i)| + |\text{IL}\,(C_i)| = |\text{IA}\,(C_i)|$.

Then, (5) could be transformed as

$$T_{dm}^{BU}(C_i) = (|\text{IO}(C_i)| + |\text{ID}(C_i)|) * T_{h\to s} + |\text{IL}(C_i)| * T_{h\to s} \tag{7}$$

Since $T_{b\to s} < T_{h\to s}$ has been proved in the first subsection of Section V, $T_{dm}^{BA}(C_i) < T_{dm}^{BU}(C_i)$. As depicted in Figure 1, three read operations for reading the pages p_1, p_3, and p_5 from HDD are eliminated in the buffer-aware data migration scheme.

As depicted in (6), as a victim chunk that is selected for the data migration has more duplicate pages and/or obsolete pages, the buffer-aware data migration cost becomes lower because more read operations, reading duplicate pages and/or obsolete pages from HDD, can be eliminated.

C. POTENTIAL BENEFITS OF BUFFER-AWARE DATA MIGRATION

As mentioned in Section II, the buffer-aware data migration scheme owns two potential benefits, which can decrease the future replacement cost for the buffer management layer in the Linux operating system and the future data migration cost for migrating the data from SDD to HDD. The decreasing of future replacement cost results from the reduction of future write operations into SSD and the reduction of future data migration cost comes from the delay of future data migration operations.

In order to help understand these two potential benefits, the process of the buffer-unaware data migration scheme is compared to that of the buffer-aware data migration scheme by utilizing two examples demonstrated in Figures 6 and 7. Initially, the buffer cache holds four pages, p_6, p_8, p_2, and p_3, respectively. Each chunk or block can hold four pages.

As shown in Figure 6, the buffer-unaware data migration scheme migrates two pages p_2 and p_3 from the HDD into

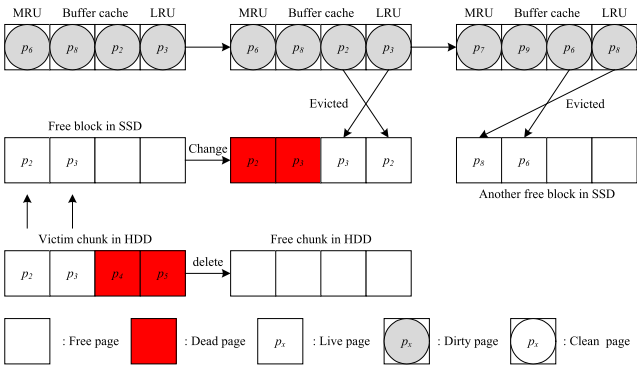


FIGURE 6. The process of buffer-unaware data migration scheme.

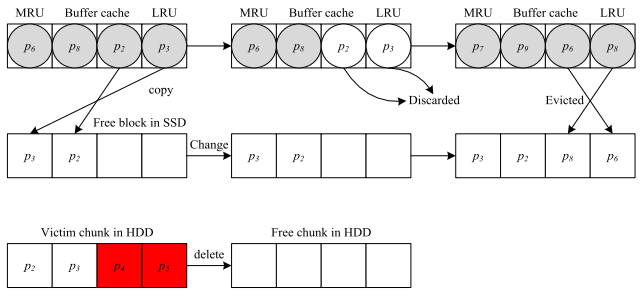


FIGURE 7. The process of buffer-aware data migration scheme.

SSD during the data migration process. However, these two pages are out-of-date and their newly updated versions are in the buffer cache. In this case, if the newly updated versions are chosen for eviction by the buffer management layer, they must be written into the SSD to achieve the data consistency. Hence, the future replacement cost, denoted by $T_{RE}^{BU}(C_i)$, for the buffer-unaware data migration scheme is $|\text{IO}\,(C_i)| * T_{b\to s}$.

During the buffer-aware data migration scheme, if there are obsolete pages in the victim chunk to be migrated, their newly updated versions in the buffer cache are written into the target device instead of obsolete pages and then these newly updated versions are marked as clean. Suppose that there do not exist further modifications that are performed on them before they are flushed from the buffer cache by the buffer management layer of Linux operating system. Then, they will be just discarded. Therefore, the future replace cost, expressed by $T_{RE}^{BA}(C_i)$, of the buffer-aware data migration scheme is 0. In this example, two write operations into SSD are saved. This benefit, which results from the reduction of future write operations, is referred to as a future replacement benefit. The future replacement benefit can be calculated as $T_{RE}^{BU}(C_i) - T_{RE}^{BA}(C_i)$, which is $2 * T_{b\to s}$ in Figures 6 and 7. Therefore, the number of write operations eliminated during the replacement process is $\left(T_{RE}^{BU}(C_i) - T_{RE}^{BA}(C_i)\right)/T_{b\to s}$.

SSD uses an out-of-place update scheme to write the new data into the free space instead of updating the original one directly, so write operations into SSD could use up its free space gradually and then the data migration operation must be performed to migrate a portion of data into the HDD for making free space for the data migration from HDD to SSD. As discussed above, the buffer-aware data migration scheme consumes less free pages than the buffer-unaware one since the buffer-aware one incurs less write operations into SSD. For the example in Figure 6, six pages are consumed during the buffer-unaware data migration scheme, while only four pages are consumed during the buffer-aware data migration scheme. Then, two free pages can be saved during the buffer-aware data migration scheme. Hence, the next data migration operation from the SSD to HDD can be delayed. This benefit that results from the decreasing of data migration operations is named a future data migration benefit.

In order to estimate the future data migration benefit, the cost for a data migration operation resulting from a single

write operation will be estimated first. A block in SSD may be selected for data migration when all its free pages have been used up. If a block has N_b pages, then a data migration operation will be incurred every N_b write operations into a block in the SSD. It means that if N_b write operations are reduced, one data migration operation can be eliminated. Let T_{fdm}^{BA} be the future data migration cost of the buffer-aware data migration scheme. In our paper, T_{fdm}^{BA} can be predicted using an exponentially weighted moving average (EWMA) method based on recent known data migration costs. Then the future data migration cost, which results from one write operation, could be estimated as $\left(T_{fdm}^{BA}/N_b\right)$. As mentioned above, the number of write operations decreased during the replacement process is $\left(T_{RE}^{BU}(C_i) - T_{RE}^{BA}(C_i)\right)/T_{b\to s}$. Hence, the future data migration benefit is $\left(T_{RE}^{BU}(C_i) - T_{RE}^{BA}(C_i)\right) * \alpha$, where the term α is denoted by $\left(T_{fdm}^{BA}/N_b\right)/T_{b\to s}$.

Based on the above discussion, the total potential benefits of the buffer-aware data migration can be defined as follows.

Definition 3: Let $T_{RE}^{BU}(C_i)$ be the future replacement cost for the buffer-unaware data migration scheme and $T_{RE}^{BA}(C_i)$ be the future replacement cost resulting from the buffer-aware data migration scheme. Hence, $T_{RE}^{BU}(C_i) - T_{RE}^{BA}(C_i)$ denotes the future replacement benefit and $\left(T_{RE}^{BU}(C_i) - T_{RE}^{BA}(C_i)\right)*\alpha$ is the future data migration benefit. Then, the total potential benefits for migrating a chunk C_i, expressed by $B_{benefit}^{BA}(C_i)$, are defined as

$$B_{benefit}^{BA}(C_i) = \left(T_{RE}^{BU}(C_i) - T_{RE}^{BA}(C_i)\right) * (1+\alpha) \quad (8)$$

Since $T_{RE}^{BU}(C_i) \geq T_{RE}^{BA}(C_i), B_{benefit}^{BA} \geq 0$. At the same time, $T_{dm}^{BA}(C_i) < T_{dm}^{BU}(C_i)$, as mentioned in subsection B. Hence, the buffer-aware data migration scheme performs better than the buffer-unaware data migration scheme theoretically.

However, if the pages, which have been written from the buffer cache into SSD during a data migration operation, are modified again before being flushed from the buffer cache, there will be no potential benefits since the newly updated versions in the buffer cache must be flushed into the SSD to keep the data consistency. Therefore, the potential benefits, expressed by $B_{benefit}^{BA}(C_i)$, depend on the update probability of each page within the IO (C_i) after the buffer-aware data migration. For the example in Figure 7, if the pages p_2 and p_3 are modified again before being flushed from the buffer cache, then no write operations would be decreased, namely $T_{RE}^{BU}(C_i) = T_{RE}^{BA}(C_i)$ since the newly updated versions must be written to SSD again to keep the data consistency. In this case, $B_{benefit}^{BA}(C_i)$ becomes 0. Hence, if all the pages in the IO (C_i) are updated one more time before they are evicted by the buffer management layer, then the potential benefits of the buffer-aware data migration scheme become zero.

VI. PERFORMANCE EVALUATION

A. EXPERIMENTAL SETUP

A prototype system for hybrid storage systems is designed and implemented to assess the performance of the proposed buffer-aware data migration scheme. It is equipped with one HDD and one SSD, which are installed in the same level of storage hierarchy. It contains one storage management layer that utilizes the data structure B+-tree so as to manage and operate the user data that are placed at the HDD and SSD, and a buffer management layer that uses the LRU algorithm to manage the temporary data that are placed in the buffer. This prototype system is developed using C++ and deployed on a commodity PC system, which is equipped with an Intel Pentium Dual Core 3.2GHz CPU and 3GB of the physical memory. The operating system used in the experiments is Fedora 14 with the Linux kernel 2.6.35.6 and the Ext3 file system is utilized in the experiment. The PC system has two HDDs and one SSD. The prototype system and operating system run on one of the HDDs. The other HDD and SSD are utilized to hold the user data. The HDD is a 7200 RPM, 500GB TOSHIBA and the SSD belongs to Intel SSD DC S3520 Series with 150GB. Both of them are linked and then communicated with the host system using SATA3 interface.

To test the effectiveness and verify the superiority of the proposed buffer-aware data migration scheme, this scheme is integrated into the MDAM [36], which is the representative HDD/SSD hybrid storage system.

B. BENCHMARKS

The experiments were conducted with multiple well-known benchmark programs. The first program is Postmark that is an industry-standard file system benchmark. It is developed to simulate I/O intensive and small-file-oriented workloads such as I/O operations of a large e-mail server or USENET news system. In our experiments, it is configured to create 1,000,000 small random files with their sizes between 500 bytes and 10,000 bytes, then perform 10,000,000 transactions to stress the file system, and finally delete these files. The second one is IOzone, which is a file system program that is designed to measure the streaming performance for the large files. It generates a number of updates to metadata and data. In our experiments, it is utilized to execute write/re-write and read/re-read operations on a 8 GB file. The R/W ratio of the workload that is created by IOzone is 3:1 and the workload follows the Zipf distribution. The last one is TPC-C, which is an on-line transaction processing benchmark. It is utilized to simulate a whole computing environment that a number of users perform online transactions against a database. In the experiments, the BenchmarkSQL software is used to run the TPC-C tests on the PostgreSQL.

For minimizing the effect on the performance of hybrid storage systems, the proposed buffer-aware data migration algorithm is designed to run background when running the benchmarks.

Figures 8, 9, and 10 present the normalized runtimes for various hybrid storage systems when running on three traces. It can be seen that the buffer-aware data migration scheme could reduce the runtime by at least 5%. Moreover, as the buffer cache size increases, the performance of the buffer-aware data migration scheme also becomes greater. That is because the buffer-aware data migration scheme

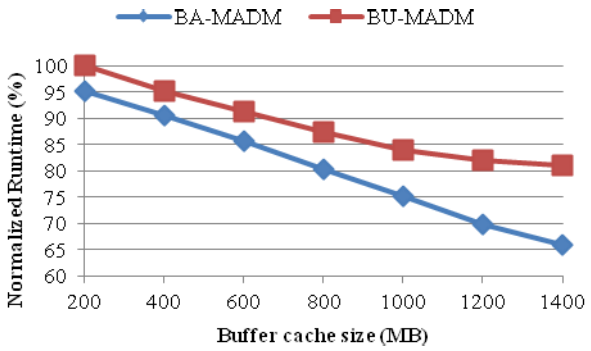


FIGURE 8. Normalized runtime for different MADM versions on Postmark.

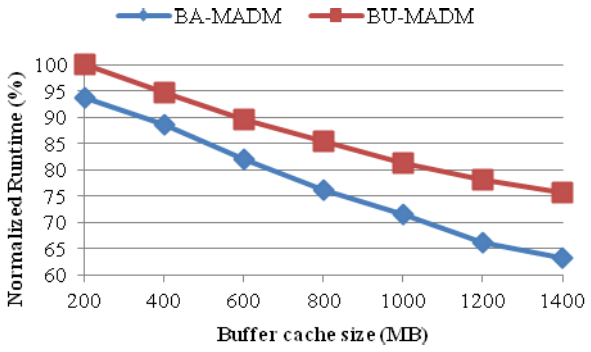


FIGURE 9. Normalized runtime for different MADM versions on IOzone.

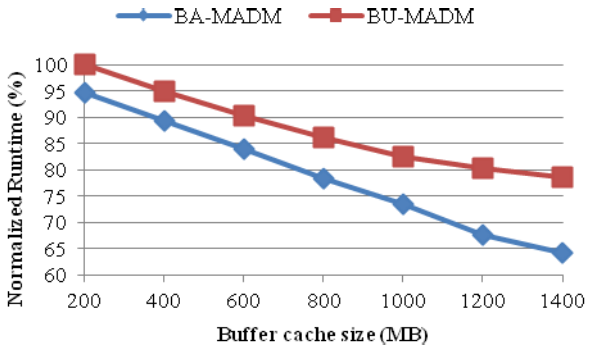


FIGURE 10. Normalized runtime for different MADM versions on TPC-C.

writes the newly updated versions in a buffer cache instead of the old data in the storage device to the other storage device. It can not only eliminate the unnecessary page migrations between HDD and SSD, but also decrease the time that is consumed to move the data placed at the HDD and SSD.

The reduction of page migrations from HDD to SSD could also delay the page migrations from SSD to HDD since the SSD should migrate the LRU pages into the HDD to make free spaces for the data, which will be moved from HDD.

Figures 11, 12, and 13 present the normalized write count for different MADM versions when running on benchmarks. It can be seen that the buffer-aware data migration scheme can decrease the write count by up to 14% over the buffer-unaware one. That is because it reduces the number of write operations during the buffer replacement process. It performs best under the IOzone benchmark because the

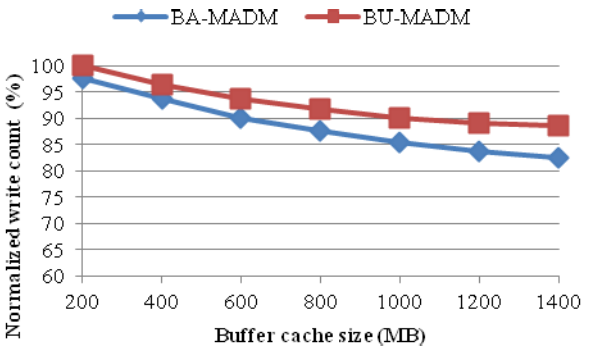


FIGURE 11. Normalized write count for different MADM versions on Postmark.

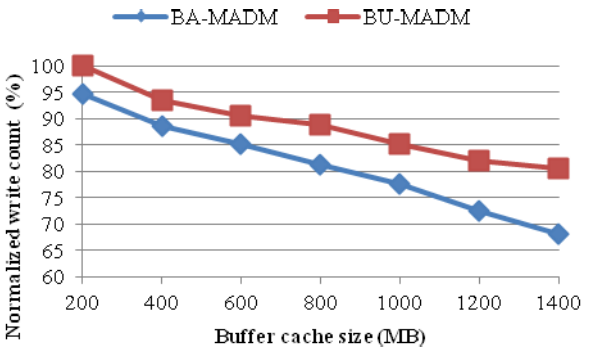


FIGURE 12. Normalized write count for different MADM versions on IOzone.

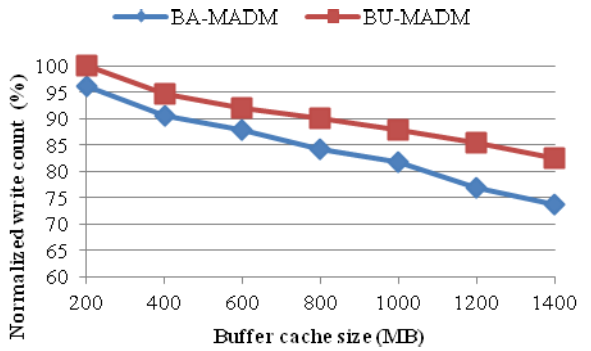


FIGURE 13. Normalized write count for different MADM versions on TPC-C.

IOzone is a write-intensive benchmark and the buffer cache owns many dirty pages.

C. REAL WORKLOAD

The proposed buffer-aware data migration scheme is further verified using a real workload provided by the Benchmarking Working Group. The real workload shows the feature that a large proportion of data access is distributed over many small files. At least 90% of data access is performed on the files the sizes of which are 4MB or less and the files owning the size of 2KB present the highest proportion. This real workload is a kind of indicative I/O operations in large-scale data centers.

Figures 14 and 15 report experimental results for different MADM versions when the real workload is replayed. Similar

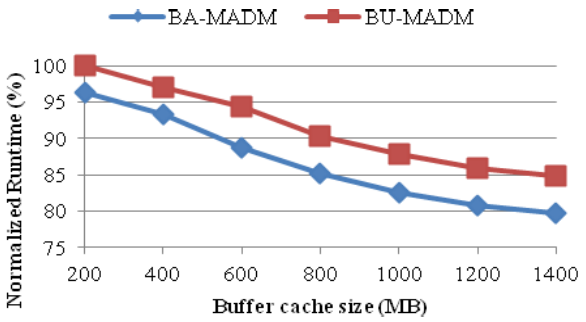


FIGURE 14. Normalized runtime for different MADM versions on real workload.

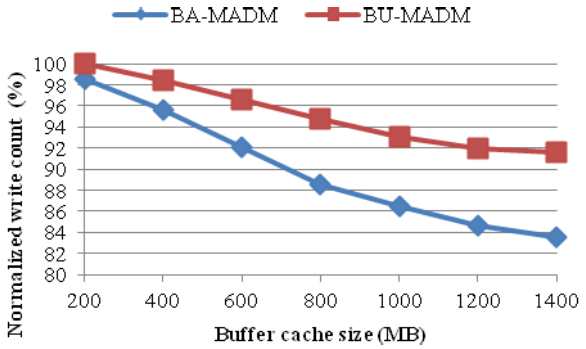


FIGURE 15. Normalized write count for different MADM versions on real workload.

to the previous results under benchmarks, the buffer-aware data migration scheme can reduce the runtime by up to 6.4% and the write count by up to 8.7% when it is compared to the buffer-unaware one.

VII. CONCLUSIONS

In this paper, a novel buffer-aware data migration scheme is put forward to exploit the content information of the buffer cache. We first show our target system architecture and then model it. Then we introduce two new states to reclassify the data stored in the hybrid storage systems, which consist of the HDD and SSD. The buffer-aware data migration scheme writes the newly updated data in the buffer instead of ones in the storage device to another one, which can not only reduce the number of page migration operations, but also improve the reliability of hybrid storage systems. Based on the new data states, a model is constructed to assess the performance advancement of the buffer-aware data migration scheme over the buffer-unaware data migration scheme. The buffer-aware data migration scheme has been theoretically proven by us to outperform the buffer-unaware data migration scheme that is commonly used in existing hybrid storage systems composed of the HDD and SSD. The experimental results also validate this fact.

In the future, we intend to exploit the proposed buffer-aware data migration scheme to boost the performance of hybrid disk arrays such as HPDA [41] and LDM [42].

REFERENCES

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

[2] J. Ryu, D. Lee, C. Han, H. Shin, and K. Kang, "File-system-level storage tiering for faster application launches on logical hybrid disks," *IEEE Access*, vol. 4, pp. 3688–3696, 2016.

[3] M. Lin, R. Chen, J. Xiong, X. Li, and Z. Yao, "Efficient sequential data migration scheme considering dying data for HDD/SSD hybrid storage systems," *IEEE Access*, vol. 5, pp. 23366–23373, 2017.

[4] L. Wu, Q. Zhuge, E. H.-M. Sha, X. Chen, and L. Cheng, "BOSS: An efficient data distribution strategy for object storage systems with hybrid devices," *IEEE Access*, vol. 5, pp. 23979–23993, 2017.

[5] B. Welch and G. Noer, "Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions," in *Proc. IEEE 29th Symp. Mass Storage Syst. Technol.*, May 2013, pp. 1–12.

[6] J. Shen, H. Tan, J. Wang, J. Wang, and S. Lee, "A novel routing protocol providing good transmission reliability in underwater sensor networks," *J. Internet Technol.*, vol. 16, no. 1, pp. 171–178, 2015.

[7] J. Hu, H. Jiang, L. Tian, and L. Xu, "PUD-LRU: An erase-efficient write buffer management algorithm for flash memory SSD," in *Proc. 18th Annu. IEEE/ACM Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst.*, Aug. 2010, pp. 69–78.

[8] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim, "A case for flash memory SSD in enterprise database applications," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 1075–1086.

[9] S.-H. Kang, D.-H. Koo, W.-H. Kang, and S.-W. Lee, "A case for flash memory SSD in Hadoop applications," *Int. J. Control Automat.*, vol. 6, no. 1, pp. 201–210, 2013.

[10] C.-H. Wu, D.-Y. Wu, H.-M. Chou, and C.-A. Cheng, "Rethink the design of flash translation layers in a component-based view," *IEEE Access*, vol. 5, pp. 12895–12912, 2017.

[11] X. Xie, L. Xiao, X. Ge, and Q. Li, "SMRC: An endurable SSD cache for host-aware shingled magnetic recording drives," *IEEE Access*, vol. 6, pp. 20916–20928, 2018.

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

[13] G. Xu, F. Lin, and Y. Xiao, "CLRU: A new page replacement algorithm for NAND flash-based consumer electronics," *IEEE Trans. Consum. Electron.*, vol. 60, no. 1, pp. 38–44, Feb. 2014.

[14] J. Liu, S. Chen, T. Wu, and H. Zhang, "A novel hot data identification mechanism for NAND flash memory," *IEEE Trans. Consum. Electron.*, vol. 61, no. 4, pp. 463–469, Nov. 2015.

[15] J. Liu, S. Chen, G. Wang, and T. Wu, "Page replacement algorithm based on counting Bloom filter for NAND flash memory," *IEEE Trans. Consum. Electron.*, vol. 60, no. 4, pp. 636–643, Nov. 2014.

[16] J. Handy, "Will the HDD market be swept by hybrid hard drives?" *Solid State Technol.*, vol. 50, no. 9, pp. S16–S17, 2007.

[17] Y. Liu, J. Huang, C. Xie, and Q. Cao, "RAF: A random access first cache management to improve SSD-based disk cache," in *Proc. IEEE Int. Conf. Netw., Archit. Storage*, 2010, pp. 492–500.

[18] P. Sehgal, K. Voruganti, and R. Sundaram, "SLO-aware hybrid store," in *Proc. IEEE 28th Symp. Mass Storage Syst. Technol.*, Apr. 2012, pp. 1–6.

[19] L. Lin, Y. Zhu, J. Yue, Z. Cai, and B. Segee, "Hot random off-loading: A hybrid storage system with dynamic data migration," in *Proc. 19th Annu. IEEE/ACM Int. Symp. Modeling, Anal., Simulation Comput. Telecommun. Syst.*, Jul. 2011, pp. 318–325.

[20] S. He, X.-H. Sun, and B. Feng, "S4D-Cache: Smart selective SSD cache for parallel I/O systems," in *Proc. Int. Conf. Distrib. Comput. Syst.*, Jun. 2014, pp. 514–523.

[21] C. Han, J. Ryu, D. Lee, J. Lee, K. Kang, and H. Shin, "File-system-level flash caching for improving application launch time on logical hybrid disks," in *Proc. IEEE 33rd Int. Perform. Comput. Commun. Conf.*, Dec. 2014, pp. 1–2.

[22] P. Yang, P. Jin, S. Wan, and L. Yue, "HB-Storage: Optimizing SSDs with a HDD write buffer," in *Proc. 14th Int. Conf. Web-Age Inf. Manage.*, Beidaihe, China: Springer-Verlag, 2013, pp. 28–39.

[23] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending SSD lifetimes with disk-based write caches," in *Proc. 8th USENIX Conf. File Storage Technol.*, 2010, p. 8.

[24] J. L. Jacobi, "Flash memory to speed up hard drives," *PC World*, vol. 23, no. 9, p. 20, 2005.

[25] J. Suk and J. No, "Performance analysis of NAND flash-based ssd for designing a hybrid filesystem," in *Proc. 11th IEEE Int. Conf. High Perform. Comput. Commun.*, Jun. 2009, pp. 539–544.
[26] J. Suk and J. No, "Hybrid file system," in *Proc. Int. Conf. Converg. Hybrid Inf. Technol.*, 2009, pp. 308–313.
[27] H. Jo, Y. Kwon, H. Kim, E. Seo, J. Lee, and S. Maeng, "SSD-HDD-hybrid virtual disk in consolidated environments," in *Euro-Par 2009 Parallel Processing Workshops*. Delft, Netherlands: Springer-Verlag, 2010, pp. 375–384.
[28] H.-T. Hsu and Y.-W. Bai, "Using NAND flash memory to improve the performance of HDDS," in *Proc. 23rd Can. Conf. Elect. Comput. Eng.*, May 2010, pp. 1–6.
[29] J. No, "Hybrid file system using NAND-flash SSD," in *Proc. Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discovery*, Oct. 2011, pp. 380–385.
[30] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "HybridStore: A cost-efficient, high-performance storage system combining SSDs and HDDs," in *Proc. 19th Annu. IEEE/ACM Int. Symp. Modeling, Anal., Simulation Comput. Telecommun. Syst.*, Jul. 2011, pp. 227–236.
[31] J. Hui, X. Ge, X. Huang, Y. Liu, and Q. Ran, "E-HASH: An energy-efficient hybrid storage system composed of one SSD and multiple HDDs," in *Proc. 3rd Int. Conf. Swarm Intell.* Shenzhen, China: Springer-Verlag, 2012, pp. 527–534.
[32] N. Fisher, Z. He, and M. McCarthy, "A hybrid filesystem for hard disk drives in tandem with flash memory," *Computing*, vol. 94, no. 1, pp. 21–68, 2012.
[33] W. Xiao, X. Lei, R. Li, N. Park, and D. J. Lilja, "PASS: A hybrid storage system for performance-synchronization tradeoffs using SSDs," in *Proc. 10th IEEE Int. Symp. Parallel Distrib. Process. Appl.*, Jul. 2012, pp. 403–410.
[34] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "HybridPlan: A capacity planning technique for projecting storage requirements in hybrid storage systems," *J. Supercomput.*, vol. 67, no. 1, pp. 277–303, 2014.
[35] C. Xu, W. Wang, D. Zhou, and T. Xie, "An SSD-HDD integrated storage architecture for write-once-read-once applications on clusters," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2015, pp. 74–77.
[36] X. Wu and A. L. N. Reddy, "Managing storage space in a flash and disk hybrid storage system," in *Proc. IEEE Int. Symp. Modeling, Anal., Simulation Comput. Telecommun. Syst.*, Sep. 2009, pp. 1–4.
[37] T. Xie and D. Madathil, "SAIL: Self-adaptive file reallocation on hybrid disk arrays," in *Proc. Int. Conf. High-Perform. Comput.*, 2008, pp. 529–540.
[38] T. Xie and Y. Sun, "Dynamic data reallocation in hybrid disk arrays," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 9, pp. 1330–1341, Dec. 2010.
[39] J. He, G. Jia, G. Han, H. Wang, and X. Yang, "Locality-aware replacement algorithm in flash memory to optimize cloud computing for smart factory of industry 4.0," *IEEE Access*, vol. 5, pp. 16252–16262, 2017.
[40] Y. Yuan, Y. Shen, W. Li, D. Yu, L. Yan, and Y. Wang, "PR-LRU: A novel buffer replacement algorithm based on the probability of reference for flash memory," *IEEE Access*, vol. 5, pp. 12626–12634, 2017.
[41] B. Mao *et al.*, "HPDA: A hybrid parity-based disk array for enhanced performance and reliability," *ACM Trans. Storage*, vol. 8, no. 1, 2012, Art. no. 4.
[42] S. Wu, B. Mao, X. Chen, and H. Jiang, "LDM: Log disk mirroring with improved performance and reliability for SSD-based disk arrays," *ACM Trans. Storage*, vol. 12, no. 4, 2016, Art. no. 22.
[43] S. He, Y. Wang, and X.-H. Sun, "Improving performance of parallel I/O systems through selective and layout-aware SSD cache," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2940–2952, Oct. 2016.
[44] S. He, X.-H. Sun, and A. Haider, "HAS: Heterogeneity-aware selective data layout scheme for parallel file systems on hybrid servers," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2015, pp. 613–622.
[45] S. He, Y. Wang, and X.-H. Sun, "Boosting parallel file system performance via heterogeneity-aware selective data layout," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2492–2505, Sep. 2016.
[46] S. He, X.-H. Sun, Y. Wang, A. Kougkas, and A. Haider, "A heterogeneity-aware region-level data layout for hybrid parallel file systems," in *Proc. 44th Int. Conf. Parallel Process.*, Sep. 2015, pp. 340–349.
[47] S. He, X.-H. Sun, Y. Wang, and C. Xu, "A migratory heterogeneity-aware data layout scheme for parallel file systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2018, pp. 1133–1142.

MINGWEI LIN received the B.S. degree in software engineering and the Ph.D. degree in computer science and technology from Chongqing University, Chongqing, China, in 2009 and 2014, respectively.

From 2015 to 2016, he was a Lecturer with the Faculty of Software, Fujian Normal University, Fuzhou, China, where he is currently an Associate Professor. He has published more than 20 research papers as first author in international journals and conference proceedings. His research interests include storage system and embedded system. He got the CSC-IBM Chinese Excellent Student Scholarship in 2012.

RIQING CHEN received the B.Eng. degree in communication engineering from Tongji University, Shanghai, China, in 2001, the M.Sc. degree in communications and signal processing from Imperial College London, U.K., in 2004, and the Ph.D. degree in engineering science from the University of Oxford, U.K., in 2010.

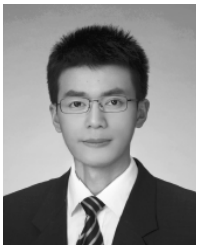
Since 2014, he has been a Full Professor with the Faculty of Computer and Information Sciences, Fujian Agriculture and Forestry University, Fuzhou, China. His research interests include big data and visualization, cloud computing, consumer electronics, flash memory, and wireless sensor networking.

LI LIN received the B.S. degree in computer science and technology and the M.S. degree in computer application from Sichuan University, Chengdu, China, in 2005 and 2008, respectively. He is currently pursuing the Ph.D. degree in computer system architecture at the Huazhong University of Science and Technology, Wuhan, China.

Since 2008, he has been with the Faculty of Software, Fujian Normal University, Fuzhou, China, where he is currently a Lecturer. His research interest is mobile computing.

XUAN LI received the B.S. degree in mathematics and applied mathematics and the Ph.D. degree in computer science and technology from the South China University of Technology, China, in 2007 and 2012, respectively.

From 2012 to 2014, she was a Lecturer with the Information Science School, Guangdong University of Finance and Economics, Guangzhou, China. Since 2015, she has been an Associate Professor with Fujian Normal University. Her research interests include consumer electronics and computer security.



JINGCHANG HUANG was born in Sanming, Fujian, China. He received the Ph.D. degree in electrical engineering from the University of Chinese Academic and Sciences, China, in 2015.

He mainly focus on the cognitive signal processing, including but not limited to acoustic, image, air quality data, all of which comes from the Internet-of-Things (IOT)-related applications. His current research interests also include patterns recognition, big data, and wireless sensor networks.