Journal of The Magnetics Society of Japan Vol.21, Supplement, No.S2 (1997)
© 1997 by The Magnetics Society of Japan

Flexible Architecture of Disk Array*

Hai JIN **Jiangling ZHANG**

Department of Computer
Huazhong University of Science and Technology
Wuhan, 430074 P.R.China

Abstract - Redundant Array of Independent Disks (RAID) is the most efficient way to avoid the bottleneck problem between I/O subsystem's performance and the CPU processing power. This paper describes a novel RAID architecture, called FlexArch, which can meet different requirements in different application environments. It is very convenient for the different users to define and construct a kind of RAID architecture according to its application environment. Two kinds of special structures, RAID_table and RAID_map are proposed in order to implement FlexArch. Based on these two structures, the scheduling algorithms for the read/write operations on FlexArch RAID are also given.

Key words: Recording System Architecture, RAID, Performance Evaluation, Data Structure, Scheduling Algorithm

I. INTRODUCTION

In the past few years, the performance of processors has been growing steadily, and with multiprocessor organizations, the processing power of a computer system has been increasing at a rapid pace. However, the I/O performance has not kept pace with the gains in processing power due to the limitation of the mechanical delays. Thus, there exists a huge disparity between the I/O subsystem performance and the processing power. Therefore it is essential that one kind of improvements be made in I/O performance to avoid the bottleneck. In order to reduce this disparity, many methods have been proposed. Redundant Array of Independent Disks(RAID) [11] [12] is the most efficient way. RAID connects several disks together to extend the cost, power and space advantages of small disks to higher capacity configurations, and it also provides partial redundancy such as parity, availability can be increased as well. There are two obvious advantages of using RAID[4]: reconstructing data to the new disk driver to ensure the availability and reliability of data, non-stop running of host even in the occurrence of system failure. Due to all these reasons, RAIDs are strong candidates for nearly all on-line secondary storage systems[1].

Many studies have been made on the architecture of RAID in order to improve its performance. But there is not an uniform architecture to fit all the requirements for different application environments. This paper describes a novel RAID architecture which can meet different requirements in different application environments. It is very convenient for the different users to define and construct a kind of RAID architecture according to its application environment. As it is very flexible to define and construct, we call it FlexArch.

II. RELATED WORK OVERVIEW

According to the different fault tolerant methods and data distributed methods, there are six different RAID architecture. They are:

RAID Level 0: only distributing data among disk driver using striping. It can achieve highest disk I/O performance, easy to manage and without extra write overhead. But because of without fault tolerant, risk of losing data is greatly increased.

RAID Level 1: disk mirroring with striping. The difference of RAID level 1 and disk mirroring is that disk mirroring is only mirrored in the file level and RAID level 1 can be mirrored at the sector level. Each driver among the array has independent channel with host, using synchronous and asynchronous method. Data striping improves the I/O performance, enhances the reliability and supports parallel read. The greatest disadvantage of using RAID level 1 is that it should use double storage space and have the highest cost.

RAID level 2: using Hamming ECC code and interleaving at bit level. Check bit is calculated using Hamming code and each driver has independent data channel, the rotation of spindle uses synchronous and asynchronous method. RAID level 2 is suitable for application with large sequential I/O requests, the disk utilization is much higher than that of RAID level 1. The problem of using RAID level 2 is that small data access performance is slow down because each time of data read/write should access each driver among array, thus it is not suitable to the application of transaction processing. In the case of small scale of disk array, the usable storage space is greatly reduced. As RAID level 2 does not utilize the inherent error correcting and detecting ability of disk driver, it is seldom used. But it gives an enlightenment of using the other ECC code to ensure even in the case of two drivers failure, the system can still be run without stopping.

* This paper is supported by National Natural Science Foundation of China under grant 69503003

RAID Level 3: using bit interleaving and parallel transferring. Check information is produced by parity check, thus reduces the number of check driver. Usually it uses one dedicate driver for fault tolerant. Parity calculation is carried out by using exclusive OR operation. The rotation of spindle uses synchronous method. It is suitable for the application with higher data transfer rate and small I/O throughput. It has the higher disk utilization and the higher read performance than RAID level 2. It is easy to extend and construct. As it takes lots of time to calculate parity information, it slow down the write performance. And as RAID level 3 can only process one task at one time, it is I/O limited. It is suitable to the application with large I/O requests and not suitable to the transaction processing.

RAID Level 4: independent transfer with block interleaved. The size of block is at least one sector. It also uses one dedicated parity disk, and can be accessed in parallel. For the small size of data, it has much more performance than RAID level 3. It uses a new method to calculate parity, that is it uses only old data, new data and old parity to calculate new parity. Thus, it only access the referred data drivers and parity driver, not all the data drivers. As it should access parity disk all the time, parity disk becomes the bottleneck of whole system. This is the great disadvantage of RAID level 4. Anyhow it still reduces total data transfer time because of large block of data transfer. The advantages of RAID level 4 are that it calculates parity data easily, can realize parallel I/O of all the devices in one group, higher capacity utilization, easy to construct and extend the scale. But it has the worse write performance and restricted to one write operation at one time.

RAID Level 5: independent transfer with rotated parity distribution. The placement of parity information in the disk array has many different ways. The most common ways are left symmetric, right symmetric, left asymmetric, right asymmetric, extended left symmetric, flat left symmetric[7]. In the case of low workload, extended left symmetric and flat symmetric have better performance for the read operation, RAID level 4 has better performance for the write operation. In the case of high workload, RAID level 5 have better performance, because in this case the uniform utilization of each disk in the array is much more important. Thus the strategy of load balance is the key factor to archive the higher performance. The rotation of spindle in RAID level 5 is asynchronous. The advantages of RAID level 5 are that higher data transfer rate, parallel read/write, higher disk utilization, multiple placement strategy of parity information. While RAID lever 5 disk arrays offer performance and reliability advantages for a wide variety of applications, they are commonly thought to posses at least one critical limitation: their throughput is penalized by reading out of the old user's data, overwriting this with new user's data, prereading the old value of the corresponding parity, then overwriting this disk block with the updated parity. In contract, mirrored disks simply write the user's data on two separate disks, and therefor, are only penalized by a factor of two. This disparity, four accesses per small write instead of two, has been termed the small write problem.

Many RAID structures are proposed to improve the on-line performance of disk array system. Sparing is one kind of architecture to reduce the repair time in the case of disk failure in the array. It uses spare disk or hot spare to reconstruct data of the failure disk to the spare disk to reduce the possibility of losing data. There are three kinds of sparing structures, dedicated sparing, distributed sparing, and parity sparing. The performance of distributed sparing and parity sparing can be improved 10% and 18% than dedicated sparing, respectively[9]. For distributed sparing and parity sparing, they have no more difference in the normal mode. But in the degrade mode and reconstruction mode, distributed sparing has better performance than parity sparing.

Parity striping structure [2]stripes only parity information. It is based on the fact that mirrored disk has higher availability and higher throughput. In most cases, there is no need to stripe data. Small random request can archive higher transfer bandwidth without striping large data into small pieces. The disk utilization of this structure is similar to that of mirrored disk.

Floating data/parity structure[10] uses method of shorten "read-modify-write" time when modify data or parity information so as to solve small write problem. The method is to let all the data block and parity block correspond to the different cylinder separately. In each cylinder, one track is preserved to keep the modification result of data block and parity block. This method can reduce response time of small write by the floating address physical space on the disk. But it is not so convenient to the large size request. That is because after many times of small write many logically continuous block is physically separated. Thus it enlarges the rotation delay of accessing logically continuous data. Moreover, after many times of small write, modify, all the free space on the disk becomes non continuously. In this case, large data is written to the physically non continuous space.

Parity logging[13] is an efficient way to solve the small write problem nowadays. It sets large capacity of cache or buffer to combine some small write to a large write to improve the data transfer rate and reduce the time of modifying parity information. It stores all the modification of parity information as a log in the logging cache. When the logging cache is fulfilled after many times of writing, it writes these parity information in large block to the parity log disk in serial way. Parity log disk can be implemented by flash memory or SSD (solid state disk). When the parity log disk is fulfilled, then it reads all the information in parity log disk and the information

of parity disk or data disk to the controller in order to reconstruct parity information, and then writes the information to the corresponding disks in the array to empty the parity log disk. One of the disadvantages of parity logging is that when parity log disk is fulfilled, the I/O request should be blocked so as to reconstruct parity information, and these operations should be carried on in front end way. This is a very serious problem to the application with quick response or with frequent small write request. Besides, it can only reduce the parity block accessing time and has no influence to the data block.

III. FLEXIBLE ARCHITECTURE FOR RAID

From the discussion above can we see that although there are many structures proposed to solve some problems corresponding to some applications. Up to now there is no uniform and efficient structure to suitable for the requirements of different applications. FlexArch[5] is one disk array structure suitable for the different requirements of different applications. It is easy to define and implement.

The main purpose of FlexArch disk array is to combine different kinds of optimal disk array structures in different application environments together in the disk array and use the method similar to the partition in operating system. As we know, in operating system, the most common method to manage the memory or storage system is partition. In storage system management policy, partition is done by divided the storage space into different small regions, the length of each region is variable. The partition can be done by schedule policy, or by the system users. Based on this point, we can view the entire disk array as a whole storage space, and divide this space into several parts, called region. Each of the region can be viewed as a small disk array with one kind of architecture which is most suitable to one kind of application. Each region is divided into several blocks, the block size is variable according to the number of outstanding request at any given time[3]. Thus, the entire disk array will be a combination of several kinds of architecture. Figure 1 shows one disk array example with FlexArch structure. In this example, there are four regions, the structure of each region is flat left symmetric, left symmetric, parity logging and right symmetric, respectively.

IV. DATA STRUCTURE AND SCHEDULING ALGORITHM

In order to implement FlexArch structure, two special data structures should be used, RAID_table and RAID_map. The definition of RAID_table is as follow:

```
structure RAID_table {
        unsigned int Region_No;
        unsigned int Region_Size;
        unsigned int Starting_Address_of_Region;
        unsigned int Number_of_Block_in_Region;
        unsigned int Block_Size_of_Region;
        unsigned int Architecture_Type_of_Region;
} RAID_table;
```

where Region_Size = Number_of_Block_in_Region × Block_Size_of_Region

and Architecture_Type_of_Region can be defined using predefined specification.

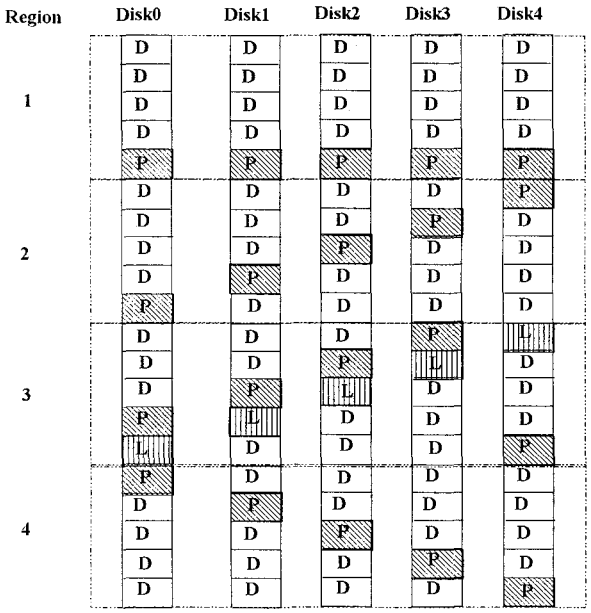


Figure 1 Example of RAID with FlexArch structure

RAID_map is similar to the disk map. It is a reserved region in the non violate memory, each row corresponds to the horizontal block in the disk array, and each column corresponds to one disk in the array. Each byte in the map illustrates the usage of the corresponding block in the disk array. There are six different usages of each block, they are unused block, data block, parity block, log block, spare block and failure block, respectively. Figure 2 is the example of RAID_map of the disk array in figure 1.

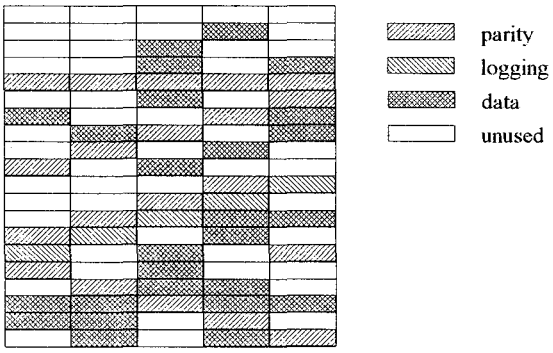


Figure 2 Example of RAID_map of RAID in figure 1

Following shows the scheduling algorithm of writing data to the disk array with FlexArch.

Step 1: accept write request from host;
Step 2: analysis command from host;
Step 3: data striping;
Step 4: calculate parity;
Step 5: convert disk array logical address to the disk physical address;
Step 6: start disk array write driver;
Step 7: write striped data or parity information to each disk;
Step 8: write finished, go to Step 9; otherwise go to Step 7;
Step 9: modify RAID_map in the disk array controller.

Following shows the scheduling algorithm of reading data from the disk array with FlexArch.

Step 1: accept read request from host;
Step 2: analysis command from host;
Step 3: convert disk array logical address to the disk physical address;
Step 4: start disk array read driver;
Step 5: read striped data or parity information from disks in the array;
Step 6: read finished, go to Step 7; otherwise go to Step 5;
Step 7: exist failure disk. go to Step 8 ; otherwise go to Step 11;
Step 8: reconstruct failure data using redundant information;
Step 9: exist uncorrectable failure, halt; otherwise go to Step 10;
Step 10: modify corresponding cell element in RAID_map;
Step 11: construct integrated data;
Step 12: send data to host.

V. CONCLUSION

In this paper, we propose a novel kind of flexible RAID architecture, FlexArch. It can meet different requirements in different application environments. It is very convenient for the different users to define and construct. It is a combination of different kinds of the optimal architecture in different application environments. In this architecture, we use the method of partition as in the operating system. We also introduce two kinds of special structures. RAID_table and RAID_map. Based on these two structures, we give the scheduling algorithms for the I/O operation to FlexArch RAID. The opening problem of the FlexArch is the architecture reconstruction of each region. It is necessary to reconstruct the region architecture when the application environment is changed. If the application is no more needed, the data in the correspondent region should be backup to the auxiliary storage system and the architecture of this region should be reconstructed. If the data in the auxiliary storage system are needed to use in RAID, the data should be restored from the auxiliary storage system and the architecture of one region in which all the blocks are unused should be reconstructed. Even this, FlexArch architecture is still the flexible way to meet all the requirements for different users of RAID.

REFERENCE

[1] P.Cao, S.B.Lim, S. Venkataraman and John Wilkes. The TickerTAIP Parallel RAID Architecture. Proceedings of the 20th Annual International Symposium on Computer Architecture, San Diego, CA, May, 1993, pp.52-63

[2] P.M.Chen, E.K.Lee, G.A.Gibson, R.H.Katz, D.A.Patterson. RAID: High-Performance, Reliable Secondary Storage. ACM Computing Surveys, vol.26, No.2, June 1994, pp.145-185

[3] P.M.Chen, D.A.Patterson. Maximizing Performance in a Striped Disk Array. Computer Architecture News, vol.18, No.2, 1990, pp.322-331

[4] G.A.Gibson. Redundant Disk Array: Reliable, Parallel Secondary Storage. The MIT Press, 1992

[5] Hai Jin. Study of Disk Array with FlexArch Architecture and its On-line Performance Evaluation. Ph.D. Dissertation. Huazhong University of Science and Technology. November, 1994

[6] R.H.Katz, G.A.Gibson, D.A.Patterson. Disk System Architectures for High Performance Computing. Proceedings of IEEE, vo.77, No.12, December 1989, pp.1842-1858

[7] E.Lee, R. Katz. The Performance of Parity Placement in Disk Arrays. IEEE Transactions on Computers, vol.C-42, No.6, 1993, pp.651-664

[8] J.Menon, J. Cortney. The Architecture of a Fault-tolerant Cached RAID Controller. Proceedings of the 20th Annual International Symposium on Computer Architecture, San Diego, CA, May, 1993, pp.76-86

[9] J.Menon, R.Mattson. Distributed Sparing in Disk Arrays. COMPCON'92, pp.410-421

[10] J.Menon, J.Roche, J.Kason. Floating Parity and Data Disk Arrays. Journals of Parallel and Distributed Computing, January, 1993

[11] D.A.Patterson, G.Gibson, R.H.Katz. A Case for Redundant Arrays of Inexpensive Disks(RAID). Proceedings of SIGMOD,Chicago, IL, June 1988, pp.109-116

[12] D.A.Patterson, P.Chen, G.Gibson, R.H. Katz. Introduction to Redundant Arrays of Inexpensive Disks(RAID). Spring COMPCON'89, San Franciso, CA, pp.112-117.

[13] D.Stodolsky, G.Gibson, M.Holland. Parity Logging Overcoming the Small Write Problem in Redundant Disk Arrays. Proceedings of the 20th Annual International Symposium on Computer Architecture, San Diego, CA, May, 1993, pp.64-75