Comput Syst Sci & Eng (2018) 5: 309–316

International Journal of
Computer Systems
Science & Engineering

ADL-RAID: Energy-saving data layout for dynamic loads

Youxi Wu[1,2], Shengyuan Shi[1,2], Jingyu Liu[1,2]*, and Huaizhong Zhu[1,2]

[1] *(School of Computer Science and Engineering, Hebei University of Technology, Tianjin, 300130)*
[2] *(Hebei province key laboratory of big data calculation, Tianjin, 300130)*

With the rapid development of information technology and the explosive growth of data, the scale of storage equipment is growing rapidly. The high energy consumption storage devices have become a serious problem for data centers. For data storage, the model of sequential data access a storage system is commonly used. Aiming at the characteristics of a sequential data storage system, this paper, we propose a structure, named Adapted to Dynamic Load based on Redundant Array Independent Disks (ADL-RAID) which is an effective energy-saving data layout for dynamic loads based on the existing Semi-Redundant Array Independent Disks (S-RAID). ADL-RAID inherits the local parallel energy-saving strategy, uses an address mapping mechanism, and allocates storage space to satisfy the performance requirements for the user requested application. By sensing different loads, ADL-RAID allocates storage space for its dynamic loads with the appropriate parallelism. One or several data disks are used when the load is minimized, and all the data disks are used in parallel when the load is maximized. Experimental results show that, for 100% continuous write request, ADL-RAID saves 33.6% energy consumption than S-RAID5 and improves write performance than S-RAID5 by 34.3%. Thus, ADL-RAID has higher availability and is ideal for sequential data storage applications

Keywords: Disk array; sequential data access; energy-saving storage; data layout; dynamic load

1. INTRODUCTION

With the advent of the internet age, big data [1] has become a popular topic in the computing industry. The surge of data has led to a dramatic increase in storage devices, and the Internet Data Center predicts that the global amount of data will reach 44ZB by 2020 [2] Thus, the problem of data storage becomes the most concerned about the data center. Disk storage is currently the most popular data storage method with regard to research.

Research on saving energy in the storage system has attracted wide attention. Energy-saving research for the early storage system mainly focused on the disk level, and the hardware of the disk is improved according to the operating principle and physical characteristics of the disk. For example, Li *et al* proposed a workload awareness scheme (WAS) [3] that writes different hot data blocks into different areas of the Solid-State Drive Redundant Array Independent Disk (SSD-RAID) to improve the performance and durability of the SSD-RAID. Sun *et al* proposed a high-performance disk array for sequential storage systems: Ripple-RAID [4], which uses a new local parallel data layout to improve the write performance. However, multi-speed disk and parallel technology is still in the theoretical stage of research, and large-scale deployment remains to be seen.

The energy-saving research of the disk-level is limited for large-scale storage systems. In recent years, energy-saving research has focused on the node level, such as the grouping strategy based on the disk, MAID [5] - the energy-saving program based on the data layout, Popular Data Concentration (PDC) [6] - the update algorithm based on the parity, BPU [7] - using queue length awareness, line up according to the order of arrival of the load, trying to reduce the average response time. Sun *et al* [8] proposed the EPU algorithm as a typical representation, analyzing which is better in the scaling scenario for read-change-write and read-reconstruct-write to reduce the request of the user for scaling time. In addition, Zhan *et al* [9] designed a two-level cache algorithm to improve performance bottlenecks. Cai *et al* [10] removed code and used other techniques to improve the write performance. Zhang designed a parallel encoding algorithm, called Equation Oriented Parallel Coding (EOPC) [11] to reduce the computational complexity.

*Corresponding author. E-mail: liujingyu@scse.hebut.edu.cn

In addition, Liu *et al* proposed a mixed S-RAID structure [12] consisting of a Solid-State Drive (SSD) and a Hard Disk Drive (HDD), placing a small amount of random read and write data including super blocks in the SSD and, placing sequential data on the S-RAID composed of the HDD disk groups in an idle state are turned off to improve the energy efficiency and performance. To further enhance the system performance they proposed a write optimization strategy [13].

According to analysis of the load characteristics and access patterns of sequential data storage, an energy-saving disk array S-RAID [14] is proposed to satisfy the performance requirement, and a local parallel data layout is used to achieve energy-saving. For typical sequential data storage applications, S-RAID achieves significant energy saving. Whereas S-RAID's local parallel data layout [15] is static, it is suitable for smoother workloads, its adaptability is poor for strong fluctuating loads or burst loads. Thus, the energy-saving effect still has some limitations.

In this study, an effective energy-saving data layout for dynamic loads structure is developed which is called Adapted to Dynamic Load based on Redundant Array Independent Disks (ADL-RAID). ADL-RAID can be a good solution for high-intensity fluctuations or the sudden load problem, and cleverly avoid the problem that S-RAID [16] only applies to a smooth load. Because the data layout is dynamically adjusted according to the load, ADL-RAID only needs to use one or several data disks when the load is in a stable period. It can also open the corresponding disk on demand according to the load size when a sudden load arrives. It is a good solution for the problem of excess performance, thus, ADL-RAID can not only ensure that most of the disk keeps long-time standby, but also provides the appropriate local parallelism. Thus, it has higher availability and energy efficiency than S-RAID.

2. IMPLEMENTATION OF ADL-RAID

2.1 Basic data layout

We suppose that the disk array consists of N disks, each divided each disk into N storage areas. The same offset forms a stripe in the disk storage area, which comprises a total of N stripes. Each stripe contains a parity area and an $N-1$ data area. The parity area in stripe r is denoted by $p(r)$ and is located on disk $N-1-r$. The data disks are denoted $D(r, c$, where r represents the stripe number, c represents the disk number, and $0 \leq r < N$, $0 \leq C < N-1$). Now, we consider ADL-RAID composed of five disks as an example. The overall structure of ADL-RAID is shown in Figure 1 below.

The storage area continues to be subdivided, and each storage area is divided into M equal size band we name it Band, the Band in the parity area is called *PBand*, each stripe in the same offset of the Band form a Bank, $PBand(v)$ is generated by N-1 $band(v)$ using exclusive OR operation in this stripe. This is expressed by the following equation.

$$PBand(v) = Band(v_1) \oplus ..... \oplus Band(v_{N-1}) \qquad (1)$$

To achieve good parallelism and satisfy the performance requirements, on the basis of in Figure1 RAID continues to be subdivided in accordance with the ADL, as shown in Figure 2.

2.2 Dynamic mapping of storage space

For local parallel data layout, if the static address mapping is used, local parallelism must be set according to the performance requirements of the maximum value for stable load. In order to meet the performance requirements, and more disks will be dispatched to the running state, thus energy consumption will be wasted. So for the strong fluctuations and the sudden load situation, we need to schedule the size of the disk dynamically, so that the written data of the upper application can be distributed to a different number of disks to meet the dynamic load requirements.

The ADL-RAID should have the following conditions:

1. local parallelism, in order to achieve energy-saving;
2. the load distribution should ensure that the running time of the disk is long enough and the disk state transition frequency is low enough;
3. the local parallelism can be adjusted dynamically according to the performance requirements;
4. when the storage space is filled, the old data are deleted in chronological order and should not conflict with condition 3 when new data are written;

For condition 4), have a brief description: for a sequential data storage system, when the storage space is full, delete the oldest data by time generally, and then write the new data. we called the order of deletion. Video surveillance, CDP and other storage applications have this feature. The feature for order deletion will conflict with condition 3) generally. Such as deleting data (the earliest stored data) where the storage space has three disks parallelism, and the current load needs six parallel disks, we need to increase the three parallel disks, but the data on the candidate disk (local parallel) may not be deleted but even is used recently. In other words, the earliest stored data is located in the current three disks of local parallel. It can not increase the three disks to run on the condition of the order deletion

The specific process of the dynamic mapping algorithm for storage space is as follows.

The storage space is allocated as the k band according to the load performance parameter k (the number of disks required to be parallel). First, the function GetMaxBank () is executed, to select the bank having the largest free band in the current Stripe (*CurStripe*) as the current bank (named *CurBank*). If the number of free bands is 0 (*CurBank*.len=0), the current bank has no free bands to write data, and we need to further determine the number of free bands that can be mapped in the next stripe (*NextStripe*). Then, the next stripe will be recognized as the current stripe and reacquire the current bank. The next stripe will be moved back in order.

When the number of free bands is not 0 in the current bank or its number of free band is enough to meet the load requirements, GetBand () is execute, and k bands are required in order. Otherwise, we first remove all the free bands in the current bank, and then remove the remaining free band from the next bank, forming k free bands in all. Storage space will be recovered if the next bank does not have enough free band to carry out and re-acquire the free bank in the next bank. The function GetBand

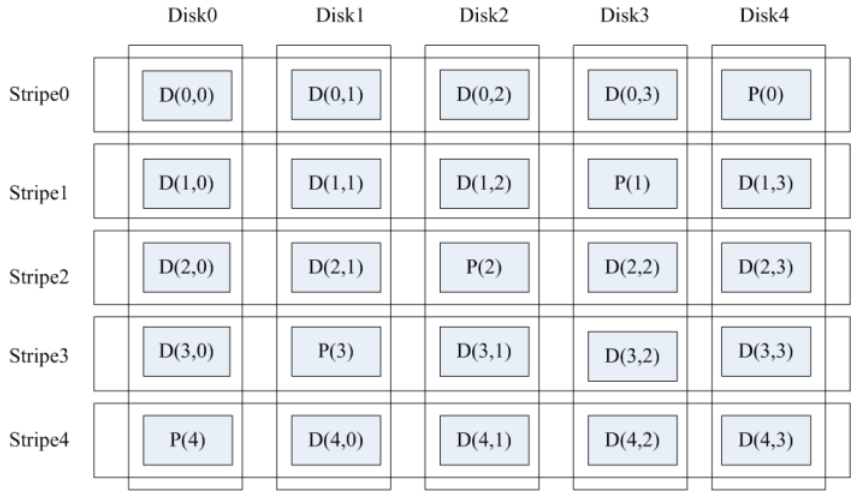

Figure 1 Overall structure of ADL composed of five disks.

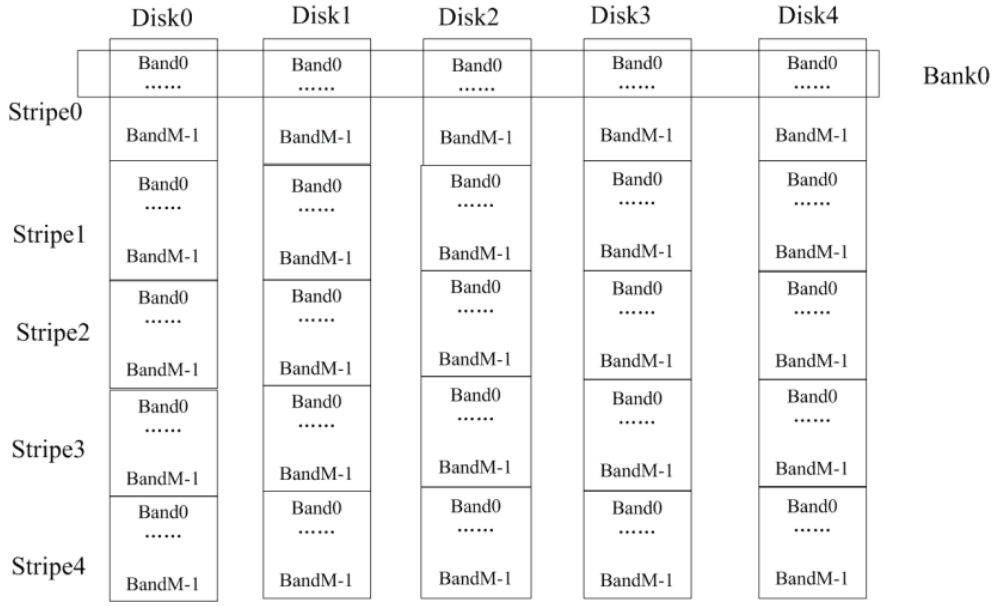

Figure 2 Five disk ADL-RAID subdivision structure.

() receives the specified number of free bands from the bank, and changes the number of free bands.

The execution of the data writing is described by Example 1

Example 1: Figure 3 shows the address mapping process. Nine kinds of loads-A to J-are written to the disks. The disk array is assumed to consist of five disks, as previously mentioned, and P represents the parity data. We select the first two stripe blocks as an example. To explain clearly, we assume that each stripe is divided into five banks, and the mapping granularity size is the band size. The numbers of the loads A to J requiring a parallel block are 2, 3, 4, 3, 3, 3, 4, 2, 1, 1, respectively, followed by numbers representing different time numbers (1-14), for example, C4 indicates that the load C is written to the disk at the 4^{th}time.

The above example, illustrates that when the load accesses the disk, the data are written to the current disk first. If the current active disk can satisfy the request, the standby disk will not be accessed, thus, not only good parallelism can be guaranteed, but also we can allocate the appropriate disks dynamically according to the load requirements.

2.3 Strategy for avoiding conflict

The strategy for avoiding the conflict when the same load is written to a different strip is illustrated by Example 2.

Example 2: From Example1, we see that there is an access violation problem in the storage space mapping process. When the bands from two stripes are parallel, they access the same disk simultaneously, which is called an access violation. As shown in Figure 4, when the load E is written to the disk at the 6^{th} time(E6), owing to the requirement of producing parity data, disks 2, 3, and 4 in stripe0 run at the same time, and disks 0 and 3 in stripe1 need to run at the same time. We see that disk 3 is accessed at the same time. Thus, the disk3 is a bottleneck for performance. The same problem exists with loads F8 and F9

Algorithm 1: AddMapping():

1. *CurBank*←*GetMaxBank*(*CurStripe*);
2. **if** *CurBank.len*=0 **then**
3. Get the free band of the next bank;
4. **end if**
5. **if** *NextBank*=0 **then**
6. *CollectBank*(*NextStripe*); //recycle storage space
7. **end if**
8. *CurStripe*←*NextStripe*;
9. *NextStripe*←*NextStripe.Next*;
10. *CurBank*←*GetMaxBank*(*CurStripe*);
11. **if** *CurBank.len*>= *k* **then** //Band within the CurBank is enough to map
12. *p*←*GetBand*(*CurBank,k*);
13. **else**
14. *p*1←*GetBand*(*CurBank*,*CurBank.len*); //Band is not enough to map
15. *NextBank*←GetMaxBank(*NextStripe*);
16. **if** *NextBank.len*<*k*-*CurBank.len* **then**
17. recover surplus space;
18. **end if**
19. *NextBank*←GetMaxBank(*NextStripe*);
20. *p*2←GetBand(*NextBand*,*k*-*CurBank.len*);
21. *p*←link(*p*1,*p*2); //Connect to Band
22. **end if**
23. AddMap(*p*);
24. **return** k;

The problem of access violation significantly affects the system performance. We must take appropriate measures to resolve the conflict. In this study, we found that the sufficient condition for avoiding the conflict is $K \leq N/2$. Of course, K is the number of bands that load needs to cross, and N is the total number of disks.

To eliminate the accessing conflict, we propose a strategy for avoiding conflict. The idea is expressed by algorithm 2.

Algorithm 2: AvoidClif() Strategy for avoiding conflict

1. *q*1←GetBand(*CurBank*,*CurBank.len*);
2. *NextBank*←GetMaxBank(*NextStripe*);
3. **if** *NextBank.len*<*k*+1-*CurBank.len* **then**
4. The free block of the current stripe is deleted;
5. *NextBank*←GetMaxBank(*NextStripe*);
6. *q*2←GetBand(*NextBank*, *k*+1-*CurBank.len*);
7. *q*←link(*q*1,*q*2); // Connect to *K*+1 Band
8. **end if**
9. *pos*←AccessConflict(*q*); // To determine the conflict
10. **if** *pos*=FFFF or *pos*=*k***then** // No conflict or end band has conflict
11. Delete(*p*,*k*)
12. **else**
13. Delete(*q*,*pos*) // the conflict band at pos
14. **end if**

When algorithm 1 is executed to line12, we must select k bands across two stripes to map. We first select K+1 bands (lines 1-7) and then check the access violation (line 8). If the current bank has no conflict or the end of the band has conflict, we delete the end of the band. Otherwise, we delete the conflict band according to the location parameter position. Finally, we can find k parallel bands such that there is no conflict

Figure 5 shows the spatial distribution of the avoidance strategy, where "/" represents the band that do not participate in address mapping.

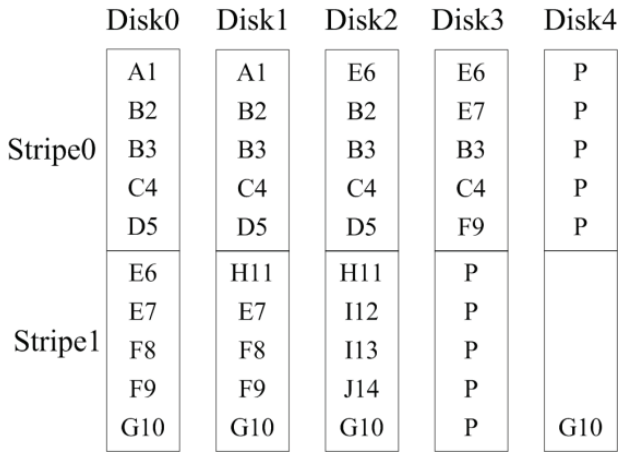

Figure 3 Example of storage space dynamic mapping.

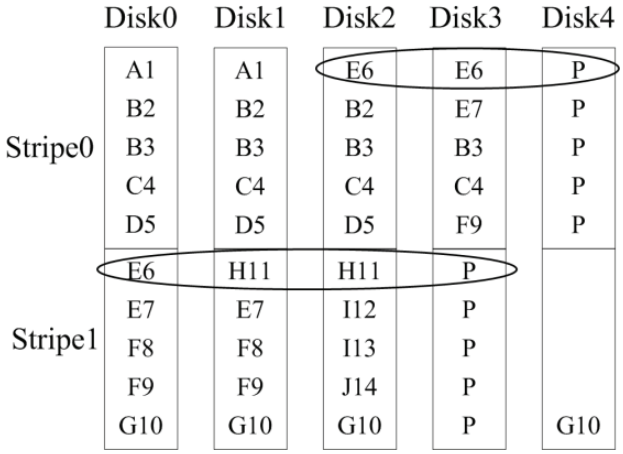

Figure 4 Problem of access violation in Example 2.

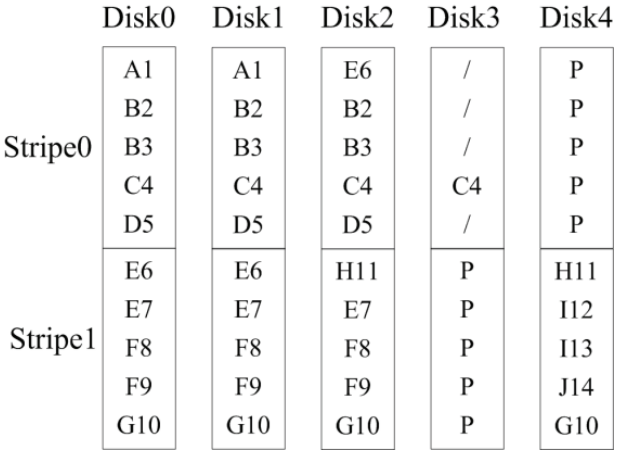

Figure 5 Storage space mapping after the avoidance policy is implemented.

We continue to explain the previous example in detail. When load E is executed to the 6^{th} time, it needs to open disks 2, 3, and 4, in stripe0, and we open disks 0, 3, and 4, when we write to the next stripe. Disk3 is accessed at the same moment. Thus, an access violation is generated. After implementing the avoidance strategy according to algorithm 2, load E on the distribution of disk 3 moves onto the next stripe and thus, does not participate in mapping, which is a very good solution for avoiding the conflict.

2.4 Space-Saving Method (SSM)

The strategy for avoiding conflict has shortcomings. From Example2, we see that a small amount of storage space is wasted after the avoidance strategy is used. Thus, we also propose a space–optimization strategy called the SSM for recycling wasted space. The details of the idea are expressed by algorithm 3. Algorithm 3 defines a structure *CurNum* and, a variable *Curband* on behalf of the remaining number of bands in the current stripe. The variable *Curstripe* represents the current stripe number, with initial value of 0.

Algorithm 3: SSM

/* * *curnum*: The current disk number, the initial value is stripe0. **curspt*: A pointer to the current disk, initially pointing to stripe0. */

1. **while** (*curspt->next*<>null) **do**
2. *curspt->Curstripe*++; // Add the stripe number to the first column of the table
3. *curspt->Curband* = *CurBand.len*; // remaining band number
4. *curspt->next* = *NextStripe*;
5. **end while**
6. **sort** (Curnum); //remaining band number for the first priority, stripe number for the second best, Prior to the Stripe sort;
7. **while** (*k*>0) **do**
8. **if** (findband (k,*CurNum*)>=0) **Then**
9. *curnum.Curband*=*curnum.Curband-k*;// Update remaining band number
10. **sort** (CurNum);
11. **end if**
12. **end while**

We continue to discuss the above example, assuming that each stripe is divided into five banks, the remaining band quantity is divided into four cases: one , two, three, and four bands. Thus, we can create a table to express the relationship between the remaining bands and the number of stripes, as shown in Table 1.

Table 1 Relationship between remaining bands and number of stripes

Stripe number	Number of remaining
0	1
1,4	2
2	3
3	4

The list presents the quantity of remaining bands in each bank, sorted from low to high. If the load needs to open two disks to satisfy the performance requirements, we first look for the remaining band quantity of 2, and check the stripe number. When the remaining band quantity is 2, the stripe number is 1 or 4. This means that the disks1 and 4 both have two free bands, because we regard the number of remaining bands for the first priority, that's when the load comes, the corresponding data are written to disk1 in order first, after which the remaining band is re-sorted. Thus, we make full use of the space that has been deleted. When we write the data full of disk each time, we can delete the earliest data in order. As above mentioned (deleted in order). Thus, when the load arrives each time, we first find the table, in order to confirm where the free band is and then locate the physical address on the disk. Finally, the data are written in the free band. The performance is greatly improved because of the recovery of space, and the storage space is fully utilized.

3. EXPERIMENTAL RESULTS AND ANALYSIS

3.1 Performance comparison

To test the energy consumption, we built ADL-RAID and S-RAID5 for comparison under Linux Kernel 3.1. We set the stripe size as 256k. We open one disk when the load size is between 0k and 256k, two disks when the load size is between 256k and 512k, and three or more disks when the load size is between 512k and 1024k. The hard disk and server parameters used in the experiment are presented in Tables 2 and 3.

Table 2 Disk parameters.

Description	Value
Model	WD10EZEX
Interface	STAT3.0
Rotational speed	7200rpm
Size	1TB
Active Power	6.8W
Idle Power	6.1W
Standby Power	1.2W
Spin up time	1.5s

Table 3 Server parameters.

Description	Value
Model	Power Edge R730
CPU	Xeon E5-2603 v3
Interface	SAS/STAT3.0
Memory type	RDIMM
Memory Size	16GB

Iometer is the most widely used tool for measuring and testing I/O subsystems. In the experiment, we used the Iometer tool to generate different write requests. The test results of the system regarding the performance and response time are shown in Figures 6 and 7.

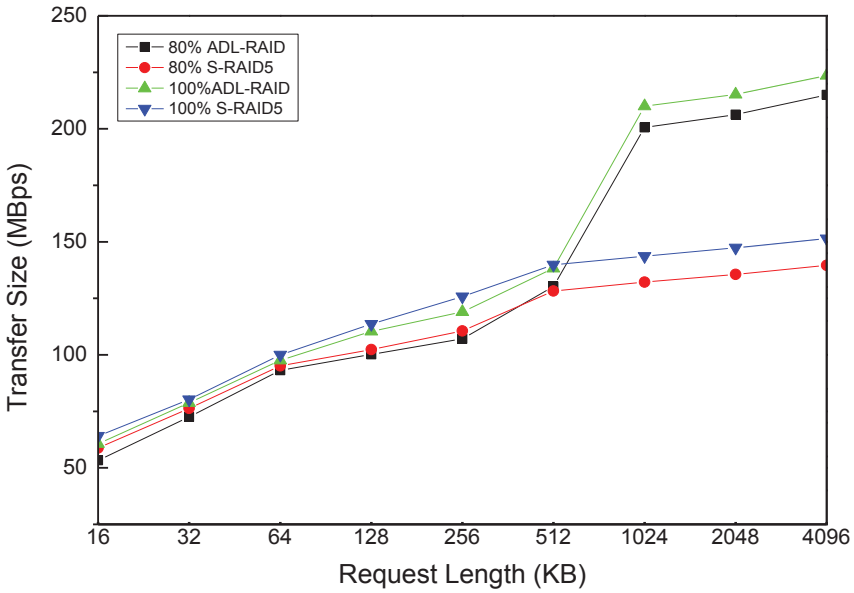

Figure 6 The comparison of the transfer size between ADL-RAID and S-RAID5.

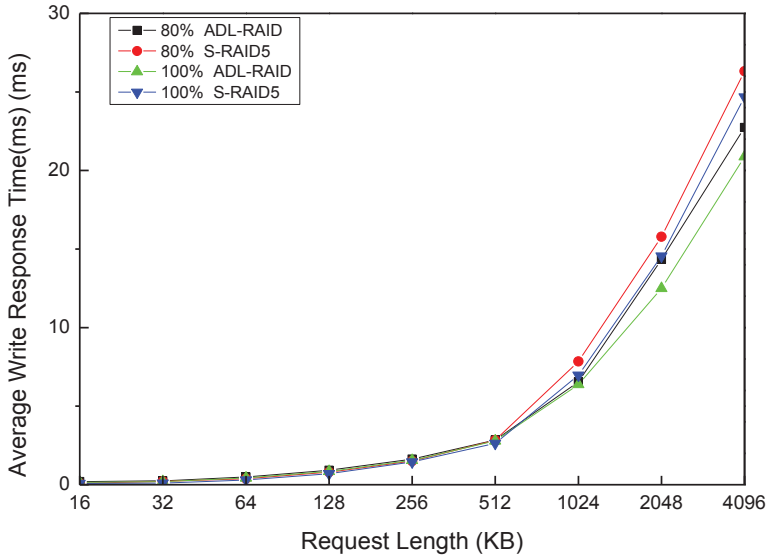

Figure 7 The comparison of the response time between ADL-RAID and S-RAID5.

Figure 6 shows the ADL-RAID and S-RAID5 transmission rates of the contrast for different load requirements, under the condition of 100% and 80% continuous write request. Because ADL-RAID is proposed for dynamic load, and the disk is divided into different areas. When the load size is less than 256K, the write performance of S-RAID5 is superior to that of ADL-RAID. The reason is that the ADL-RAID only needs to open a disk to meet the performance requirements at this time, and S-RAID5 need to open two parallel disks to execute tasks. So the overall performance of S-RAID5 is better than ADL-RAID. Until the load size is between 256k and 512k, in this case ADL-RAID also began to open two parallel disks to perform tasks, so the data of transfer size has no big difference compared to S-RAID5. From the experimental data we can also see that when the load size is 512k, the data transfer rate of ADL-RAID is 138.27MB/s, the data transfer rate of S-RAID5 is 139.86MB/s, the data is almost flat. However, when the load size is from 512k to 4096K, ADL-RAID opens three disks. Thus, the write performance is significantly increased. When the load is 4M, two system structures have the greatest performance. Under the condition of 80% continuous write request, the average transfer rate data for ADL-RAID and S-RAID5 are 220.05MB/S and145.62MB/S, respectively. Hence, the write performance is improved by 33.8%. For 100% continuous write request the average transfer rate data for two structures are 230.56MB/S and 151.43MB/S, respectively. Therefore, the write performance is improved by 34.3% which verifies that ADL-RAID can adapt to dynamic load characteristics.

Figure 7 shows the change in the response time with the load changes. When the load size is between 512K and 4096k, the average write response time for ADL-RAID and S-RAID5 is 10.51ms and 12.21ms. So the write response time of ADL-RAID is far shorter than that of S-RAID5, Thus, for the response time, ADL-RAID has significant advantages compared with the

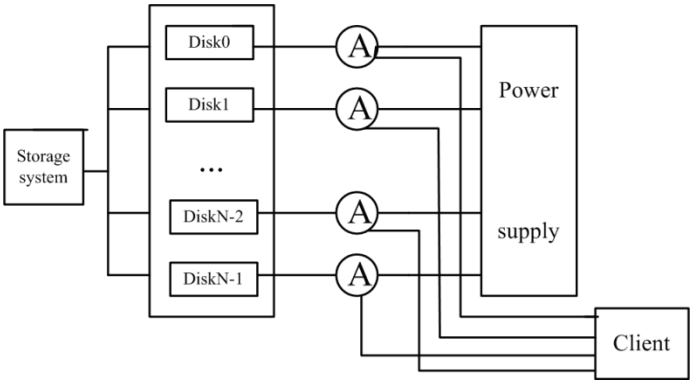

Figure 8 The test structure of energy consumption.

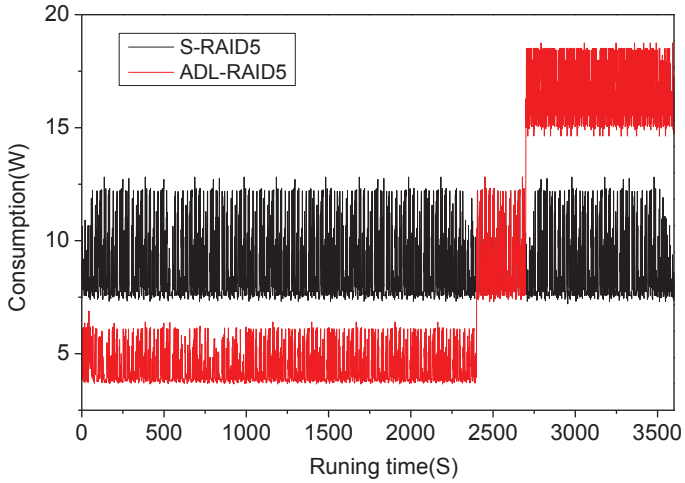

Figure 9 The comparison of the energy consumption of S-RAID5 and ADL-RAID.

previous S-RAID5.

3.2 Comparison of energy consumption

Follow the energy consumption structure of S-RAID5, we add ammeter between power supply and disks, formed the energy consumption test structure of ADL-RAID. As shown in Figure 8.

The voltage for the disk array storage media is provided by power supply (5v/12v). The value of electric current is used to record by ammeter between power supply and disks, The server side samples the current value at regular intervals.

In the experiment, we calculate the total energy consumption of the disk using the following formula(2):

$$W = \sum_{i=0}^{n-1} Vi \times Ii \tag{2}$$

Where, "n" represents the number of disks, "W" represents the total energy consumption of the disk array, "V_i" represents the real-time voltage of the disk, "I_i" represents the real-time current value, and "i" represents the number of disks in the array.

To obtain stable experimental data, we let the storage system run for 0.5 h first. In the test process, the video data stream (64 road, 128 road, 256 road of D1 format) was simulated respectively by C language data generator, to produce 256MB, 512MB, and 1024MB data of traffic. We tested 1 h of energy consumption. The results of the test are shown in Figure 9.

As shown in Figure 9, at the beginning of the system operation, because S-RAID5 is a fixed group structure, its energy consumption is flat. But as far as ADL-RAID concerned, ADL-RAID open the disk according to dynamic load. When the load less than 256k, open one disk, in this case the energy consumption is lower than the S-RAID5. With the load size increasing, ADL-RAID can not meet the load requirements when open one disk, so it will open two disks at this time to meet the load's requirement. During this time the energy consumption about two storage structures are almost the same. We can see it clearly in figure9. When the bigger load comes, ADL-RAID needs to open three disks to meet the load requirements. Therefore, the energy consumption will increase obviously. The average energy consumption of S-RAID5 is 9.528W. Compared with S-RAID5, ADL-RAID has an average energy consumption of 6.325W, which is 33.6% lower. Thus, the energy consumption exhibits an obvious decline in the condition where the write performance is improved and the write response time is reduced.

4. CONCLUSION

To realize energy-saving in continuous data storage applications, such as video surveillance, CDP and VTL, we propose an energy-saving data layout for dynamic loads called, ADL-RAID, that is proposed based on S-RAID [17], which is a static local parallel data layout. According to the perceived load performance requirements, ADL-RAID uses an address mapping mechanism to allocate storage space dynamically with appropriate parallelism for the load. Finally proved by experiment, in condition of 80% and 100% continuous write request, compared with S-RAID5 the write performance improve 33.8% and 34.3% respectively, energy consumption saves 33.6%. So ADL-RAID can adapt to high-intensity fluctuations in the load and burst load, with higher energy efficiency and availability than S-RAID.

Acknowledgement

The work was supported in part by the National Natural Science Foundation of China under Grant 61673159, in part by the Natural Science Foundation of Hebei Province under Grant F2016202145, and in part by the Science and the Technology Project of Hebei Province under Grant 15210325.

REFERENCES

1. Meng Xiaofeng, Ci Xiang. Big data management: concepts, technologies and challenges. Journal of Computer Research and Development, 2013, 50(01):146-169
2. Chen Shimin. Big data analysis and data velocity. Journal of Computer Research and Development, 2015, 52(02): 333-342
3. Li Yongkun, Shen Biaobiao, Pan Yubiao, Xu Yinlong, Li Zhipeng. Workload-Aware Elastic Striping With Hot Data Identification for SSD RAID Array. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2017, 36(5): 815-828
4. Sun Zhizhuo, Zhang Quanxin, Tan Yuan, Li Yuanzhang. Ripple-RAID: A high - performance disk array for sequential data storage. Journal of Software, 2015, 26(07): 1824-1839
5. Colarelli D, Grunwald D. Massive Arrays of Idle Disks for Storage Archives. Proceeding of the 2002 ACM/IEEE Conference on Supercomputing, 2012, 1-11
6. Xie Tao. SEA: A String-Based Energy-Aware Strategy for Data Placement in RAID-Structured Storage Systems. IEEE Transactions on Computers, 2008,57(6):748-761
7. Chen Youxu, Xu Yinlong, Li Yongkun, Xu Jun. Balanced Parity Update Algorithm with Queueing Length Awareness for RAID Arrays. 2016 IEEE 22^{nd} International on Parallel and Distributed Systems, 2016, 818-825
8. Sun Dongdong, Xu Yinlong, Li Yongkun, Wu Si, Tian Chengjin. Efficient Parity Update for Scaling RAID-like Storage Systems. 2016 IEEE International Conference on Networking, Architecture and Storage, 2016, 1-10
9. Zhan Ling, Men Yong, Tang Chenlei, Xu Peng, Wan Jiguang. SHGA: Design and Implementation of Two - level Cache Algorithm Based on. Small microcomputer system, 2017, (05): 1152-05
10. Cai Jieming, Fang Pei, Jia Siyi, Dong Huanqing, Liu Zhenjun, Liu Guoliang. Research on Hybrid RAID System with Multiple Strip Layout. Small microcomputer system, 2017, (05): 1143-09
11. Zhang Wenhui, Cao Qiang. A parallel codec algorithm based on XOR based RAID6 code. Computer research and development, 2015, 52(S2): 90-95
12. Dong Yong-Feng, Liu Jing-Yu, Yan Jie, Liu Hong-Pu, Wu You-Xi. HS-RAID2: Optimizing Small Write Performance in HS-RAID. Journal of Electrical and Computer Engineering, 2016
13. Liu Jingyu, Tan Yuan, Xue Jingfeng, Ma Zhongmei, Zheng Jun, Tan Yuan. S-RAID5: An energy-efficient disk array for data access. Journal of Computer Science, 2013, (06): 1290-1302
14. Li Xiao, Tan Yu-An, Sun Zhi-Zhuo. Semi-RAID: A reliable energy-aware RAID data layout for data access. 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies, 2011, 1-11
15. Li Yuanzhang, Sun Zhizhuo, Ma Zhongmei, Zheng Jun, Tan Yuan. S-RAID5: An energy-efficient disk array for data access. Journal of Computer Science, 2013, (06): 1290-1302
16. Liu Jingyu, Zheng Jun, Li Yuanzhang, Sun Zhizhuo, Wang Wenming, Tan Yuan. Mix S-RAID: An energy-efficient data layout for sequential data storage. Computer Research and Development, 2013, (01): 37-48
17. Li Xiao, Tan Yu-an, Sun Zhizhuo. Semi-RAID: A reliable energy-aware RAID data layout for sequential data access. Proceeding of IEEE Symposium on MASS Storage Systems and Technologies. IEEE Computer Society, 2011: 1-11