WeLe-RAID: A SSD-based RAID for System Endurance and Performance

Du Yimo, Liu Fang, Chen Zhiguang, Ma Xin

Department of Computer Science, National University of Defense Technology, Changsha, China
poshand@163.com, fangliu@nudt.edu.cn, chenzhiguanghit@gmail.com, mxviking@yaho.com.cn

Abstract. Due to the limited erasure/program cycles of flash memory, flash-based SSDs need to prolong their life time using wear-leveling mechanism to meet their advertised capacity all the time. However, there is no wear-leveling mechanism among SSDs in RAID system, which makes some SSDs wear out faster than others. Once any one of SSDs fails, reconstruction must be triggered immediately. But, the cost of this process is so high that the reliability and availability is affected seriously. We propose WeLe-RAID which introduces Wear-Leveling mechanism among flash SSDs to enhance the endurance of entire SSD-based RAID system. As we know that under the workload of random access pattern, parity stripes suffer from much more updates because every update to the data stripe would cause the modification to the related parity stripe. Based on this principle, we introduce age-driven parity distribution scheme to guarantee the wear-leveling among flash SSDs. At the same time, because of age-driven parity distribution, it brings into the performance benefit with better load balance. Compared with conventional RAID mechanism, it significantly improves the life span and performance with ignorable overhead.

Keywords: SSD, RAID, wear-leveling, endurance, performance, reliability

1 Introduction

SSD (Solid State Drive) exhibits higher speed and lower power consumption than disk. To some degree, it alleviates the I/O bottleneck in computer system by replacing disk. For the compatibility, it offers standard interfaces like disk, which can use previously calculated hardware and software based on disks. However, there are three critical technical constraints for flash memory [1]: (1) No in-place update, that means the whole block must be erased before overwriting data in any page; (2) No random write in a page, for the reliability, pages in a block need to be programmed in a sequential order rather than random writes. (3)Life limits, block will wear out after a certain number of program cycles. To cope with these obstacles, many strategies have been proposed respectively. In this paper, we mainly focus on the problem of the third constraint that flash memory has the limited erasure/program cycles. Actually, almost all the SSD products in the market supplied by kinds of vendors adopt the wear-leveling schemes to make all blocks in the SSD wear out evenly to guarantee their advertised capacity. Over provision of capacity also has the same goal, while not just meet the requirement of garbage collection. These two strategies work together to prolong the lifespan of SSD, although they cannot increase the total program cycles of all the blocks.

RAID mechanism has already been a very effective and popular method to construct high performance and reliable storage system since it is firstly published in 1988 [2]. It uses redundancy scheme to improve reliability and stripe scheme to promote throughput. In this case, using inexpensive disks with a little cost, it can construct a high performance storage system.

As SSD has wider applicable area, it would be a nature idea to construct storage system using the techniques mixed with RAID mechanism and state-of-the-art SSD. SSD has internal wear-leveling strategies to prolong its lifespan with advertised capacity. Once it cannot afford equivalent capacity as what vendors claimed in their product introductions, it no longer can supply the good service to meet users` requirements. So, wear-leveling is very necessary in SSD. As we know that, RAID controller does not have the wear-leveling mechanism to guarantee that all the SSDs in the RAID system wear out synchronously. Once a disk fails because of reaching its life limit, it costs too much time to replace it and reconstruct data on it using the algorithm based on parity. In this paper, we propose a novel method which adopts parity distribution based wear-leveling scheme among SSDs named WeLe-RAID to make the entire RAID system effectively work longer. The WeLe-RAID has three properties as follows:

(1) Age-driven parity distribution. As we know that, RAID4 assigns parity in unique device, and RAID5 assigns parity evenly which means every device has the same fraction of parity, while WeLe-RAID distributes parity according to its age dynamically. If some SSDs have higher erased number than others, and once this gap reaches the previously assigned critical value, we need reallocate the parity on these SSDs: more parity on younger SSDs and less parity on older SSDs.

(2) Less replacement in the lifecycle of entire RAID system. Since using the wear-leveling mechanism in the entire SSD-based RAID system, every device has afforded a part of workload so that all the devices can serve longer comparatively. It will be a long time until all the devices simultaneously approach their life critical value. Before that point, we have enough time to back up all the data on other new devices. Then totally replace the old ones. Consequently, in the lifecycle of the entire RAID system, less replacement needed than the previous system without weal-leveling mechanism among SSDs.

(3) Optimized addressing method with age-driven parity distribution. Conventional RAID mechanism adopts round-robin data layout [16], which the mapping relationship can be represented through simple function. However, age-driven parity distribution makes addressing more complex. In this paper, we give the original and optimized data layout and addressing method respectively. And the optimized one is much more effective.

The rest of the paper is organized as follows: Section 2 gives the motivation of this paper by analyzing previously main work which cannot meet the needed requirement. Section 3 describes the design and related algorithms in detail. Section 4 is the evaluation of WeLe-RAID. Section 5 introduces some related work. The last part is conclusion summing up the works in this paper.

2 Problem Description

2.1 Why need wear-leveling

SSD has the internal wear-leveling mechanism to prolong its life span with the capacity advertised by the vendors. However, Diff-RAID [3] figures out that wear-leveling mechanism among SSDs will lead to high probability of correlated failures. On the contrary, it attempts to create and maintain the age difference among SSDs to guarantee at least some devices have lower bit error rate to avoid high correlated failure rate. We think it is very useful when the bit error rate of flash chip gradually rises in its whole life. Actually, for SLC, the bit error rate of flash chip does not have linear relationship with its age, even almost maintain zero until they reach their rated lifetime. For most MLC models, the bit error rate increases sharply shortly after their rated life times, and some start to increase sharply even earlier. Before they hit their rated lifetime, they can maintain comparatively stable bit error rate. And with the correctness of ECC, this climbing trend will be slowed down further [4]. In order to keep the age difference while the oldest SSD is retired, Diff-RAID has to replace the retired one with the new one, then reconstruct data and redistribute parity. We use a common equation 1 [10] to approximately evaluate the reliability of SSD-based RAID5 system. In this equation, MTTDL (Mean Time To Data Lose) is marked as the metric of system reliability. MTTF means Mean Time To Failure of single device. MTTR means Mean Time To Repair a failed device. From the equation 1, we can see that if the procedure of reconstructing data and redistributing parity is complex and high time-cost, it will be apt to loss data because any device failing at this moment could cause data corruption. If we use wear-leveling among SSDs, we can prolong the endurance of the entire system, which reduces the number of replacement to avoid more fragile moments.

$$MTTDL = \frac{MTTF^2}{N(N-1)MTTR} \qquad \textbf{(1)}$$

Otherwise, wear-leveling among SSDs brings into the performance benefit with better load balance since parity stripes suffer from much more updates because every update to the data stripe would cause the modification to the related parity stripe.

2.2 Why not RAID5

Through the above discussion, we know that, in most occasions, wear-leveling in the entire RAID system is useful and necessary. Since parity is the key factor of affecting wear-leveling because devices allocated more parity wear out faster, RAID5 adopting evenly parity distribution scheme may work well on wear-leveling in system level. However, through the experiment, we find that RAID5 cannot ensure the wear-leveling among devices either under some workloads. Figure 1 has proven that by showing the result of wear distribution under different workloads. This experiment is done on the simulator described in paper [6]. Set a counter in each SSD and increase itself when meet an erasure. After running the trace, total counter number on each device is its wear situation. This situation is resulted from that some workloads access some certain parity more often so that devices holding this parity suffer more updates and wear out faster than others.

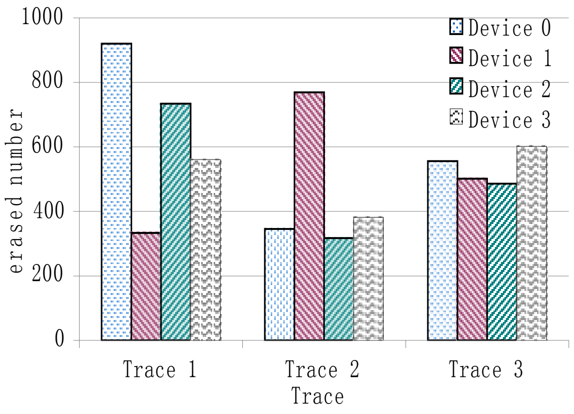


Fig. 1. Erased number of each SSD for RAID5. Here RAID5 consists of four SSDs.

2.3 Why not other schemes

Wear-leveling mechanism among SSDs in RAID system has been referred in previously paper. Kwanghee Par etc [7] give a brief design of wear leveling for SSD-based RAID5. It uses a big table to restore the erased number of stripe in each SSD respectively. When some parity hit the previously sat number, exchange the hot parity and cold parity with the greedy algorithm. This method costs much extra space and greedy algorithm is so complex that the performance is restricted seriously.

WeLe-RAID can balance the wear grade among devices based on parity distribution to prolong endurance and improve performance. Meanwhile it is simple to implement and has tolerable time cost and space cost.

3 WeLe-RAID

3.1 The architecture of WeLe-RAID

Figure 2 illustrates the architecture of WeLe-RAID. WeLe-RAID has two controllers. One is RAID controller which manages a group of running SSDs called active devices in figure 2 to offer the service; the other is migration controller triggered when the entire system approaching the end of its lifetime to migrate the data from the active devices to the prepared ones, then replace the old devices with prepared ones. After replacement, prepared devices have become the active ones and new devices are brought in as prepared ones. Because wear-leveling scheme among SSDs is incorporated in our RAID mechanism, all SSDs of RAID system can be promised to maintain the same level of wear grade and can be totally replaced in one time. If any one of devices failed earlier before its life limit, the corresponding prepared one would replace it at once and reconstructing process would be triggered to resume the data on it.

Control flow and data flow both can be seen from figure 2. RAID controller administrates active devices below it and connects with migration controller to activate it when active devices nearly approach their life limits. Then migration controller put on the switch between active devices and prepared ones to create data path between them. In common cases, data flows merely through RAID controller to supply service for users. RAID controller has the implementation of basic RAID mechanism and our proposed parity distribution schemes. Migration controller doesn`t need much complex hardware because it only needs the function of migration which copies data from old devices to the same address of the new ones. Certainly, this process usually proceeds when the system is idle to avoid competition of responding I/O requests.

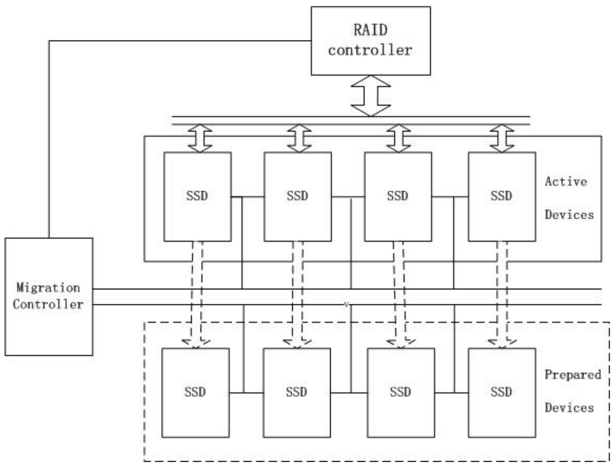
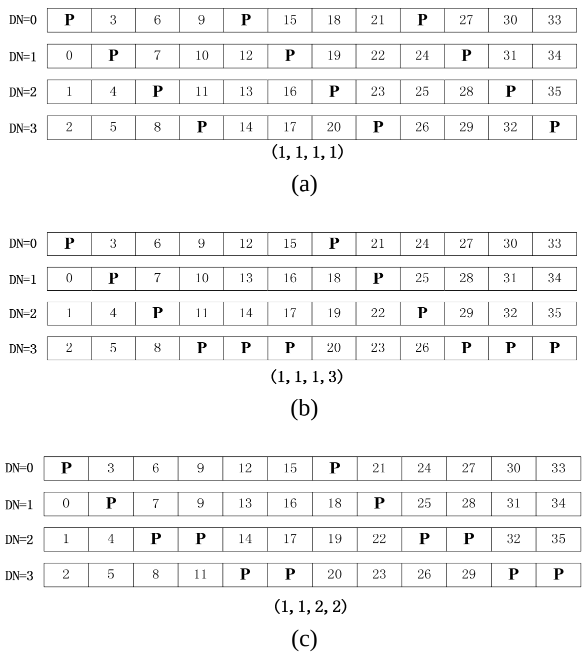


Fig.2. The architecture of WeLe-RAID. Prepared devices in dash line frame are not in the system all the time. They are plugged in only when they are needed.

3.2 Data Layout

WeLe-RAID introduces dynamic age-driven parity distribution strategy like Diff-RAID [3]. We describe this relationship between parity distribution and age distribution quantitatively as follows. Given the age of RAID system consisting of n SSDs represented by a N-tuple (a_1, a_2, … a_n) which a_1, a_2, … a_n has no prime number, we can compute its variance (S) to evaluate its age difference. If it exceeds the critical value (CA), the process of parity redistribution must be called to make the entire RAID system retain the similar wear grade. The parity distribution represented with (p_1, p_2, …, p_n) can be made according to the age distribution through following two steps: 1. Sort a_1, a_2, … a_n in descend order; 2. p_k equals the k_{th} value in age`s descend order.



DN=0	**P**	3	6	9	**P**	15	18	21	**P**	27	30	33
DN=1	0	**P**	7	10	12	**P**	19	22	24	**P**	31	34
DN=2	1	4	**P**	11	13	16	**P**	23	25	28	**P**	35
DN=3	2	5	8	**P**	14	17	20	**P**	26	29	32	**P**

(1, 1, 1, 1)

(a)

DN=0	**P**	3	6	9	12	15	**P**	21	24	27	30	33
DN=1	0	**P**	7	10	13	16	18	**P**	25	28	31	34
DN=2	1	4	**P**	11	14	17	19	22	**P**	29	32	35
DN=3	2	5	8	**P**	**P**	**P**	20	23	26	**P**	**P**	**P**

(1, 1, 1, 3)

(b)

DN=0	**P**	3	6	9	12	15	**P**	21	24	27	30	33
DN=1	0	**P**	7	9	13	16	18	**P**	25	28	31	34
DN=2	1	4	**P**	**P**	14	17	19	22	**P**	**P**	32	35
DN=3	2	5	8	11	**P**	**P**	20	23	26	29	**P**	**P**

(1, 1, 2, 2)

(c)

Fig.3. Basic data layout of WeLe-RAID. (a) shows the data layout under parity distribution (1, 1, 1, 1); (b) shows the data layout under parity distribution (1, 1, 1, 3); (c) shows the data layout under parity distribution (1, 1, 2, 2).

Figure 3 is the basic data layout of WeLe-RAID. It exhibits some characteristics. At the first beginning, we suppose the devices are new and their age distribution is (1, 1, 1, 1) shown in figure 3-(a). For wear-leveling, we make parity distribution be (1, 1, 1, 1). Actually, it is the RAID5 scheme assigning parity evenly across all devices. However, just like what we said in section 2.2, RAID5 cannot ensure wear-leveling completely among SSDs either. After a running period, the age gap among SSDs appears which age distribution is (3, 3, 3, 1). The age difference can be described as variance (S) which the value is 3. If it exceeds CA, parity redistribution needs to be called to mitigate age difference: more parity on younger device and less parity on older device. Figure 3-(b) illustrates the data layout of new parity distribution (1, 1, 1, 3) made according to age distribution. Then, after another running period, the age distribution is (2, 2, 1, 1) whose variance is 1. Compared with last variance, its difference gap significantly becomes smaller. Suppose it still exceeds CA, then parity redistribution need to be called again. Figure 3-(c) shows the data layout of corresponding parity distribution.

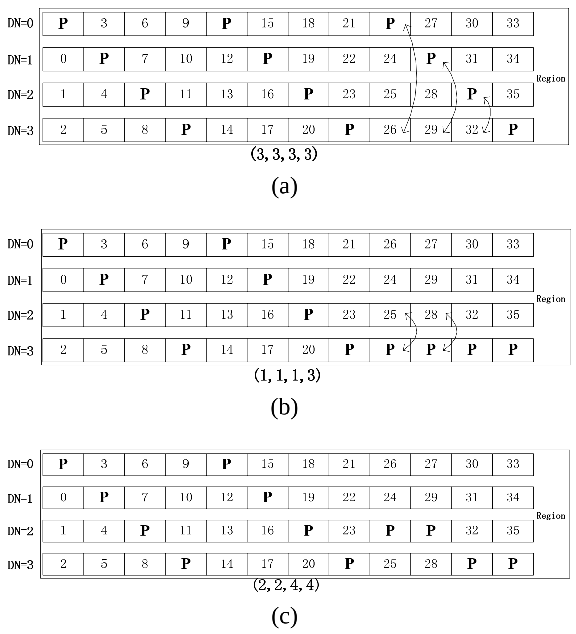
In figure 3, data layout adopts round-robin striping scheme that has simple addressing policy but causes huge migration once parity distribution scheme changed. Figure 4 gives an improved data layout. Adopting this data layout, every parity redistribution operation brings small amount of shifts between data and parity. The procedure of parity shift from an original distribution to a new distribution can be depicted as follows:

1. Compute the region number. The region number is determined by computing the minimum common multiple of last region and sum of each fraction in new distribution. The first region is the sum of each fraction in the original parity distribution. From figure 4-(a) to figure 4-(b), we can compute the region number which is 12 equaling to minimum multiple common of 4 (sum of (1, 1, 1, 1)) and 6 (sum of (1, 1, 1, 3)).

2. Amplify the fraction in each part of parity distribution equation according the region number. So the parity distribution is changed from figure 3-(a) (1, 1, 1, 1) to figure 4-a (3, 3, 3, 3) and from figure 3-(b) (1, 1, 1, 3) to figure 4-(b) (2, 2, 2, 6).

3. Exchange the parity and data block in corresponding area according to the newly computed parity distribution.

Compared with basic data layout, improved data layout migrates less data. Although it cannot guarantee parity distribution according to age evenly in the finest grain, it is quite uniform from the point of the whole layout.



DN=0 P 3 6 9 P 15 18 21 P 27 30 33
DN=1 0 P 7 10 12 P 19 22 24 P 31 34
DN=2 1 4 P 11 13 16 P 23 25 28 P 35
DN=3 2 5 8 P 14 17 20 P 26 29 32 P
Region
(3, 3, 3, 3)
(a)
DN=0 P 3 6 9 P 15 18 21 26 27 30 33
DN=1 0 P 7 10 12 P 19 22 24 29 31 34
DN=2 1 4 P 11 13 16 P 23 25 28 32 35
DN=3 2 5 8 P 14 17 20 P P P P P
Region
(1, 1, 1, 3)
(b)
DN=0 P 3 6 9 P 15 18 21 26 27 30 33
DN=1 0 P 7 10 12 P 19 22 24 29 31 34
DN=2 1 4 P 11 13 16 P 23 P P 32 35
DN=3 2 5 8 P 14 17 20 P 25 28 P P
Region
(2, 2, 4, 4)
(c)

Fig.4. Improved data layout of WeLe-RAID. (a) shows the data layout under parity distribution (1, 1, 1, 1); (b) shows the data layout under parity distribution (1, 1, 1, 3); (c) shows the data layout under parity distribution (1, 1, 2, 2).

3.3 Addressing method

The key to implement WeLe-RAID is to design the mapping mechanism between logical address and physical address in the controller. When the controller receives an

I/O request, it uses striping scheme to partition the data into several parts. And send the data and parity to the certain related devices according to the mapping relationship. Round-robin placement scheme is popularly used in RAID system. In this method, data layout is ascertained before. So any logic block address can be mapped to physical address easily through function without looking up operation. However, it lacks a little flexibility. The other method seldom used is mapping table. It is more flexible, but lead to high pressure on time cost and space cost. Usually, dynamic parity distribution like WeLe-RAID needs more flexible mapping data structure just like mapping table. But, in this paper, we use former method which still meets our requirement.

Traditional RAID scheme distribute parity either to a dedicated device like RAID4 or across all devices evenly like RAID5. From their different data layouts, we can compute the physical address using a linear function respectively. The addressing function of RAID4 can be summarized as follows:

$$\begin{aligned} SN &= LBA / \left(N - 1 \right) \\ PN &= N - 1 \\ DN &= LBA \bmod \left(N - 1 \right) \end{aligned} \qquad (2)$$

In above equations, LBA means logical block address of the data unit after partition. N means the number of devices including data devices and parity devices. SN means stripe group number representing the stripe group allocated for the data. PN means the number of device that stores parity related with current data. DN means the number of device that stores current data. RAID5`s addressing function is displayed as follows:

$$\begin{aligned} SN &= LBA / \left(N - 1 \right) \\ PN &= SN \bmod N \\ DN &= \begin{cases} LBA \bmod \left(N - 1 \right) + 1 & if\ LBA \bmod \left(N - 1 \right) >= PN \\ LBA \bmod \left(N - 1 \right) & if\ LBA \bmod \left(N - 1 \right) < PN \end{cases} \end{aligned} \qquad (3)$$

For WeLe-RAID, in different time period, it has different parity distribution caused by age difference of devices. Every device`s age can be denoted by the average number of all blocks` age. If we use the data layout as figure 3 shows. The address function can be stated as follows:

$$\begin{aligned} SN &= LBA / \left(N - 1 \right) \\ PN &= \begin{cases} 0 & if\quad SN \bmod \left(p_1 + p_2 + \ldots + p_n \right) <= p_1 - 1 \\ 1 & if\quad p_1 <= SN \bmod \left(p_1 + p_2 + \ldots + p_n \right) <= p_1 + p_2 - 1 \\ \ldots & \\ N - 1 & if\quad p_{n-1} <= SN \bmod \left(p_1 + p_2 + \ldots + p_n \right) <= p_1 + p_2 + \ldots + p_n - 1 \end{cases} \\ DN &= \begin{cases} LBA \bmod \left(N - 1 \right) + 1 & if\ LBA \bmod \left(N - 1 \right) >= PN \\ LBA \bmod \left(N - 1 \right) & if\ LBA \bmod \left(N - 1 \right) < PN \end{cases} \end{aligned} \qquad (4)$$

We have pointed out the drawback of data layout shown in figure 3. Then we present an improved data layout to reduce the data migration. With improved data layout, we can give the algorithm depicted in pseudo code 1 to solve addressing problem. This addressing algorithm is run when redistribution has been completed according to the layout depicted in last section.

In the algorithm, parity redistribution history and region history must be recorded as permanent variables. SN can be computed directly like any previous RAID mechanism. Because of exchange between parity and data when parity redistribution occurred, only using one function cannot meet the requirement of addressing parity device number (PN) and data device number (DN). Hence, we reserve all region history and parity distribution history in order to attain the current physical address of parity and common data through the algorithm described in pseudo code 1.

Pseudo Code 1

```
  Algorithm: Address (LBA, t, p[t][N], region[t], SN, PN,
DN)
  Input:
  LBA: logical block address
  t: redistribution times
  p[t][N]: parity distribution history
  region[t]: region history
  Output:
  SN: stripe group number
  PN: parity device number
  DN: data device number
  1. SN=LBA/(N-1)
  2. if (t=0)         //no parity redistribution happens
  3.    PN=SN%N
  4.    if LBA%N-1>=PN then
  5.        DN=LBA%(N-1)+1
  6.    else
  7.        DN=LBA%(N-1)
  8. else
  9.    mmn = minimum multiple number(region[t],
         region[t-1])
  10.   x=mmn/region[t-1], y=mmn/region[t]
  11.   for(j=0; j<N; j++) //amplify the fraction in
                            parity distribution expression
  12.       d[j]=x*p[t-1][j]-y*p[t][j]
  13.       for(j=0; j<N; j++)
  14.           if (d[j]>0)
  15.           for(k=N-1; k>=0; k--)  //exchange data and
                                        parity
  16.              if (d[k]<0)
  17.                 if(PN==j)
  18.                    PN=k
  19.                 if(DN==k)
  20.                    DN=j
  21.   Address(LBA, t-1, p[t-1][N], region[t-1], SN, PN,
            DN)
```

Actually, there is another way to deal with addressing problem. We could use a table to reserve the number of device to which the data and correlated parity should be sent. Through looking up the table, we can get its physical address of any logical block. But the operation of looking up may cost too much time and the table must be huge to accommodate all the mapping relationship which causes extreme pressure to the space in the RAID controller. Compared with this mapping table method, what we proposed is more effective. We know that the time of redistribution is very few because it is only called when the age difference exceeds the critical value which is usually a little high. So the algorithm described in pseudo code 1 costs little time and absolutely saves space since it has no complex data structure.

4 Evaluation

We measure the performance, endurance and reliability of WeLe-RAID compared with other SSD-based RAID system. We have constructed a SSD simulator in paper [6]. In order to compare the performance and endurance between WeLe-RAID and other RAIDs, we implement these RAID schemes respectively in software RAID mode. Then we run some traces [9] collected from PC and servers under the real workloads. Through the simulation experiment, we can attain the average latency which can be used as a metric of performance. After running the traces, we check the average erased number of each block on each SSD which represents the endurance of the entire RAID system. For reliability, we adopts mathematic model to analyze the MTTDL (mean time to data lose) that is the common criteria to evaluate reliability.

4.1 Performance of WeLe-RAID

The performance experiment is done on the state that redistribution has been completed, and the data layout is stable. Figure 5 shows the average latency for different RAID mechanism under kinds of workloads. We can see from the figure that the performance of WeLe-RAID is better than another two: outperforms RAID5 about 10% and outperforms Diff-RAID 30% approximately sometimes even 40%. The reason for that is WeLe-RAID adopts age-driven parity distribution. Since the device that has the higher wear grade means it suffers from more writes, transforming some parity from older device to younger device, to some degree, balances the write workload. That is why it outperforms RAID5 although RAID5 distributes parity absolutely evenly. Diff-RAID distributes more parity on the older device to create age gap to improve reliability. However, it makes older device which has already been responsible for more requests to undertake more coming ones. It aggravates the unbalance of loads among SSDs and lead to the bottleneck of some certain device. Consequently, its performance is extremely influenced which is the worst among the three methods.

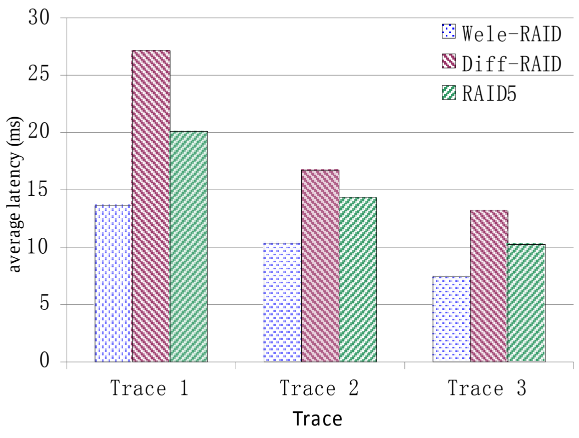


Fig.5. Average latency of different RAID systems under various traces

Parity redistribution is necessary in the WeLe-RAID and Diff-RAID. But this procedure must cost so much time that the performance is extremely decreased. Diff-RAID has not given the implementation of parity redistribution in detail. If it does not make any improvement in parity migration, it will cost too much time. Figure 6 displays parity redistribution time of WeLe-RAID under different data layouts with optimization and without optimization. It shows that improved layout has much smaller overhead.

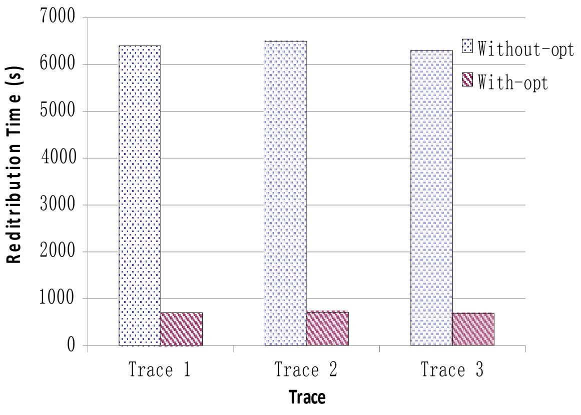


Fig.6. Redistribution time of WeLe-RAID with unimproved data layout and improved data layout under various traces

4.2 Endurance of WeLe-RAID

After running each trace, we collect the erased number of each block on each SSD. We use average erased number of total blocks in each SSD to represent the age of each SSD. Figure 7 displays the age difference of different RAID schemes under different workloads. We use standard deviation of all SSDs` erased number to evaluate the devices` age difference. Apparently, WeLe-RAID has the most uniform age distribution which means the endurance of entire WeLe-RAID system has been prolonged.

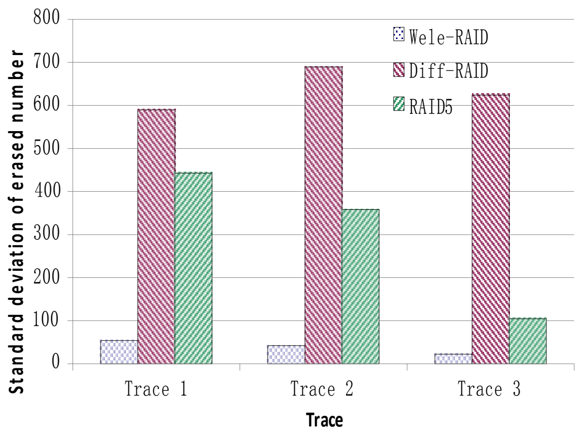


Fig.7. Devices` age difference of different RAID under various traces

4.3 Reliability of WeLe-RAID

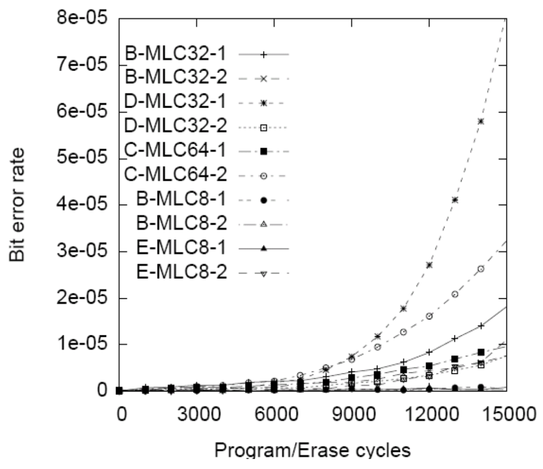


Fig.8. Wear out and error rate for MLC flash device [4]

Currently, reliability is a tough problem for flash-based SSD. For one thing, reliability constrains the speed of market development of flash. For other thing, there is no dependable model on reliability for SSD. For traditional disks, some research has been done on their failure models. In term of these models, various methods have been proposed such as Markov model [10], simulation and shortcut method. [11] This paper analyzes the reliability of SSD-based RAID system with mathematic method.

Before we measure the reliability of SSD-based RAID system, we must understand the reliability of single flash chip. Figure 8 shows the relationship between raw bit error rate (RBER) and age for MLC flash. RBER can be reduced prominently with ECC, which is called uncorrectable bit error rate (UBER). And UBER is some orders magnitude lower than RBER because ECC in every page can correct most errors.

WeLe-RAID and Diff-RAID are both one failure tolerant RAID like RAID5 belonging to 1-of-N system. For disk-based RAID5, we can use equation 1 referred in section 2 to estimate the reliability. When disks are replaced by SSD, the failure rate of each device cannot be treated as same constant any more. Due to the uniform age of each device in WeLe-RAID, we can assume the failure rates of all devices are the same. For SLC flash, RBER is stable before flash approaches its rated lifetime. Especially, it is almost changeless with ECC. For MLC, RBER rises along with the increase of the age. But the change is not obvious when it is far away from its rated lifetime, and ECC has narrowed down the changing speed. WeLe-RAID transforms data from old devices to prepared ones before they reach their life limits. So no device whose age approaches life limit is in service of storing data. Based on this point, we can use a constant to approximately express the failure rate. Therefore, we can still use equation 1 to evaluate the reliability of WeLe-RAID. The data loss possibility (DLP, reciprocal of MTTDL) of Diff-RAID converges to steady state after several replacements [3]. We can attain the DLP of Diff-RAID and RAID5 from paper of Diff-RAID. Given the UBER of flash used in Diff-RAID [3], we can compute the DLP of WeLe-RAID and compare it with Diff-RAID in table 1. We can see from table 1 that WeLe-RAID`s DLP is approaching the Diff-RAID`s.

Table 1. Reliability of Diff-RAID and WeLe-RAID

	Diff-RAID	WeLe-RAID
Data lose possibility	1e-06	5e-06

5 Related work

RAID mechanism can improve the performance and reliability of storage system based on flash memory. We classify flash storage system as two categories according to the grain of device applied with RAID mechanism. One is coarse grain which uses RAID mechanism on SSDs. This mode is completely compatible to previous RAID technology including software and hardware. But while solid state disks hide the difference with hard disks by flash translation layer (FTL), the potential advantages of flash is hidden too. So the fine grain which uses RAID mechanism on flash chips can exploit the characteristic of flash totally.

Mao Bo, Jiang Hong etc [12] have designed a hybrid RAID4 named HPDA composed by disk and SSDs. It uses disk to respond to the write-intensive accesses as dedicated parity device which is the shortage of SSD. It improves the performance and reliability compared with RAID4 composed totally with SSDs. But it also has the drawback of traditional RAID4 that parity device is the bottleneck. Kadav Asim etc [3] have proposed a novel SSD-based RAID called Diff-RAID which has been discussed in detail in section 2.

Meanwhile, some research has been done on constructing RAID system with flash chips. Lee Yangsup etc [13] design a flash-aware redundancy array abbreviated FRA. It separates the parity updating from the write procedure, and deal with it when the system is idle. Thereby, this lazily parity updating method improves write performance prominently. Yu-Bing Chang etc [14] have proposed a self-balanced flash array. It encodes hot data into several replicas stored on different banks. Thus, requests on hot data can be directed to cold banks that are responsible for fewer requests. However, every update brings modification of parity extremely affecting the write performance and causing high space cost. Sooju Im etc [15] present a flash-aware RAID technique for dependable and high-performance flash memory SSD. It delays the parity update accompanied with every data write to decrease parity handling overhead.

6 Conclusion and future work

In this paper, we propose a novel SSD-based RAID called WeLe-RAID. It uses dynamic parity distribution scheme which is adaptive to age of SSD. Based on the principle that parity stripe suffers from more modification, we allocate more parity to younger SSD and less parity to older SSD which ensures wear-leveling among SSDs and to some degree alleviates unbalance of write loads on devices. We give the data layout and addressing algorithm of WeLe-RAID in detail. To implement this method, what we need is only some simple data structure to record a little redistribution history information. Through the experiment, we can see that with ignorable overhead, WeLe-RAID outperforms other RAIDs in average response time. And it has comparable reliability with Diff-RAID.

What this paper discussed is one device failure tolerant RAID such as RAID4 and RAID5. Two or more devices failure tolerant RAID like RAID6 has not been introduced currently. It is more complex because multiple parity in one stripe group. In future work, we will intend to design the good layout for RAID6 to decrease the migration when redistribution.

Acknowledgments. We appreciate Lipin Chang, Po Liang and Joshon for some of the traces, and anonymous reviewers for valuable suggestions to improve this paper. This work is partially supported by National Natural Science Foundation of China grants NSFC60736013 and NSFC60903040.

References

1. Chen, F., Luo, T., Zhang, X.D.: CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives. In: Proceedings of FAST`11. San Joes, USA (2011)

2. Patterson, D., Gibson, G., Katz, R.H.: A Case for Redundant Arrays of Inexpensive Disks (RAID). In: Proceedings of SIGMOD`88. New York, USA (2011)
3. Balakrishnan, M., Kadav, A., Prabhakaran, V., Malkhi, D.: Differential RAID: Rethinking RAID for SSD reliability. In: Proceedings of Eurosys`11. Salzburg, Austria (2011)
4. Grupp, L.M., Caulfield A.M., Coburn, J., Swason, S., Yaakobi, E., Seigel, P.H., Wolf, J.K.: Characterizing flash memory: anomalies, observations, and applications. In: Proceedings of MICRO`09. New York, USA (2009)
5. Thomasian, A., Blaum, M., High Reliablity Redundant Disk Arrays: Organizations, Operation, and Coding. In: ACM Transaction on Storage, 2009, Vol. 5(3): Articale 7.(2009)
6. Du, Y.M., Xiao, N., Liu, F., Chen, Z.G.: A Customizable and Modular Flash Translation Layer(FTL) Design and Implementation. In: Journal of Xi'an Jiaotong University, 2010, Vol. 44(8): Page 42-47.(2010)
7. Park, K., Lee, D.H., Woo, Y., Lee., G.Y.: Reliability and Performance Enhancement Technique for SSD array storage system using RAID mechanism. In: Proceedings of ISCIT 2009. Incheon, Korea (2009)
8. Zheng, W.M., Zhang, G.Y.: FastScale: Accelerate RAID Scaling by Minimizing Data Migration. In: Proceedings of FAST`11. San Jose, USA (2011)
9. Chang, L.P., Kuo., T.W.: An Adaptive Stripping Architecture for Flash Memory Storage Systems of Embedded Systems. In: Proceedings of IEEE Eighth Real-Time and Embedded Technology and Applications Symposium (RTAS). San Jose, USA (2002)
10. Geist, R., Trivedi., K.: An analytic Treatment of the Reliability and Performance of Mirrored disk subsystems. In: Proceedings of Twenty-Third Inter. Symp. On Fault-Tolerant Computing, FTCS-23. (1993)
11. Thomasian, A.: Shortcut method for reliability comparisons in RAID. In: The Journal of Systems and Software, 2006, 79: 1599-1605. (2006)
12. Mao, B., Jiang, H., Feng, D., Wu, S.Z., Chen, J.X., Zeng, L.F., Tian, L.: HPDA: A Hybrid Parity-based Disk Array for Ehnaced Performance and Reliability. In: Proceedings of IPDPS`10. San Atlanta, USA (2010)
13. Lee, Y., Jung, S., Song. Y.H.: FRA: A Flash-aware Redundancy Array of Flash Storage. In: Proceedings of CODES+ISSS`09. (2009)
14. Chang, Y.B., Chang. L.P.: A Self-Balancing Striping Scheme for NAND-Flash Storage Systems. In: Proceedings of the 2008 ACM symposium on Applied computing. (2008)
15. Soojun, I.M., Shin, D.K.: Flash-Aware RAID Techniques for Dependable and High-Performance Flash Memory SSD. In: IEEE Transaction On Computers, 2011, Vol. 60(1): Page: 80-92. (2011)
16. Zhen, W.M, Zhang, G.Y.: FastScale: Accelerate RAID Scaling by Minimizing Data Migration. In: Proceedings of FAST`11. San Joes, USA (2011)