2014 IEEE 28th International Parallel & Distributed Processing Symposium

EDM: an Endurance-aware Data Migration Scheme for Load Balancing in SSD Storage Clusters

Jiaxin Ou, Jiwu Shu*, Youyou Lu, Letian Yi, Wei Wang
Department of Computer Science and Technology, Tsinghua University, Beijing, China
Tsinghua National Laboratory for Information Science and Technology, Beijing, China
**Corresponding author: shujw@tsinghua.edu.cn*
{*ojx11, luyy09, yilt09, wangwei11*}*@mails.tsinghua.edu.cn*

***Abstract*—Data migration schemes are critical to balance the load in storage clusters for performance improvement. However, as NAND flash based SSDs are widely deployed in storage systems, extending the lifespan of SSD storage clusters becomes a new challenge for data migration. Prior approaches designed for HDD storage clusters, however, are inefficient due to excessive write amplification during data migration, which significantly decrease the lifespan of SSD storage clusters.**

To overcome this problem, we propose EDM, an endurance-aware data migration scheme with careful data placement and movement to minimize the data migrated, so as to limit the worn-out of SSDs while improving the performance. Based on the observation that performance degradation is dominated by the wear speed of an SSD, which is affected by both the storage utilization and the write intensity, two complementary data migration policies are designed to explore the trade-offs among throughput, response time during migration, and lifetime of SSD storage clusters. Moreover, we design an SSD wear model and quantitatively calculate the amount of data migrated as well as the sources and destinations of the migration, so as to reduce the write amplification caused by migration. Results on a real storage cluster using real-world traces show that EDM performs favorably versus existing HDD based migration techniques, reducing cluster-wide aggregate erase count by up to 40%. In the meantime, it improves the performance by 25% on average compared to the baseline system which achieves almost the same effectiveness of performance improvement as previous migration techniques.

***Keywords*-Storage Cluster, Data Migration, Load Balancing, Solid State Drive (SSD), Endurance**

I. Introduction

Data intensive applications have been increasingly popular in high-performance computing systems in recent years, and the data volume has dramatically increased to the peta-scale. I/O performance becomes the critical part of the parallel computing. Parallel file systems, e.g., PVFS2 [5], Lustre [3] and Ceph [23], achieve high aggregate I/O throughput by leveraging the parallelism of multiple storage devices in a shared storage cluster. In parallel file systems, hash-based data placement scheme is commonly used to uniformly distribute data among all available storage devices. While this scheme results in a uniform data placement, the I/O load may be imbalanced across different storage servers because of the non-uniform access distribution in the workloads [14]. This makes heavily loaded servers become the performance bottleneck of the cluster while others are still under-utilized.

Load balancing on storage systems with hard disk drives (HDDs) has been extensively studied [9], [20]. Frequently accessed data (a.k.a. hot data) are moved from the heavily loaded servers to the lightly loaded ones when the cluster suffers from unbalanced load. However, these approaches may not work well in a storage cluster with NAND flash solid state drives (SSDs), which have been deployed in many storage systems. NAND-flash SSDs have an advantage over HDDs in I/O performance and energy consumption. But they have limited lifespan [16], which is a critical design issue in SSD storage systems. Unfortunately, existing approaches for HDDs have not taken the lifetime issue into account, and the data migration can even make them wear faster due to extra data writes.

First, read and write operations are not differentiated in existing schemes because both types of operations impose similar load on the HDDs. That is, the workload is measured simply as the total number of accesses, either reads or writes. Such a simplification, however, is not appropriate for SSDs because of asymmetry of read and write operations. Since the service time of a write operation is significantly longer than that of a read one, whether to move hot write data or hot read data can have a substantial impact on the efficiency of data migration. Moreover, the garbage collection process (GC), which is quite time consuming, is also influenced by the write intensity. In SSDs, reads and writes are performed in flash page units (e.g., 4KB), while the erase operations are performed in flash block units (e.g., 512KB). SSDs update pages by writing the data to free pages and invalidating the old ones (a.k.a. out-of-place update) in a log-structured way. Garbage collection is used in SSDs to reclaim the invalid pages by moving valid pages to new blocks and then erasing the old blocks. However, this process leads to high latency and has a significant impact on storage performance. Consequently, the write operations, which influence the garbage collection frequency, have larger impact on the SSD's load than read operations. Therefore, migrating hot written data is more effective than migrating hot read ones to achieve the same effectiveness of load

1530-2075/14 $31.00 © 2014 IEEE
DOI 10.1109/IPDPS.2014.86

balancing, as less data would be moved.

Second, a new model should be built for SSD storage cluster to carefully calculate the amount of data movement so as to minimize the write amplification caused by migration. Since garbage collection process is quite time consuming, and can block the normal I/O operations during its process, it is likely to be the main reason that influences the SSD's performance. Accordingly, balancing the wear across SSDs can help balance the storage load and improve the system performance. Consequently, a new model needs to be built to calculate the amount of data movement according to the frequency of garbage collection. Moreover, the frequency of garbage collection is influenced by both the write intensity and storage utilization. On one hand, because garbage collection is triggered when the free space is not enough, more writes cause higher frequency of garbage collection and thus worse storage performance. On the other hand, since garbage collection is affected by the size of the over-provisioned space in an SSD, the frequency of garbage collection is also determined by the storage utilization of the SSD [18]. In summary, both write intensity and storage utilization need to be considered in this new model.

In order to address these problems, we propose EDM, an endurance-aware data migration scheme with carefully arranged data placement and migration in order to reduce the wear on SSDs caused by data migration, while improving the performance. Since garbage collection frequency is influenced by the storage utilization and write intensity, we introduce two complementary migration policies to explore the trade-offs among throughput, response time during migration, and lifetime of SSDs in a storage cluster. The contributions of this paper can be summarized as follows.

- We reveal the problem of wear-variance among different devices in an SSD storage cluster through measuring erasure count of each SSD in a real system. To the best of our knowledge, this is the first work to address this problem and study its impact on system performance degradation.
- Based on the observation that performance degradation is dominated by the wear speed of an SSD, which is affected by both the storage utilization and the write intensity, two data migration policies are designed to explore the trade-offs among throughput, response time during migration, and lifetime of SSD storage cluster. To minimize the write amplification caused by migration, we propose HDF (Hot Data First) migration policy which selects the most *write-frequently* objects to be moved. However, it has a significant impact on response time of normal data access during migration. Therefore, another migration policy, which is called CDF (Cold Data First), is proposed to alleviate this impact by migrating the *rarely-accessed* objects which slightly relaxes the amount of data movement.
- To reduce the write amplification produced by data migration, we propose a new model to quantitatively calculate the amount of data movement according to the frequency of garbage collection.
- We implement the EDM scheme in a real storage cluster system based on pNFS. Results on real-world workloads show that EDM achieves almost the same effectiveness of performance improvement as existing HDD based migration techniques. In the meantime, it can reduce up to 40% aggregate erase operations compared to conventional techniques.

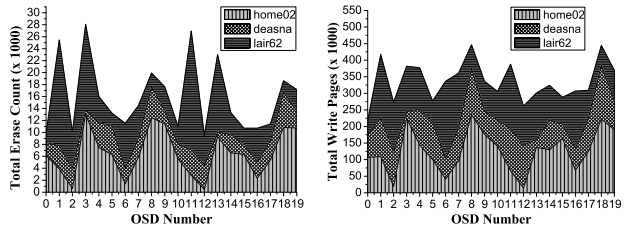
The rest of our paper is organized as follows. In Section II, we analyze the issue of wear variance across different devices in an SSD storage cluster. We describe the EDM scheme and present HDF and CDF migration policies in Section III. The implementation of EDM is introduced in Section IV, and the evaluation is provided in Section V. Section VI discusses the related work and the conclusions are given in Section VII.

II. Motivation: Wear Variance across SSDs

In an SSD, since garbage collection has long latency and can interfere with the service of normal I/O requests, the high wear frequency of an SSD is more likely to be the main reason that degrades its performance. Therefore, balancing the wear among SSDs can help balance the load across them, thus improving the system performance. In fact, the wear variance is affected by both the uneven storage utilization and the non-uniform write intensities across different SSDs.

First, wear variance can be caused by unbalanced storage utilization across SSDs. Higher disk utilization leads to more live pages in victim blocks that are selected for erasure [18]. This decreases the garbage collection performance as more live pages should be moved to other places before erasing these blocks. Unfortunately, storage utilization is uneven in existing distributed file systems [3], [23], [25] for the following two reasons: first, the heavily skewed object size distribution makes a uniform random object placement performs poorly, as indicated by high unevenness ratio of storage utilization; second, perfect balance of the number of objects is hard to achieve using randomized hash functions, which provide only probabilistically uniform distribution.

Second, even with similar storage utilization, wear variance can also exist due to non-uniform write intensities to different SSDs. In SSDs, each cell has limited program/erase (P/E) cycles, and the reliability of the cell degrades as the P/E cycles approach the limit. Moreover, P/E cycles are related to the write intensity which increase when more writes are issued to the SSD. However, data accesses have locality, and the writes are distributed to different locations non-uniformly [16]. For example, a large body of the writes might go to a small part of the data set. In SSD storage clusters, different SSDs have non-uniform write intensities thereby resulting in wear variance.



(a) Erase Count of Different SSDs (b) Write Pages of Different SSDs

Figure 1. Erase Count and Write Pages on Different SSDs.

To demonstrate the existence of wear variance in an SSD storage cluster, we use three real-world workloads, including *home02*, *deasna*, and *lair62* (see Section V.A for more details), and replay them on a baseline storage cluster system, which will be described in Section IV. In our experiments, we measured the total block erasure count and write pages of each SSD during the trace-replaying process. The results are shown in Figure 1. In Figure 1(a), we can see that the total block erasure count among all SSDs varies widely, especially for trace *home02* and *lair62*. In Figure 1(b), we can observe that an OSD (Object-based Storage Device) with more block erasure count tends to have more pages written into it in most cases, but not exclusively. For example, although OSD_2 and OSD_3 in trace *lair62* have roughly the same number of pages written into them, their block erasure counts are quite different. The revealed heavily skewed wear distribution strongly suggests that we need to balance the wear speeds across SSDs so as to improve the performance.

III. The EDM Scheme

In this section, we first introduce the hash-based object placement and intra-group data migration in EDM. Before describing the data migration policies, we present the SSD wear model and discuss how to estimate the object *temperature*. Then, we introduce two migration policies, HDF and CDF, respectively. Finally, we discuss how to manage the moved objects in the cluster and the reliability issue caused by data migration in SSD storage clusters.

A. Hash-based Object Placement and Intra-Group Data Migration

In this paper, we propose the design of EDM scheme based on the emerging object-based storage systems. In such systems [3], [23], [25], each file or directory consists of a relatively small number of logic data units with variable size, named objects, which are distributed throughout the storage clusters.

In EDM, we allocate k objects for each file. The k objects are placed onto k continuous SSDs, and the SSD to locate the first object of a file is selected as $inode\ number\ mod\ n$, where n is the total number of SSDs in the cluster. In this way, all objects can be almost uniformly distributed to the SSDs. As SSDs are facing with endurance problem, file data are striped over its k objects using object-level RAID-5 algorithm, which is more cost-effective than replication for SSD storage cluster to protect data from loss.

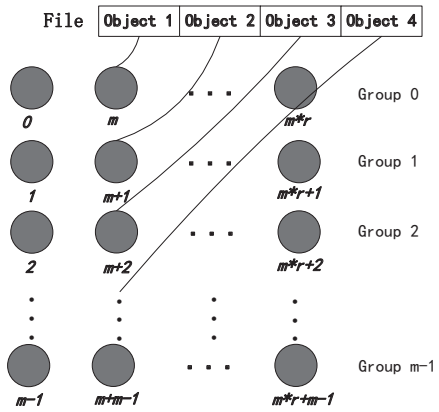


Figure 2. Hash-based Object Placement and Intra-Group Data Migration. Each gray cycle indicates an SSD in the cluster. The number below the cycle represents the number of each SSD in the cluster.

Moreover, we divide all the SSDs into several groups and guarantee that any two objects of a file are distributed to different groups (as shown in Figure 2). Assume that the total group number is m and each group consists of $r+1$ SSDs, then in Figure 2, we can see that $Group_i$ consists of ssd_i, $ssd_{(m+i)}$, ... , $ssd_{(m*r+i)}$. The data migration process in EDM is object-orient, and it only occurs among SSDs within one group. This intra-group migration can help resolve the reliability problem caused by migration in SSD storage cluster (see details in Section III.D).

B. Data Migration Scheme

1) The Wear Model of an SSD: In SSDs, the out-of-place update property indicates that garbage collection process needs to be triggered to reclaim free pages after a number of page writes have been processed. Each GC process selects one or more blocks to clean which are called the *victim blocks*. Moreover, the well-known greedy reclaiming policy [6] has been used for garbage collection in this paper. In such policy, the GC process first selects the block with the least number of valid pages as the victim block, then all valid pages in that block are copied to another block with free pages and the victim block is erased subsequently.

Assume that each block in flash memory has N_p pages, and the average ratio of valid pages in victim block is u_r. Thus, each victim block has $u_r * N_p$ valid pages averagely. Then, the GC process first relocates the $u_r * N_p$ valid pages from the victim block to another block with free pages, and then erases the victim block. It means that each GC operation generates N_p free pages while consuming another $u_r * N_p$ free pages at the same time. As a result, the GC operation

produces only $N_p * (1 - u_r)$ free pages actually. In other words, an SSD has to erase one block for every $N_p * (1 - u_r)$ page writes averagely in the steady state. Assume that the total number of page writes to an SSD during a certain period is W_c, then the block erasure count during this period E_c can be calculated as follows:

$$E_c = \frac{W_c}{N_p * (1 - u_r)} \quad (1)$$

However, the higher level of a storage system like file systems are typically unaware of the average utilization of the victim block. That is to say, u_r is not visible to upper applications. Fortunately, previous works [13], [15], [17], [22] have found that u_r has a great relationship with the real utilization of an SSD, denoted as u. And the following relation has already been used in these studies.

$$u = \frac{u_r - 1}{\ln u_r} \quad (2)$$

To confirm that whether Equation (2) is really suitable for real-world workloads, we measured u_r in our trace-based simulation experiments which will be discussed later. In the experiments, the measured u_r is compared with the estimated u_r obtained from Equation (2) in four different traces, including three real-world workloads (i.e., *home02*, *deasna*, and *lair62*) and one synthetic workload (i.e., *random*). The synthetic I/O request generator creates a random accessing workload, and each request size is ranging from 4KB to 16KB which is generated randomly. The results are shown in Figure 3. In Figure 3, we can observe that although the measured u_r in trace *random* matches well with the estimated value of Eq.(2), the results show great difference between them in the rest three realistic workloads. The reason is that lots of real-world workloads have access locality, either temporal locality or spatial locality. Moreover, most page writes may go to a relatively small portion of the objects due to the high skewness in I/O workloads [16]. Consequently, the cold objects tend to be separated in different blocks from hot objects over time, thus leading to fewer valid pages in a victim block than uniformly random accessing workload.

In order to get a more precise estimated value of u_r, we try to make minor modification to Equation (2). After adding an impact factor σ to Equation (2), we can derive a new Equation (3) as follows:

$$u = \frac{u_r - 1}{\ln u_r} + \sigma \quad (3)$$

In Figure 3, we can observe that the "estimated u_r of Eq.(3)-EDM" curve matches well with the three real-world workloads at least when the disk utilization is lower than 85% if we set σ to 0.28 empirically. Hence, we use this equation in our experiments. Let F(u) be the function which translates u to u_r (i.e., $u_r = F(u)$), then we can derive the

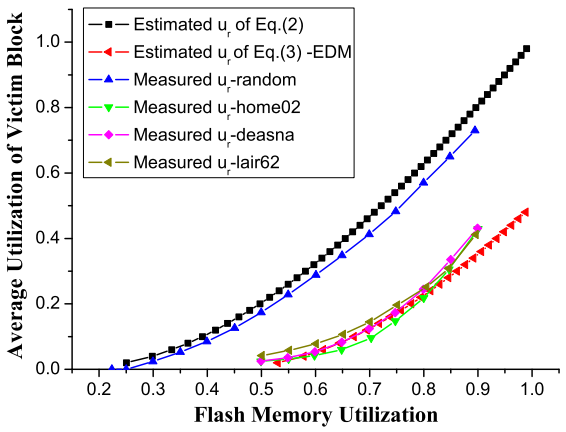


Figure 3. The Measured and Estimated Values of u_r and Its Relation with u in Different Traces

wear model of an SSD by combining the Equation (1) and Equation (3) leading to Equation (4) :

$$E_c(W_c, u) = \frac{W_c}{N_p * (1 - F(u))} \quad (4)$$

2) Data Migration Trigger Condition: Data migration trigger condition is the time when we should trigger the process of data migration. In EDM, data migration is desirable when there is significant wear imbalance. To this end, we calculate the block erasure count on each SSD according to Eq.(4) every minute. Significant wear imbalance means that $\sigma_e/\bar{E_c}$ (Relative Standard Deviation) $> \lambda$, where σ_e reflects the standard deviation and $\bar{E_c}$ is the cluster-wide average. The threshold λ can be adjusted in real cases. If device i satisfies $E_{c_i} - \bar{E_c} > \bar{E_c} * \lambda$, where E_{c_i} reflects the erasure count, we denote it as a source device for data migration. In contrast, all devices whose erasure count is lower than cluster-wide average belong to the destination set of migration.

3) Object Temperature: In EDM, each object has its own *temperature*, which indicates how likely the object is to be accessed in the near future. To estimate this metric, both frequency and temporal locality are considered in EDM. First, objects should be prioritized based on their access frequencies. That is, frequently accessed objects have a high probability of being accessed recently. Moreover, as the temporal locality really exists in most I/O workloads, recently accessed objects tend to be accessed quickly. On the contrary, objects that were accessed frequently in the past, but have not been accessed for a relatively long time should be a cold object. In addition, previous work [21] has shown that the temperature of a page can be estimated by weighting the page access over the time-line using an exponential decay function. Thus, we define the *object temperature* as follows:

Definition 1: OBJECT TEMPERATURE

Assume that we split the time-line which is from the time when the object is created to present into k equivalent continuous interval, and it contains k+1 discrete time points, which are t_0, t_1, t_2 ... t_k, respectively. Moreover, the total number of accessing the object in interval $[t_{i-1}, t_i]$ is A_i. Thus, the object temperature at time point t_k is defined as follows:

$$T_k(O) = \sum_{i=1}^{k} \frac{A_i}{2^{k-i}} \tag{5}$$

In fact, we do not need to keep the detailed history of read and write accesses by time in the past for each object, $T_k(O)$ can be derived from $T_{k-1}(O)$ using the following recurrence formula:

$$T_k(O) = \frac{T_{k-1}(O)}{2} + A_k \tag{6}$$

4) Migration Policy (Hot-Data First vs Cold-Data First): We propose two complementary migration policies which are called *Hot-Data First (HDF)* and *Cold-Data First (CDF)*, respectively, with different data selections for migration.

Hot-Data First (HDF): As discussed in Section II, the block erasure count of an SSD largely depends on the total pages written into it. From Eq. (4), we can observe that the less data written into an SSD, the fewer block erasure count an SSD has. Intuitively, HDF rebalances the wear by moving some most *write-frequently* objects out from the hot devices to the cold ones.

Cold-Data First (CDF): Though HDF can rebalance the wear by rearranging the location of the write-frequently objects across devices, it has a significant impact on normal file access during migration process. To overcome this problem, we propose *Cold-Data First (CDF)*, which makes new trade-offs among throughput, response time during migration, and lifetime of SSDs. In CDF, cold data are defined as the *rarely-accessed* objects. According to Section II, we note that the block erasure count of an SSD is also related to the disk utilization, and a hot SSD can be cooled down by reducing its disk utilization. To this end, CDF prefers to move the cold objects out from a hot SSD so as to alleviate its disk utilization thereby reducing the wear speed of it. Migrating cold objects has minimum impact on normal file access during migration, but causes a few more objects being migrated than HDF, because utilization can have smaller impact on wear speed compared to write intensity.

5) Object Selection for Migration: In EDM, all the actions of data movement are carefully decided before starting the migration process. Each data movement action is indicated by a triple, denoted as $(oid, source\_id, dest\_id)$, which means that object with its number equal to oid needs to be migrated from server $source\_id$ to server $dest\_id$.

For data selection, EDM first calculates the total amount of data movement needed for each target device, including the sources and destinations. The process of calculation is shown in Algorithm 1, we devise an iterative algorithm to achieve this objective. The rationale behind this algorithm is that: A well-balanced wear can be achieved by iteratively balancing the wear between the SSD with maximum wear-frequency and another one with minimum wear-frequency. Consequently, we can obtain an approximately optimal solution with certain iterative steps. In Algorithm 1, the impact of migration on disk utilization is ignored for HDF (i.e. array u is considered to be kept unchanged for HDF in Algorithm 1), because the skewness in I/O workloads results in that only a relatively small portion of the total objects should be moved, thereby the migration can have small impact on disk utilization. In Algorithm 1, array ΔW_c returned in the end represents the total amount of write pages that should be reduced or increased on each device. If $\Delta W_{c_i} < 0$, then $\Delta W_{c_i} * (-1)$ page write operations should be shifted from device i to others. Otherwise, device i is regarded as a destination that can accommodate ΔW_{c_i} page write operations.

Algorithm 1 :Calculate the Amount of Data Movement on Each Source or Destination Device

▷ Calculate the amount of data movement iteratively
▷ W_{c_i} is the total write pages on device i
▷ u_i is the average disk utilization of device i
▷ In our experiments, $total\_iteration\_step$ is set to 500
▷ It is for HDF, the one for CDF is similar to it

CALCULATE-AMOUNT-OF-DATA-MOVEMENT(W_c,u)

```
1  while i < total_iteration_step
2      do calculate the erase count of each source or
         destination device ▷ Call Eq.(4)
3         extract the device x with the max erase cnt
4         extract the device y with the min erase cnt
5         for ε ← 0; ε < 1; ε ← ε + 0.001
6             do Δw_pgs ← W_cx * ε
7                Δe ← E_c(W_cx − Δw_pgs, u_x) -
                 E_c(W_cy + Δw_pgs, u_y) ▷ Call Eq.(4)
8                if Δe ≤ 0
9                    do break ;
10        ΔW_cx ← ΔW_cx − Δw_pgs
11        ΔW_cy ← ΔW_cy + Δw_pgs
12        W_cx ← W_cx − Δw_pgs
13        W_cy ← W_cy + Δw_pgs ▷ update the total write
          pages of device x and device y
14        i ← i + 1
15 return ΔW_c
```

With the above information of array ΔW_c, EDM can decide what objects and how many objects should be moved away from the source devices under the help of the object temperature information. When estimating the object temperature, A_i is the write frequency of an object

(not including the read operations) for HDF in Equation (5), because we want to move the most *write-frequently* objects for HDF. In contrast, A_i represents the total access frequency (including both read and write operations) for CDF to calculate the temperature, as the *rarely-accessed* objects are first selected for migration. With the temperature of each object, HDF selects the objects from the highest temperature to the lowest one until the selected objects are sufficient to reduce the total write pages by $\Delta W_{c_i} * (-1)$ on each source device i, and then all these objects are relocated to the destination devices in proportion to ΔW_c. To avoid disk saturation, we guarantee that the free space in each destination device does not exceed a predefined threshold during data migration.

In CDF, we assume that array Δu represents the relative change in utilization needed for each device during migration. Moreover, we ignore the impact of migration on total write pages of the target devices when calculating array Δu (i.e. array W_c is considered to be kept unchanged for CDF in Algorithm 1), because the objects selected for migration are *rarely accessed*. Thus, the calculation process of array Δu is similar to Algorithm 1, which calculates array ΔW_c for HDF.

The data selection policy for migration in CDF is different from that in HDF. In CDF, target objects which meet $T_k(O)$ less than a threshold are first extracted from a source device. Then all these objects are regarded as cold objects, and sorted by their sizes. Afterwards, objects with the largest size are first selected for migration, because we want to minimize the total number of moved objects so as to slow down the growth rate of the size of the remapping table (see Section III.C). If the disk utilization of the source device is less than 50%, the migration of cold data is not helpful, because further reduction of the disk utilization has almost no effect on the wear frequency in such scenario (see Figure 3). Consequently, we never migrate a cold object from a source device whose disk utilization is less than 50 percent to another one in CDF policy.

Through the above steps, we can derive all the triples which indicate the data movement actions. Then we send these information to the sources and perform all the migration processes in parallel.

C. Remapping Management

In EDM, since hash-based data placement policy is employed for the cluster, we prefer to use a remapping table to manage the moved objects. As the size of the remapping table is directly related to the total number of moved objects, it increases with the number of data migration, thus consuming a huge memory for storing it over time. In order to slow down the growth rate of the remapping table, EDM prefers to select the objects which have corresponding entries in the remapping table for migration, because moving those objects would only need an update operation of the remapping table and does not increase the size of it. For example, HDF can select the hottest written objects among objects related in the remapping table for migration.

D. Reliability Discussion

Balancing wear among all SSDs in a storage cluster increases the probability of multiple disk failures due to simultaneous SSD worn-out, thus causing reliability issue. Diff-RAID [2] tackles this problem by distributing different ratios of writes to each SSD, but causes imbalanced load across different SSDs and degrades system performance significantly.

In EDM, we resolve this problem by differentiating the number of SSDs assigned to each group and conducting intra-group migration policy. As described in Section III.A, each file is organized in object-level RAID-5 to protect data from loss, and all its objects are striped across groups. Accordingly, any two objects of the same group belong to different files and never compose the RAID-5, thereby simultaneous failures arisen from SSDs within one group do not influence the reliability of the system as long as they never occur among SSDs of different groups. Moreover, as RAID-5 is composed of SSDs across different groups, any two groups have almost the same total wear. As a result, differentiating the number of SSDs assigned to each group can result in SSDs belong to different groups having different wear speeds, thereby avoiding simultaneous worn-out occurs among SSDs across different groups, thus resolving the reliability problem.

IV. Implementation

We implement EDM in a *baseline system* which is based on pNFS. The baseline system consists of three sub-parts: the Clients, the MDS (Metadata Server), and the OSDs (Object-based Storage Devices). In clients, we use the protocols of pNFS which is implemented as a part of the NFS v4.1 in Linux Kernel. Additionally, the object layout is selected for pNFS storage. In MDS, we modified the FSAL layer of nfs-ganesha-1.4.0, which is a NFS server running in user space, to let the pNFS export object layout from it. In OSDs, the osc-osd-1.7.0 is used to receive the I/O requests from both clients and mds, and then handle them serially. As to EDM, both EDM-HDF and EDM-CDF are implemented in nfs-ganesha-1.4.0 and osc-osd-1.7.0.

The architecture of EDM is illustrated in Figure 4. As we can see, EDM mainly contains four independent modules. The *remapping table manager* is responsible for managing the location of the moved objects. The *wear monitor* decides when the migration needs to be triggered. The *access tracker* updates the temperature of an accessed object whenever the OSD receives a write or read request. To reduce memory consumption, we cache only part of the objects' metadata in memory, for example, we only cache the k hottest objects

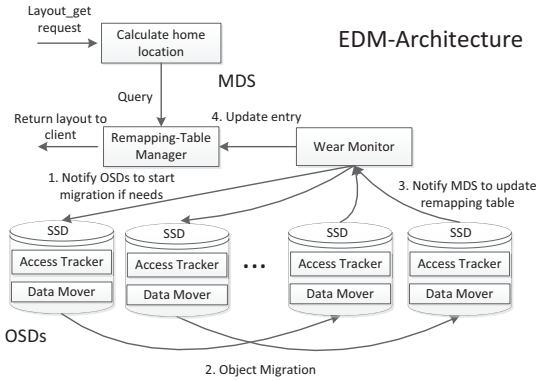


Figure 4. EDM Architecture

in memory for HDF. Lastly, the *data mover* is responsible for shuffling the objects across OSDs using multi-threads.

V. Evaluation

In this section, we evaluate EDM by comparing the two migration policies with the baseline system which has no migration scheme, and a conventional migration technique which is based on *Sorrento* [20]. In the baseline system, we select a hash-based data distribution policy to place the objects of each file, just as Section III.A describes. Additionally, the object number in the cluster is allocated continuously. To compare EDM with conventional data migration policies, we implement a conventional migration technique, which is called CMT, based on *Sorrento* in the baseline system. Different from *Sorrento*, CMT measures the load factor of an SSD by EMWA of the I/O latency, rather than I/O wait percentage, to facilitate our implementation.

A. Experimental Setup

All experiments were performed on a cluster up to 21 nodes, including a metadata server (MDS) and 20 storage nodes (OSD), and the number of load-generating clients is half of the number of OSDs. Moreover, each storage node in the cluster is equipped with only one SSD, and we enforce the OSDs to shuffle objects in the middle time point of trace replay. Since the object placement and migration policy is group-based as Section III.A describes, we set the group number m to 4, and the group size is set by dividing the number of OSDs by m in the experiments. Moreover, each file has 4 objects. Each node was configured with Quad-Core AMD Opteron(tm) Processor 2378 running at 2.40 GHz and 16GB physical memory. All nodes were running Linux 3.2.9 kernel (SUSE Linux 11 SP1).

In this paper, the experiments are conducted using seven traces collected from the network storage servers in Harvard University [8]. The characteristics of the seven workloads are shown in Table 1. For each trace, we extract the write, read, open and close operations from the NFS trace file and replay them on the clients of pNFS. Moreover, we implemented a multi-thread trace replaying tool. All trace records of multiple users are evenly assigned to each client in our experiments. Before replaying each trace, all files related in the trace file are pre-created and populated with sufficient data.

To accurately simulate the trace replaying process and measure the block erasure count, we also implement a flash simulator for EDM by modifying the flashsim simulator [12]. Moreover, to measure the overall performance of the storage cluster, our simulator chooses to store the real data on ramdisk and set a certain delay before completing each I/O operation to emulate the real SSD's I/O delay. Also, we set the delay of reading a page to 25 μs, and 200 μs to write a page, 2 ms to perform a block erase operation. Moreover, the OSDs are run on an FTL simulator which we implemented with the page-level FTL scheme [11]. In FTL scheme, we configure the SSD with 4KB page and 128KB block. To skip the *cold-start* and get the erasure count of each SSD in the steady-state, dummy data equal to the SSD's capacity are first written into each SSD. To emulate the disk utilization in realistic scenarios, the capacity of each SSD is set to the same dynamically before runing each trace-replaying program, which allows the maximum utilization among all SSDs is about 70 percent.

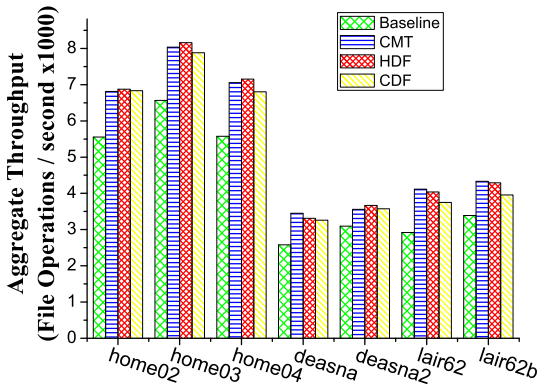
Table I
Characteristics of the Workloads

workload	file cnt	write cnt	average write size (byte)	read cnt	average read size (byte)
home02	10,931	730,602	8,048	3,497,486	8,191
home03	8,010	355,091	7,938	2,624,676	8,190
home04	7,798	358,976	8,013	2,034,078	8,192
deasna	9,727	232,481	24,167	271,619	23,869
deasna2	8,405	269,936	18,489	372,750	20,529
lair62	19,088	740,831	5,415	890,680	7,264
lair62b	27,228	409,215	5,496	736,469	7,612

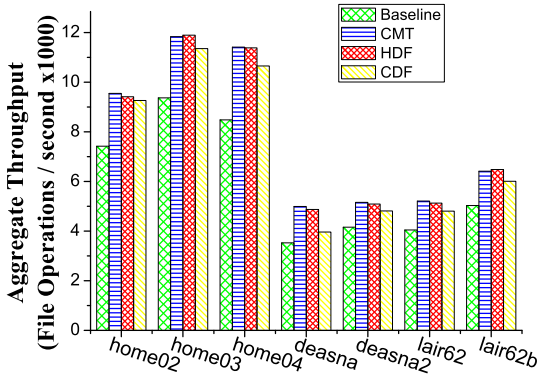
B. Performance Comparison

To evaluate the system performance, we calculate the aggregate throughput in terms of the number of completed file operations. The results are shown in Figure 5. Due to space limitation, we only show results in case of 16 OSDs and 20 OSDs. As we can see in this figure, the aggregate throughput when replaying the *home* traces is relatively higher than that when replaying others. This is due to that the *home* traces have higher read ratio than others.

Comparing EDM-HDF and EDM-CDF with the baseline system, we can see that the throughput increases by 15%-40% either using HDF or CDF policy. In the worst case, we observe that the throughput only improves by about 15% with CDF in Figure 5(b) for trace *deasna*. This is mainly due to that the wear variance in this case is already very

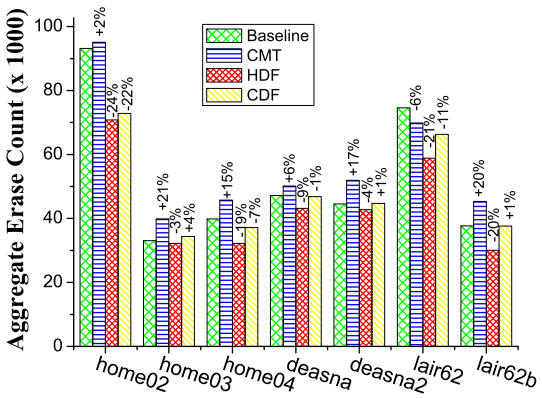


(a) 16-OSDs

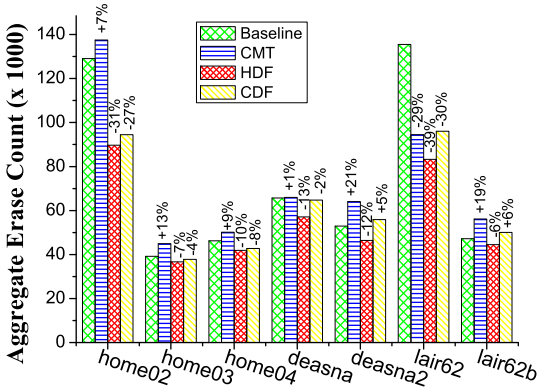


(b) 20-OSDs

Figure 5. Aggregate Throughput with Different Traces



(a) 16-OSDs



(b) 20-OSDs

Figure 6. Aggregate Erase Count among All OSDs. The numbers above the bars indicate respective difference with the baseline system from the view of aggregate erasure count.

small compared to that in other cases, and the oscillation is prone to appear when we force the migration in such case.

In Figure 5, we also observe that HDF and CMT achieve almost the same effectiveness of boosting the system performance, while both of them have a little higher aggregate throughput over CDF in most cases. We explain it as follows: EDM-CDF balances the wear across SSDs by reducing the utilization of the hot servers. Since the utilization has negligible impact on block erasure count when it is less than 50% (see Figure 3), the effectiveness of CDF is weakened if the utilization of some hot servers are less than 50%. In contrast, the total write pages always have great impact on the block erasure count regardless of the disk utilization.

C. Flash Lifetime

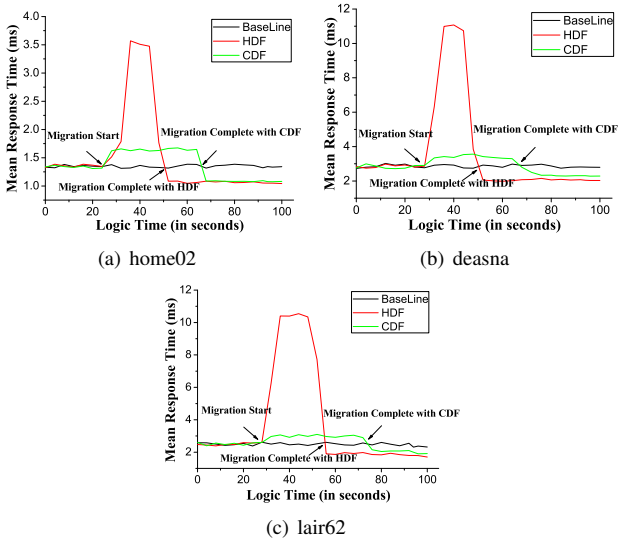
As data migration also produces some write amplification into the cluster, we use the cluster-wide *aggregate block erasure count* to evaluate this impact.

In Figure 6, we can observe that EDM can also reduce the cluster-wide aggregate block earsure count in most cases. We explain the reasons as follows. A well-balanced wear distribution mitigates the overall write amplification caused by garbage collection. For CDF, the migration process reduces the disk utilization of hot servers and the GC's write amplification of that server can also be reduced a lot accordingly, however the GC's write amplification of the destination servers would not increase too much since few write requests tend to be sent to them. As to HDF, servers with larger disk usage ratio tend to have more write requests sent to them and are more likely to be the sources of migration. In contrast, servers with lower disk utilization tend to be the destinations of migration. If this occurs, the overall write amplification can be reduced as some write requests are shifted from the servers with larger disk usage ratio to others. Consequently, the total block erases can be reduced accordingly. Figure 6 just demonstrates our point. In Figure 6, we observe that EDM can save up the aggregate block erases by up to 39%.

Additionally, although EDM-HDF can reduce the aggregate block erasure count in all cases, this value is increased in a few cases for EDM-CDF and in most cases for the CMT. This is mainly due to the fact that the total volume of moved objects in CMT is the most, followed by CDF, and both of them reflect larger amount of moved objects than

HDF. Hence, the reduced write amplification due to data shuffling might have been covered up by the increased write amplification produced by data migration in those cases. Nevertheless, in Figure 6, we observe that the aggregate block erase in CDF increases by only less than 6% compared to the baseline system. On the contrary, it increases by 21% for CMT at most. Moreover, we note that EDM-HDF and EDM-CDF show reduction of aggregate block erases by up to 40% and 34% respectively compared to the CMT. Thereby, both HDF and CDF extend the lifetime of the flash compared to the CMT.

D. Migration Costs



(a) home02 (b) deasna

(c) lair62

Figure 7. Mean Response Time with different Migration Strategies during Data Migration

To further investigate how data migration impacts on normal file access performance during migration process, we select three traces (including *home02*, *deasna*, and *lair62*) to measure the mean response time for file operations served during data migration. In this section, we compare HDF and CDF with the baseline system. The results are shown in Figure 7. The x-axis denotes the elapsed time. The y-axis is the response time per file operation. Each data point in Figure 7 is the average response time for file operations served in the past 3 minutes. For HDF, we see that the mean response time rises rapidly when the migration process starts because all the requests related to the objects being moved are blocked. After that, the mean response time recovers to below the initial setting, because the wear variance is reduced after data migration. In contrast, CDF has a relatively small impact on normal file access when migration starts, because most moved objects in CDF have very small probability of being accessed and the impact only comes from the competition of disk bandwidth.

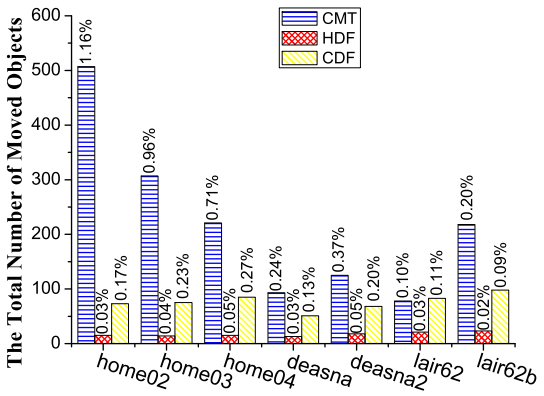


Figure 8. The Total Number of Moved Object in Different Traces. The numbers above the bars indicate percentage of the number of moved objects.

E. Remapping Table Overhead

As the total number of moved objects during migration directly impacts the growth rate of the size of the remapping table, as well as the memory consumption, we measure it during each trace replaying process for different migration policies. The results are shown in Figure 8. In this figure, we observe that the percentage of total moved objects is relatively small (at most 1%). Moreover, the total number of moved objects with CMT is the most among the 3 migration techniques, and then followed by CDF and HDF. This is due to that CMT dynamically balances both the load and storage usage to achieve performance improvement. In addition, it does not differentiate read and write operations during migrations which results in more objects being moved during migration than HDF or CDF. This experiment illustrates that CMT results in the size of the remapping table growing faster than both HDF and CDF.

VI. Related Work

Hash-based data placement policies have been widely used in parallel file systems to distribute data across servers evenly. Chord [19] uses a variant of consistent hashing [10] to assign keys to storage nodes. Choy [7] proposes extendible hashing for distributing data among servers uniformly which relocates an optimal amount of data whenever disks are added. Brinkmann [4] uses hash-functions for evenly distributing and efficiently locating data in dynamically changing SANs. Although hash-based functions are used to locate new data to storage devices with available capacity in these studies, the data would rarely be relocated to maintain a uniform distribution over time except for addition and removal of storage. CRUSH [24] supports for controlled weighting of devices and maintains a well-balanced data distribution dynamically, however, it only takes the storage usage into consideration and can not deal with the dynamical changes of I/O workload ideally.

For I/O load balancing, data migration is an on-line data reorganization method which is commonly used in many previous studies. Anderson [1] presents an automatic scheme to generate the plan for migration by measuring six parameters for each I/O stream, including request rate, request size, queue length, etc. In the system Sorrento, Tang [20] adjusts the segment's location dynamically by weighting two factors of I/O load and storage usage among the providers. BASIL [9] uses I/O latency as the primary metric for modeling of workloads and devices, and recommends migrations between devices accordingly. However, none of these migration policies has taken the lifetime issue of SSDs into consideration.

In contrast, although EDM focuses on improving the system performance, it addresses the issue of load balancing from an unique perspective that can minimize the wear caused by migration in SSD storage clusters.

VII. Conclusion

In this paper, we reveal the problem of wear variance among different devices in an SSD storage cluster. Since garbage collection process is quite time consuming, and can block the normal I/O operations during its process. It is likely to be the main reason that influences the SSD's performance, thereby the wear variance can potentially lead to load imbalance across SSDs.

Based on our observation, we propose an endurance-aware data migration scheme with carefully arranged data placement and migration in order to reduce the wear on SSDs caused by data migration, while improving the performance. To minimize the write amplification caused by migration, an SSD wear model is proposed to quantitatively calculate the amount of data movement according to wear frequency. By using this model, two complementary migration policies are designed to explore trade-offs among throughput, response time during migration, and lifetime of SSDs in a storage cluster. Our evaluations of an EDM's implementation on a real storage cluster demonstrate that EDM achieves the same effectiveness of performance improvement as existing HDD based migration techniques. In the meantime, it significantly reduces aggregate erase count compared to conventional techniques.

VIII. Acknowledgment

We would like to thank the anonymous reviewers for their comments and suggestions. This work is supported by the National Natural Science Foundation of China (Grant No. 61232003, 60925006), the National High Technology Research and Development Program of China (Grant No. 2013AA013201), Shanghai Key Laboratory of Scalable Computing and Systems, Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, Huawei Technologies Co. Ltd., and Tsinghua University Initiative Scientific Research Program.

References

[1] E. Anderson et al. Hippodrome: running circles around storage administration. In FAST, 2002.

[2] Mahesh Balakrishnan et al. Differential RAID: Rethinking RAID for SSD Reliability. In Eurosys, 2010.

[3] Peter J. Braam. The Lustre Storage Architecture. http://www.lustre.org/documentation.html, 2004.

[4] A. Brinkmann et al. Efficient, distributed data placement strategies for storage area networks. In SPAA, 2000.

[5] P. Carns et al. Pvfs: A parallel file system for linux clusters. In Proceedings of the 4th Annual Linux Showcase and Conference, 2000.

[6] L.-P. Chang et al. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. TECS 2004.

[7] D. M. Choy et al. Efficiently extendible mappings for balanced data distribution. In Algorithmica, 1996.

[8] D. Ellard et al. Passive nfs tracing of email and research workloads. In FAST, 2003.

[9] A. Gulati et al. Basil: Automated io load balancing across storage devices. In FAST, 2010.

[10] D. Karger et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In STOC, 1997.

[11] A. Kawaguchi et al. A flash-memory based file system. In Proceedings of the USENIX Technical Conference, 1995.

[12] Y. Kim et al. Flashsim: A simulator for nand flash-based solid-state drives. In Proceedings of the First International Conference on Advances in System Simulation, 2009.

[13] H. Kwon et al. Janus-ftl: Finding the optimal point on the spectrum between page and block mapping schemes. In EMSOFT, 2010.

[14] A. W. Leung et al. Measurement and analysis of large-scale network file system workloads. In USENIX ATC, 2008.

[15] J. Menon. A performance comparison of raid-5 and log-structured arrays. In HPDC, 1995.

[16] C. Mina et al. Sfs: Random write considered harmful in solid state drives. In FAST, 2012.

[17] Y. Oh et al. Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems. In FAST, 2012.

[18] M. Rosenblum et al. The design and implementation of a log-structured file system. TOCS 1992.

[19] I. Stoica et al. Chord: A scalable peer-to-peer lookup service for internet applications. In SIGCOMM, 2001.

[20] H. Tang et al. Sorrento: A self-organizing storage cluster for parallel data-intensive applications. In SC, 2004.

[21] L. Useche et al. Exces: External caching in energy saving storage systems. In HPCA, 2008.

[22] W. Wang et al. Hylog: A high performance approach to managing disk layout. In FAST, 2004.

[23] S. A. Weil et al. Ceph: A scalable, high-performance distributed file system. In OSDI, 2006.

[24] S. A. Weil et al. Crush: controlled, scalable, decentralized placement of replicated data. In SC, 2006.

[25] B. Welch et al. Managing scalability in object storage systems for hpc linux clusters. In MSST, 2004.