THE DESIGN AND ANALYSIS OF CAPACITY EXTENDIBLE DISK ARRAY SYSTEM: THE DIAGONAL MOVE ALGORITHM.

By

CHENG-YUAN TSENG

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1994

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1997

THE DESIGN AND ANALYSIS OF CAPACITY

EXTENDIBLE DISK ARRAY SYSTEM:

THE DIAGONAL MOVE ALGORITHM.

Thesis Approved:

[illegible]

Thesis Advisor

J Chandler

H. Lu

Thomas C. Collins

Dean of the Graduate College

PREFACE

I wish to express my special appreciation to my thesis advisor, Dr. Kayikkalthop M. George, for his timely supervising on my research work. I also would like to thank my thesis committee members Dr. John P. Chandler, Dr. George E. Hedrick, and Dr. Huizhu Lu, for their administrating of my final oral exam and assistance of my master degree study.

I give my sincere appreciation to my family who have supported my education and life. Without steady financial support from my mother - Yen-Hsiang Lin, I would not be able to finish this degree now.

Finally, I also say thanks to all my friends in Stillwater Oklahoma for their company with my study during many years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Bn	Stripe unit for backup disk
C	Single disk capacity
H	Number disks as Hamming Code disks
N	Number of disks in the disk array system
Pn	Parity stripe unit
Qn	Parity stripe unit in RAID level 6
Ph	Probability of hitting locked data
Pl	Probability of locking data
S	Number of stripes per disk
Sl	Locking data size
St	Total data size of disk systems
Su	User's total write request size
Ta	Average time for user access disk
Tar	Algorithm run time (AlgRunTime)
Te	Total execution time (ExecutionTime)
Th	The time duration that the user can access the disk without waiting
T_{level_x}	Operation for specific RAID level

Tm	Memory operation time
Tr	Stripe unit read time
Tur	User run time (UserRunTime)
Tuw	User wait time (UserWaitTime)
Tw	Time duration that the user is locked by the algorithm
Tw	Stripe unit write time

CHAPTER 1

INTRODUCTION

1.1 Background

With increased I/O performance and at least one disk failure tolerance, data redundant disk array as secondary storage system efficiently translate from a conventional computer storage system to be with better I/O rate, higher data transfer rate, and stronger reliability than traditional large single-disk systems.

The increased I/O performance in measurement of I/O operating rate and data transfer rate are mostly gained from simultaneous data retrieval from several disks which are organized in parallel as shown in Figure 2-1. This parallel disks organization lets us have better I/O operating rate than the singular disk architecture, since it has several disk-I/O operations running concurrently. The I/O operating rate is defined as the number of I/O operations per second. And the simultaneous data access from disks lets us have better data transfer rate than any singular disk architecture, since we are retrieving data from more than one disk at the same time. The data transfer rate is defined as amount of data transferred through the bus or network per second such as bits per second (BPS). The disk data access time is the composition of seek time, rotation time, and data transfer time. Because data transfer speed over the bus or network electronically is much faster than the slow mechanical disk drives, accessing several disks in parallel can contribute to the bus or network having better utilization.

The data redundancy schemes using mirroring or parity information contribute to at least one disk failure recovery in the data redundant disk array systems. Traditional single disk storage system has absolutely no fault tolerance. If it fails, all data on the disk were gone. Armed with data redundant schemes, the data redundant disk array systems have failure recovery ability for one disk failure. With more complicated data redundancy architecture, we can against more disk failures from happening at the same time. With such a superior performance and robust reliability, the RAID will likely be the next generation secondary storage system.

The Redundant Arrays of Inexpensive Disks (RAID) is based on the observation that the disk I/O bandwidth can be improved by using several inexpensive concurrent disks instead of a Single Large Expensive Disk (SLED) [Patt88]. Its seven-level (RAID level 0 through RAID level 6) organization defines differences on both redundancy for reliability and data striping for performance strategies [Lee93]. The RAID-II is derived from RAID concepts by adding a unique high performance controller to communicate in a high-bandwidth network [Lee93] which has even faster data transferring speed. There are some other related researches on the data redundant disk array such as TickerTAIP parallel RAID [Cao94], HP AutoRAID hierarchical storage system [Wilk96], Zebra network file system [Hart95], Large Scale DIY RAID [Asam96], and Network Of Workstations [Ande95]..,etc.

The TickerTAIP aggressively distributed the disk controllers across the storage system to avoid single point failure on the centralized controller which is used in most RAID models. The HP Hierarchical AutoRAID employs two level storage hierarchy: mirrored on the first level to compensate for the second level RAID-level-5 data retrieval speed, because the mirrored disk array has better read performance and the RAID level 5 has lower cost with acceptable performance. The Zebra striped network file system invents a new striping methodology, together with log-structured file system (LFS) [Rose91] for effectively reducing the overhead from small-write on disk array. Exploring the best cost/performance disk array system, the Large Scale DIY RAID is a disk array architecture ranging from ten to thousands off-the-shelf disks with price/performance SCSI-16 as disk interface. The combination of off-the-shelf disks and price/performance SCSI-16 interface lets us find a disk array model with sufficient performance at acceptable price. The term "price/performance" means the SCSI-16 has adequate performance at acceptable price while others have either a little better performance at high price or poor performance at a low price such as SCSI-32 and ISA disk interfaces [Asam96]. Although the goal of the Network of Workstations (NOW) [Ande95] is to make a distributed computers system to compete with Massive Parallel Processing (MPP) systems, it treats all disks on the connected workstations as serverless network disk array storage system [Ande95].

1.2 Motivation

Many publications have shown that a computer system with fault-tolerant/fail recovery parallel I/O architecture improves its performance and reliability at lower cost [Patt88]. The parallel I/O systems such as a disk array system provides high data bandwidth by allowing simultaneous disk data received from disks [Kim86]. The data redundant disk array designs on the secondary storage systems support reliability of stored data by implementing mirroring or parity information techniques in the disk systems[Lee93].

Having a capacity extendible ability on this disk array model will be able to let us either to create more available disk storage space for users or to equip more robust hot spare mechanism for fast failure recovery. Furthermore, it will also allow maximum data access parallelism available on the disk array systems for better disk storage performance. By increasing data parallelism and adding a new disk, we can project the new disk arrays to have better performance, more disk driver space, or reliability. For example in the Figure 1-2, we are increasing the degree of parallelism from four to five. The degree of parallelism is defined as maximum possible number of simultaneously running disk I/Os in a system. In our case, the greater degree of parallelism, the greater the performance. As we mentioned before, it increases performance because both the data transfer rate as well as the I/O operating rate in the disk array systems are improved.

Definition 1: A capacity extendible disk array is a disk array model in which we can enlarge maximum amount of data in the disk array system by adding new disks without changing the data layout scheme for any existing RAID level.

The capacity extendible disk array is a disk array model which can be extended in total available storage size by adding new disks without changing the data layout scheme for any RAID level as shown in the Figure 1-2. Although in the Figure 1-1, the disk array has increased in storage capacity, it doesn't have the same data layout as RAID level 5 should be. We will briefly introduce the seven RAID levels defined by University of California at Berkeley in Chapter 2. After adding one new disk on the disk array, we expect to improve read/write performance by increasing parallelism of the disk. We interpret the capacity as the maximum amount of data that a disk array can hold. And, it should be a fixed number of bytes. The term extendible denotes the disk array systems should be able to enlarge during the course of time. In this thesis, we discuss a Straight Forward algorithm and a Diagonal Move algorithm as the methodologies for extending disk array storage capacity. Any disk array system implemented with either Straight Forward algorithm or Diagonal Move algorithm could be a capacity extendible disk array model.

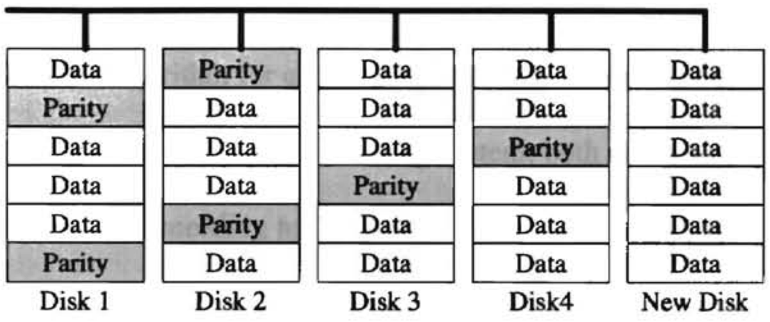


Figure 1-1 Adding a new disk to the RAID level 5 without rearranging data.

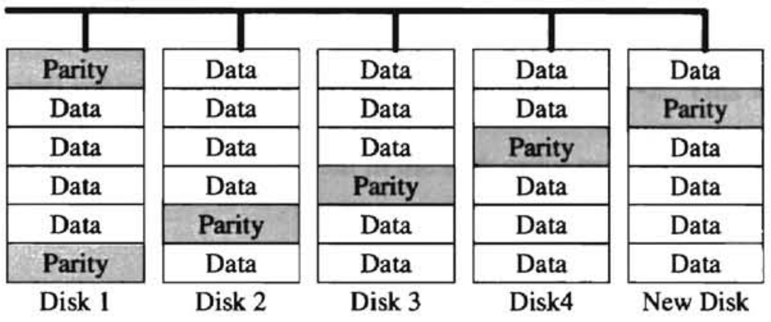


Figure 1-2 Adding a new disk to the RAID level 5 with rearranged data.

In this thesis, we propose the Diagonal Move algorithm model which can universally make disk arrays' size extendible with minor modification on different disk array levels and the Straight Forward algorithm model which can give users more accessibility on disks data. Comparing to the Straight Forward algorithm, the Diagonal Move algorithm is an improved algorithm with less operation time because of its parallel disk access method. We will define both the Diagonal Move algorithm and Straight Forward algorithm in Chapter three later. The simulation, performance analysis and comparison of different algorithms are discussed in Chapter four.

1.3 Problems

We believe the algorithm for capacity extendible disk array model is necessary, because after we add a new disk into an existing system, both data and its redundant information must be kept according to the layout scheme for the disk array system as defined in RAID [Lee93] for maximum performance. While we make the new disk be part of the system, we not only need to keep the redundant information updated with original RAID level schemes, but also distribute all data on all the disks as it would be in the original RAID level scheme as shown in Figure 1-1 and Figure 1-2. This will also ensure that we can maintain maximum parallelism in the disk array systems.

Limited and Fixed Storage Capacity:

The current disk array systems are fixed in the number of disks as well as storage capacity. Since future applications might take more storage space and data transfer bandwidth, the capacity extendible disk array is an ideal solution for increasing storage capacity as well as for improving data transfer bandwidth over the bus and network.

Bottleneck of New Disk Without Striping:

It is possible to add a new disk on the array without striping with others as in Figure 1-1, but the data will not be distributed to all disks even though capacity of the system increased. The data gathering in one disk will neither improve the performance nor the reliability, so the new data need to be striped as those already existing on the array as in Figure 1-2.

No Guaranteed Backup:

The idled single hot spare (backup) disk possibly will not guarantee system reliability if it failed before we need it as illustrated in Figure 1-3. The disk array systems has absolutely no failure recovery ability, if the installed hot spare disk is not able to function when we need it. The distributed hot spare is a more reliable protection mechanism for the system when there is a disk fails. In our proposed algorithm, we have an option to make the distributed hot spare embedded in the disk array system after adding a new disk in the system. If we could distribute the backup disk across the disk array systems as illustrated in Figure 1-4, a backup disk is guaranteed.

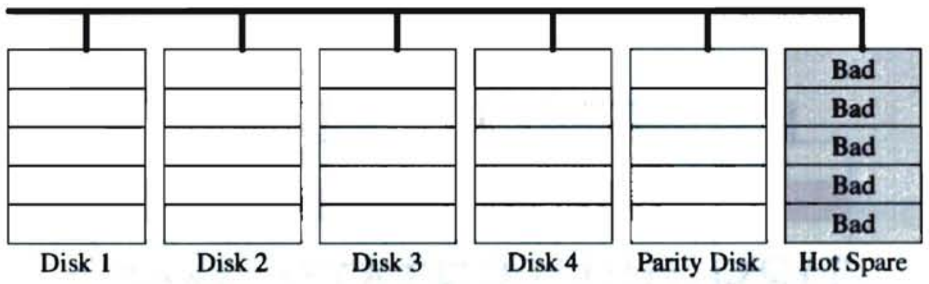


Figure 1-3 No recovery ability if the hot spare unit is not reliable when we need it.

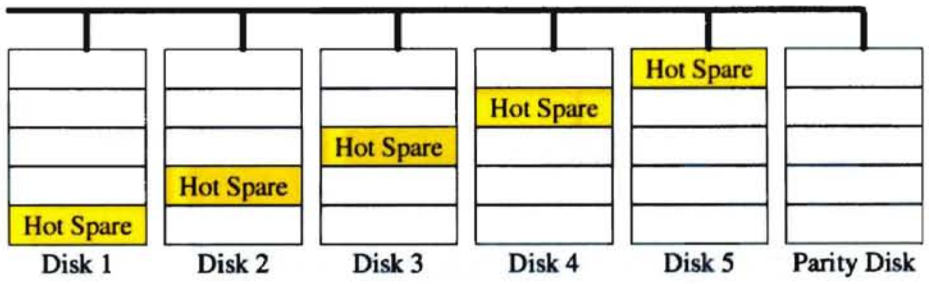


Figure 1-4 Reliable distributed hot spare disk.

Zebra Strategy Has Wasted Space:

On Zebra file system [Hart94], with special mechanism, the file manager will let the new data use all disks as a stripe, but those data which exist before we add a new disk will not be changed until they are deleted as in Figure 1-5 and Figure 1-6. That is Zebra file system can have two different stripe sizes in the storage system. Zebra file system uses a special structure - file manager and stripe cleaner - to let more than one stripe size exist in disk array, but the mechanism is not implemented on any other disk array yet. And, it is quite inefficient while we add a new disk after the previous system is already almost full with few future deletion (cleaning) on old stripe. This wasted space is equal to the number of previously existing stripe times the size of stripe unit. We need a disk array model and algorithm which has a little or no wasted space after upgrade.

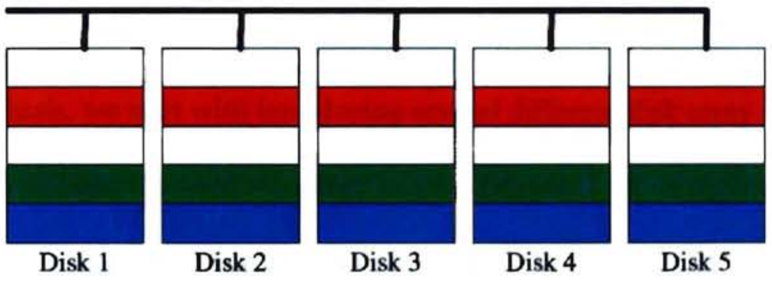


Figure 1-5 Before adding a new disk on a Zebra disk array.

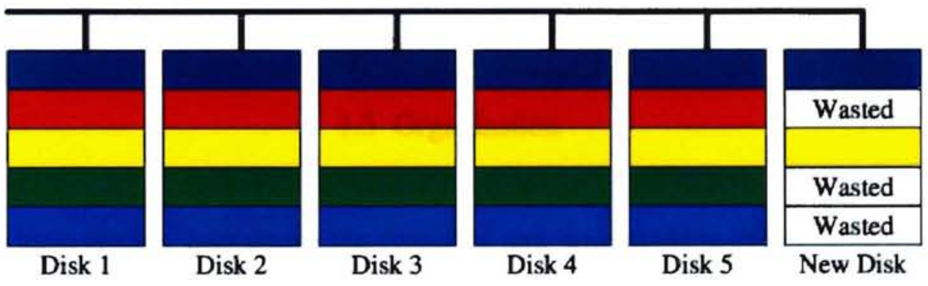


Figure 1-6 After adding a new disk on a Zebra disk array.

1.4 Thesis

In this thesis, I propose new disk array capacity extendible algorithms which no one has discussed yet for those existing disk array models. With our Diagonal Move algorithm, the disk arrays can easily add a disk for increasing system storage capacity and I/O system parallelism with possibly better failure recovery based on configuration. And we solve all the problems that we described above on Section 1.2. Comparing to the Straight Forward algorithm, theoretical and experimental analysis results show that the Diagonal Move algorithm will perform the same result in less time and in less number of moving operations, but the Straight Forward algorithm may have better disk array data availability for users.

In this thesis, we start with introducing several different disk array designs. We then present the Straight Forward algorithm and the Diagonal Move algorithm as capacity extendible algorithms for disk arrays. At last, we give a simulation and performance comparison of the two algorithms mentioned above.

1.5 Organization

We have already introduced the problems and limitations in the existing disk array systems and benefits of our capacity extendible disk array models in the Chapter one. Next, in Chapter two, we review some disk array terminology from publicly available literature. We will also describe several different disk array models and key comparisons of these systems. Chapter three presents our capacity extendible disk array algorithms and shows how it works on different disk array levels. I also provide theoretical analysis of these algorithms and illustrates operations using figures. Chapter four shows the algorithm simulation and provides performance comparison. Chapter five provides a summary of our study.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we explain the disk array mechanism and some previous works. The basic elements in a disk array are data interleaved parallel disks. With different ways of data and parity layout, we can have many designs with different read/write performance and failure recovery ability.

2.1 Disk Array Mechanism

Disk Array

Figure 2-1 shows a disk array system with six disks connecting in parallel. It organizes a number of physical disks in data interleaved parallel fashion and makes the disk group appear as a single logical disk to applications [Kim86]. It is essential to solve the performance gap between CPU and disk I/O, since the performance of microprocessor technology had more rapid growth than the growth in disk I/O [Patt88]. The parallelism of the disk for data retrieving contributes the most on improving disk storage system performance. Since a single disk can only allow one transaction at a time, an application can't access a disk until the previous application finished accessing it. The singular disk's sequential data access and slow mechanical operation on the disk create a serious bottleneck for accessing data from disk. Furthermore, it is not a bad idea collecting some inexpensive disks as a RAID to compete with one expensive large disk.

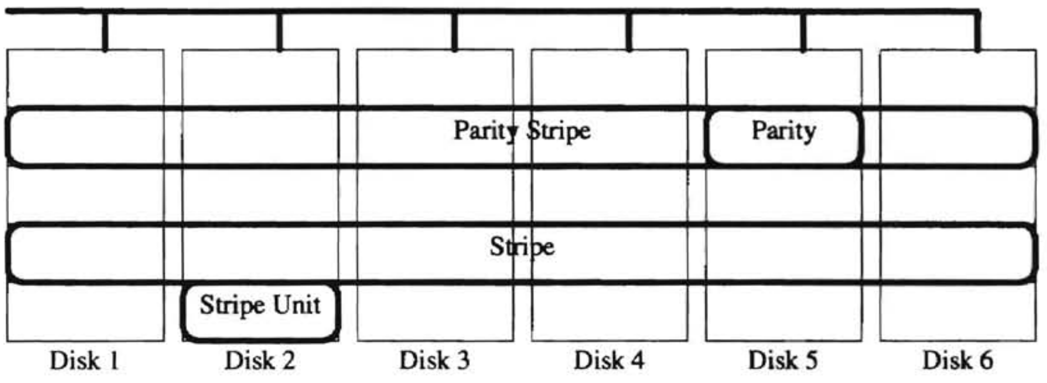


Figure 2-1 A disk array.

Stripe Unit

A *stripe unit* (also known as a *strip*) is an interleaved unit of data or parity information as shown in Figure 2-1. A group of *stripe units* can be organized as a *parity stripe* for failure recovery. A *stripe unit* which stores data is *data stripe unit.* A *stripe unit* which stores parity information is called *parity stripe unit* [Lee93]. A *stripe unit* can be a block with several kilo-bytes or a unit with only one-bit. That is, the data can be interleaved either as fine-grained or coarse-grained. The fine-grained disk arrays interleave data in small units which make all requests to access all disks in the array. The fine-grained interleaving has higher data transfer rate, since all the I/O are serving one request in parallel. The coarse-grained disk arrays interleave data in large unit size which allows many small requests to be served simultaneously [Ng89]. The Bit-Interleaved Parity (RAID level 3) is the finest grained among all RAID levels (some references may use Byte-Interleaved instead of Bit-Interleaved). Since the University of California at Berkeley has the most intensive RAID study, we adopt these RAID technology terms consistent with their publications through out this thesis [Lee93].

Stripe

A *stripe* is basically a collection of *stripe units* organized in a row as illustrated in Figure 2-2. Accessing the disk array in *stripe* can maximize the parallelism of the disk array. A disk array is organized as a collection of *stripes*.

Parity Stripe

When a *stripe* contains parity information, we can call it a *parity stripe*. The *parity stripe* is a collection of *stripe units* with common computed parity as shown in Figure 2-1. In most readings, a *stripe* can refer to both *stripe* and *parity stripe*, but the *parity stripe* only represents only the *stripe* with parity information. The computed parity information (*parity stripe units*) as disk redundancy scheme will provide information for recovering from data failure on the *parity stripe*.

Parity Scheme - Cost and Performance

The storage efficiency is defined

$$\left[\text{storage effciency} = \frac{\text{effective usable data capacity}}{\text{total disk capacity}}\right]$$ as the cost for the disk array

systems [Chen90]. The Table 2-1 summarizes the storage efficiency of different RAID levels. The RAID level 1 has no redundant data, so the storage efficiency is hundred percent. The RAID level 2 is mirrored where everything is duplicated, so the storage efficiency is only fifty percent. With H disks as Hamming Code disks, the RAID level 2 has storage efficiency $\frac{N-H}{N}$. With exactly one parity disk per parity stripe, we can expect the RAID level 3, level 4, and level 5 to have $\frac{N-1}{N}$ storage efficiency. Having two disks for parity information, the RAID level 6 has $\frac{N-2}{N}$ storage efficiency.

	LEVEL 0	LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 4	LEVEL 5	LEVEL 6
Efficiency	1	$\frac{1}{2}$	$\frac{N-H}{N}$	$\frac{N-1}{N}$	$\frac{N-1}{N}$	$\frac{N-1}{N}$	$\frac{N-2}{N}$

Table 2-1 Storage efficiency of parity scheme.

After discussion about the cost, we discuss the disk array read/write performance on different RAID levels. The RAID level 1 has the best read performance, since the disk set is duplicated for better data availability. We can always get data from one of two disks. The RAID level 0 has the best write performance, because we don't need to pay attention to write the parity information back to disks. With about the same read performance, the RAID level 5 is ideal for write among RAID level 2, level 3, level 4, level 5, and level6, because it distributes the parity information across the disks to avoid the bottleneck associated to write on the parity disk [Frie96].

Disk Data Access Scheduling for Parallelism

Figure 2-2 shows four different disk data access scheduling examples. This example has a job with three 3-block files. In schedule 1, all files are read sequentially from one disk which makes the job competition time at $T=9$. In schedule 2, the three files are interleaved into one disk which still makes the job completion time $T = 9$. Parallel access as shown in schedule 3 and schedule 4 efficiently reduce the job running time to $T = 3$. And, schedule 3 is more likely to be used, since it distributes a file to different disks for maximizing data parallelism as in disk array systems.

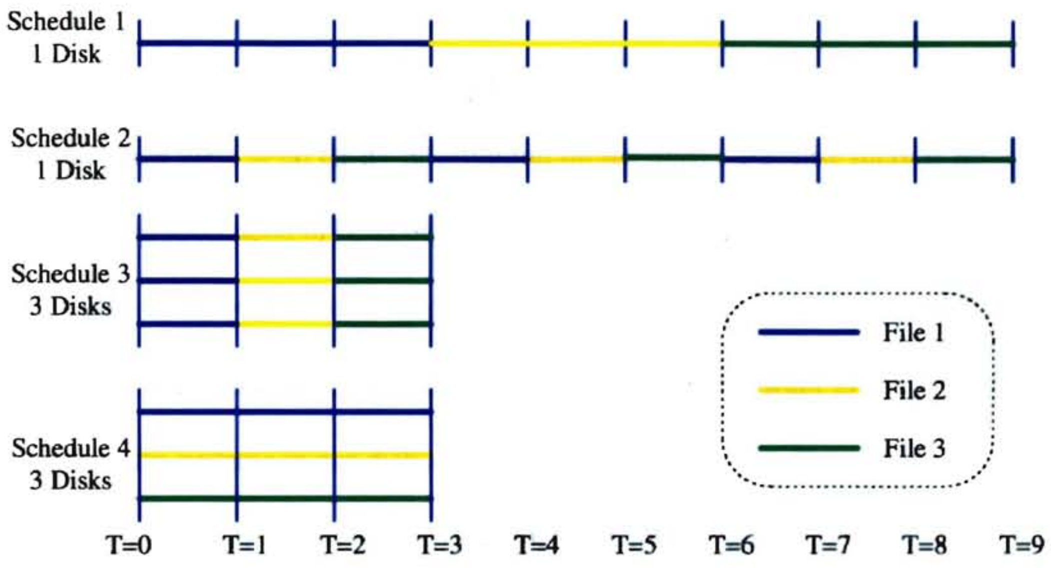


Figure 2-2 Disk Access Scheduling.

Redundancy Schemes

Basically non-redundant, mirroring, and check disk(s) are three different kinds of disk array redundancy schemes. The non-redundant disk array systems have absolutely no redundant data in the systems. The mirrored disk array systems have all data duplicated as mirroring. The check disk(s) disk array systems have check disk(s) storing parity information per parity stripe. The RAID level 0 uses the non-redundant scheme, and the RAID level 1 uses the mirrored scheme. All the Hamming Coded and parity checked RAID levels such as level 2, level 3, level 4, level 5 and level 6 are classified as check disk(s) redundant scheme. Usually RAID level 2 and level 6 use more than one check disk. The RAID level 5 and level 6 have their check disk(s) distribute to all disks in the disk array systems. The RAID level 3, level 4, and level 5 must have exactly one check disk per parity stripe [Katz89].

Non-Redundant

The non-redundant parity scheme gives us 100% usage of the disk space for storing data, since there is no parity information on the disks. Because there is no redundant information need to be kept, the non-redundant scheme has the best performance on write with no fault tolerance and higher mean time to failure (MTTF) due to increased number of disks in the system [Schu89]. We don't want to use those high failing probability systems in our disk array designs. For example, if we have two disks connect together, the risk a disk failure is double that of a single disk system. It is the lowest cost and highest risk system. The RAID level 0 is a non-redundant disk array system [Chen93].

Mirroring

It simply duplicates all disks for reliability. It is the most expensive as well as simplest redundancy design with good read performance. Since the read operation is available from one of two disks, the read performance is the best one among all RAID levels. But, the write performance will be poor because of writing on two disks, and it can't be done at the same time [Lee93]. The RAID level 1 is a mirrored disk array systems.

Check Disk

The RAID level 2, level 3, level 4, level 5, and level 6 are redundant by check disk(s). With one or more than one check disk(s), the different check disk designs can recover the disk array systems from disk failure. In the RAID level 2 check disks use Hamming Code method while the RAID level 3 and level 4 has one parity disk for failure recovery. For improving availability to users on the check disk(s), RAID level 5 and level 6 distribute the check disk(s) information across all disks in the disk array systems. There is no individual disk responsible for the parity information. Comparing to the mirrored scheme, the genuine check disk designs are more efficient in utilizing the storage capacity. The write performance will be better than the mirrored disk array as well, since they only need to write on one set of disks rather than two sets of disks in the mirrored disk array systems [Lee93].

Mapping

As a traditional disk, the RAID has analogous hierarchical mapping structure. At the top is the Stripe Table Allocation Table (STAT). It keeps the information to locate the Stripe Allocation Table (SAT) which stores the pointer for every (parity) stripe. Following the SAT, the Stripe Unit Allocation Table (SUAT) tells the actual data location. There is one SAT per disk array, and one SUAT per disk as illustrated in Figure 2-3 [Wilk96]. While we change the disk data allocation, we update the mapping tables as well.

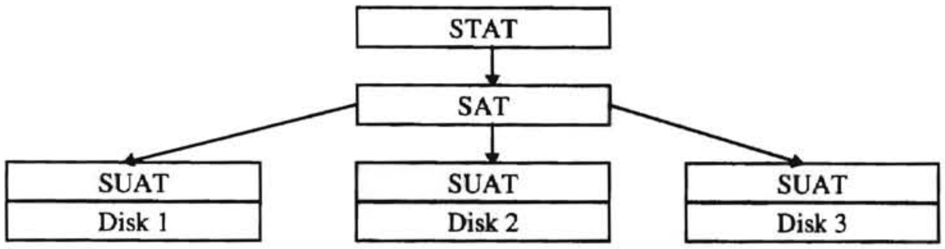


Figure 2-3 Mapping structure of disk array.

Small Read/Write Operations

The small read/write operations is disk array read/write operations which access data size smaller than a (parity) stripe as shown in Figure 2-4. To read a disk array, we can just read the data stripe units and ignore the parity information. To update a disk array with a small write, we need to read the old stripe and calculate the new parity information then write them back to the disks. These operations can be done by *read-modify-write* or *reconstruct write* as illustrated in Figure 2-4 (terms and figures adopted and modified from [Lee92]). The disk array use the *read-modify-write* if majority of the stripe units in a parity stripe is not changed. It will first read the old data stripe unit(s) (those data stripe units that we want to update) and old parity stripe unit, then compute the new parity stripe unit and write new data stripe unit(s) with new parity stripe unit back to the disk array. The *reconstruct-write* is used if the majority of the stripe units in a parity stripe need update. It will first read the rest of the old data (those data stripe units that we are not updating) and old parity stripe unit, then compute the new parity stripe unit and write the new data stripe units with new parity stripe unit back to disk array. Frequent read and write access on the parity disk degrade disk array performance in RAID level 2, level 3 and level 4, since they all have dedicated disk(s) for parity information.

The distributed parity scheme in RAID level 5 and level 6 are designed for overcoming these small write problems.

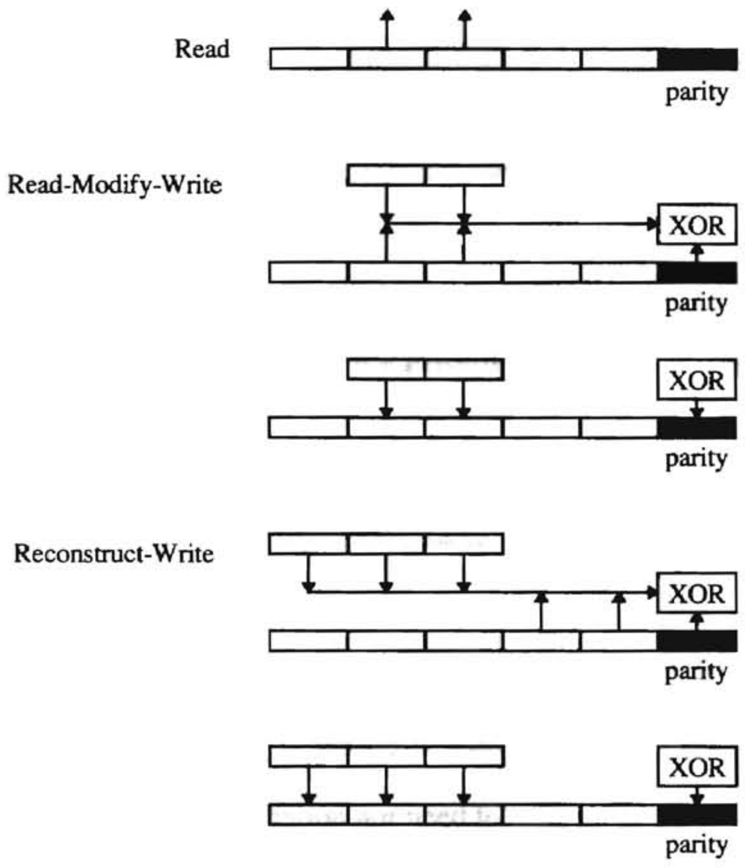


Figure 2-4 Small read and write operations.

Log-structures file systems(LFS)

The LFS appends new information to file system in a sequential buffer - *log* [Rose91], and then writes the *log* buffer to disks at one time. It increases the write performance with reduced disk seek time by batch writing, since one large write operation request has better performance than many small write operations. The Zebra file system successfully borrows this approach for its *log-based striping*. It is mainly targeted at the small write operations that degrade the disk array performance. The small writes are those write operations which only involve part of the parity stripe. The problem doesn't happen on the RAID level 0.

2.2 Disk Array Models

RAID prototype

The RAID prototype is defined as RAID level 0 through level 6 with different tradeoff in data availability, parallel I/O performance, hardware cost and redundancy overhead [Lee93, Patt88].

- Non-redundant (RAID Level 0)

  As shown in Figure 2-5, this model has no redundant data for failure recovery. The only advantage of this model is easy to write on the array since we neither need to compute redundant information nor need to store them. It has the best write operation performance among all RAID levels.

- Mirrored (RAID Level 1)

  As shown in Figure 2-6, this model is RAID level 0 with duplicated set of disks for mirroring. It has the best read operation performance and high reliability with the most expensive hardware cost. Poor write operation performance is another disadvantage of this RAID level.

- Hamming Coded (RAID Level 2)

  This model use Hamming Code scheme as redundancy algorithm. Hamming Code are stored at disk positions 2^n where $0 \le n \le \lfloor \log_2 N \rfloor$ and N is the number of disks in the disk array. For example, in a seven disks disk array, the disk 1, disk 2, and disk 4 are used for storing the Hamming Code and the disk 3, disk 5, disk 6, and disk 7 are used for storing data as illustrated in Figure 2-7.

- Bit-Interleaved Parity (RAID Level 3)

  It is fine-grained data interleaving as we discussed previously. It has exactly only one parity disk per system. It also has all the parity information placed at one parity disk which will potentially create a jam while we have several write requests on the disk array.

- Block Interleaved Parity (RAID Level 4)

  This level is similar to the RAID level 3. It has more coarse-grained data interleaving which we discussed earlier. The parity information are all in one parity disk, and it is the bottleneck while we write on the disks as shown in Figure 2-8.

- Block Interleaved Distributed Parity (RAID Level 5)

  As shown in Figure 2-9, the RAID level 5 has coarse-grained interleaved distributed parity information stored across the disk array. Unlike the RAID level 3 and level 4, it performs better when writing on the disk because of no single parity disk.

- PQ Redundant Parity (RAID Level 6)

  Figure 2-10 shows the RAID level 6 model. Instead of just using one interleaved block per parity stripe, it has two interleaved blocks (stripe units) in a parity stripe. These two blocks consist of parity information and Reed-Solomon code for recovering two disks failures.

In the Figure 2-5 to Figure 2-10, we conceptually illustrate the seven RAID levels. The shaded areas represent the redundant part of the disk array. In this example, with different number and different types of redundant disks, they all have four disks for storing data.

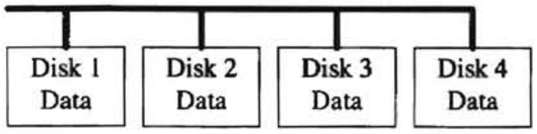


Figure 2-5 RAID level 0 as non-redundant disk array.

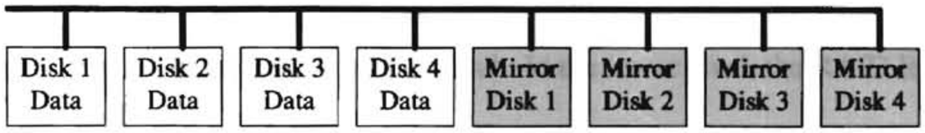


Figure 2-6 RAID level 1 as mirrored disk array.

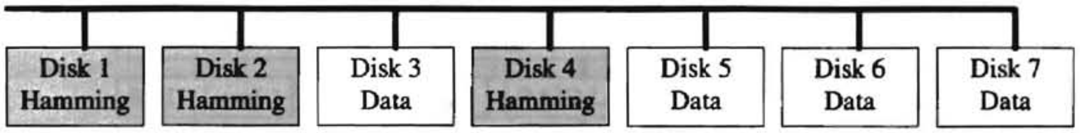


Figure 2-7 RAID level 2 as Hamming Coded disk array.

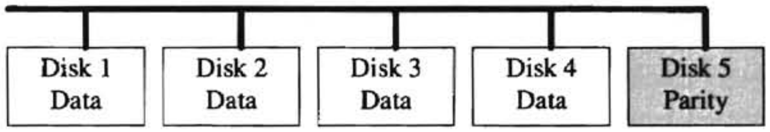
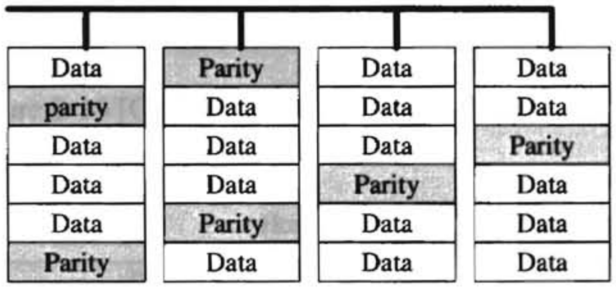
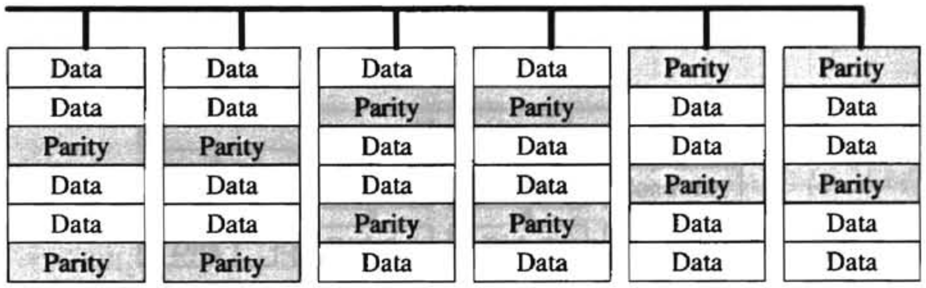


Figure 2-8 RAID level 3/level 4 as bit/block interleaved parity disk array.



Data	Parity	Data	Data
parity	Data	Data	Data
Data	Data	Data	Parity
Data	Data	Parity	Data
Data	Parity	Data	Data
Parity	Data	Data	Data

Figure 2-9 RAID level 5 as distributed block interleaved parity disk array.



Data	Data	Data	Data	Parity	Parity
Data	Data	Parity	Parity	Data	Data
Parity	Parity	Data	Data	Data	Data
Data	Data	Data	Data	Parity	Parity
Data	Data	Parity	Parity	Data	Data
Parity	Parity	Data	Data	Data	Data

Figure 2-10 RAID level 6 as distributed block interleaved parity disk array.

HP AutoRAID

The HP AutoRAID combines the mirroring performance with RAID level 5 cost-capacity benefit by mirroring active data only and storing inactive data in RAID level 5 [Wilk96]. With faster disk read operation, the HP Auto RAID applied RAID level 1 at top level of the disk array for active data retrieval. In the lower level, the RAID level 5 has faster disk write operation on inactive data.

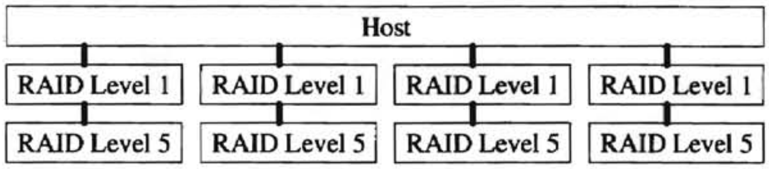


Figure 2-11 HP AutoRAID hierarchical storage system.

TickerTAIP

The TickerTAIP distributes disk controller nodes across the disks instead of using single central controller to avoid single point failure and for better fault tolerance as shown in Figure 2-12 and Figure 2-13 [Cao94].

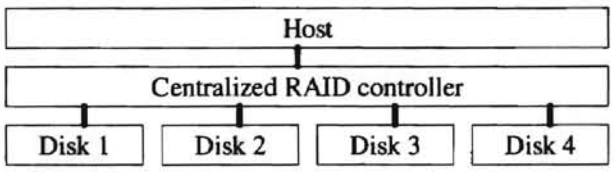


Figure 2-12 Traditional RAID with centralized controller.

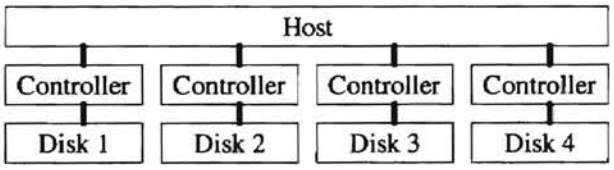


Figure 2-13 TickerTAIP disk array with distributed controller.

Zebra

Zebra file system using its log-based striping distributes all new data across multiple file servers [Hart95]. A big difference from other models is it only writes to the disk array as a new stripe instead of updating the old stripe. And a 'cleaner' mechanism will clean the old stripe later. A big advantage of the Zebra is better write operations performance in response time because it replaces the *read-modify-write* and the *reconstruct-write* with the *log-structure* as in Figure 2-14.

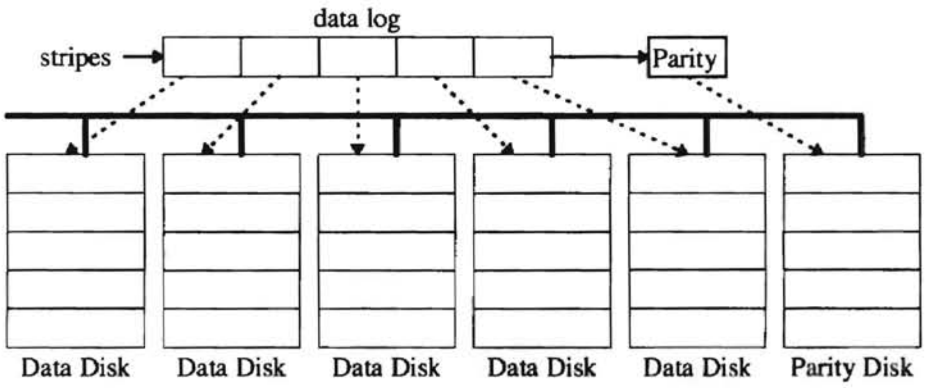


Figure 2-14 Zebra striped network file system.

NOW

One of many revolutionary results in NOW (Networks Of Workstations) research is building the RAID style secondary storage systems in workstations on the network. The NOW system totally distributes its memory and processing power to workstations in its network. All the workstation disks in NOW are accessed as a RAID (or Zebra) strategy. For example, we can imagine that there are fifty workstations connected together, and all the workstations share their local disks as components of a RAID.

After review of several disk array designs, we briefly summarize a comparison of the designs in table 2-2. Notice that the capacity extended algorithm and integrated distributed hot spare disk study is not on these researches yet.

Characteristics Comparison

	Network Based	Advantages	Striping
RAID	No	Simple	File based
HP AutoRAID	No, SCSI bus	Hierarchical architecture	Log-structure in RAID level 1 and level 5
Ticker TAIP	SCSI or FDDI	Distributed controller	File based
Zebra	Yes	Client striping	Log-structure
NOW	Yes	Distributed computing	Log-structure
DIY-RAID	No, SCSI-II bus	Cost vs. performance	N/A

Table 2-2. Comparison of several different disk array designs.

In the next chapter, we introduce our capacity extendible algorithm on any existing disk array with distributed hot spare option.

CHAPTER 3

ALGORITHMS

In this chapter, we present our algorithms - capacity extendible disk array algorithms - using examples and figures. The algorithm completion time as well as the number of read/write operations are also discussed and compared. Section 3.1 first introduces the Straight-Forward Algorithm, and then the Section 3.2 explains the Diagonal-Move Algorithm for each RAID level. Two configurations, with compaction for extra disk system capacity and without compaction for guaranteed disks system backup are described.

Compaction of disks is implemented in our Straight Forward Algorithm and optional for Diagonal-Move Algorithm. Compacted disk has files stored in contiguous units and free space is consolidated into one contiguous block. In our model, compaction will not only free space for new stripe, but also will reduce disk mechanical access time by rearranging the file fragments.

Since the backup disk can effectively reduce the necessary mean time to repair (MTTR) [Patt88] for disk arrays, a reliable hot spare disk is important for the disk array systems. Integrated with distributed hot spare backup disk in disk array, the disk system is guaranteed with more reliable spare disk embedded which is always ready for recovering failed disks immediately. The backup disk, spare disk, and hot spare disk are interchangeable terms through out this thesis.

Definition 2: Let **A** be the disk array. Then **A** has the following properties.

(a) Number of disks N. It also represents the number of stripe units on a stripe. That is exactly one stripe unit in a stripe belongs to a disk.

(b) Number of stripes per disk S. Any disk in the disk array contains S stripe units.

(c) Single disk capacity C is the capacity (number of bytes) on any single disk in the disk array.

Example 1: An illustration of a disk array with six disks.

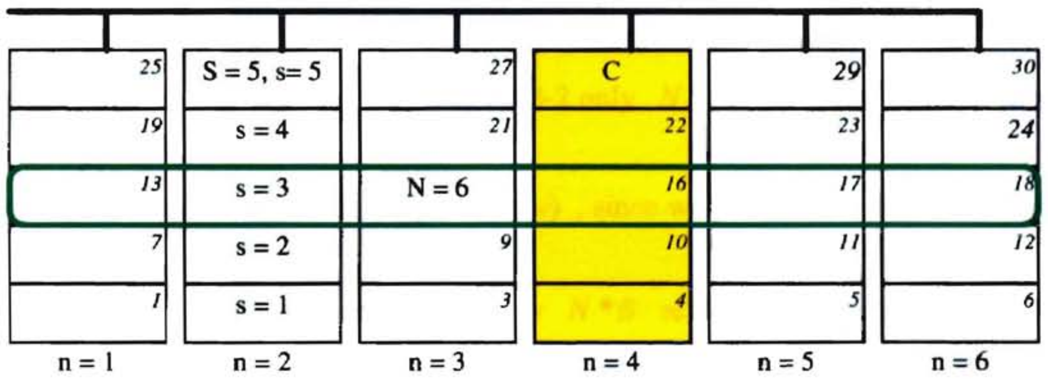


Figure 3-1 A disk array with 6 disks.

With $N = 6$, and $S = 5$, this disk array consists of six disks and each disk is divided into five stripe units. It has thirty stripe units in total. All stripe units are labeled using unique identifiers (integers). We use *s* as a variable to represent stripe units on a disk. So, *s* varies from 1 to S. And *n* is used as a variable to represent a disk in the array. So, n varies from 1 to N. Each stripe unit has its unique sequential number for identification. With N disks, S stripes in the disks systems, we shall have the disk array capacity $C * N$ bytes. And stripe unit size is $\frac{C}{S}$ bytes. The total number of stripe units is $N * S$.

3.1 Straight-Forward algorithm

In Figure 3-2, we show stripe moving of Straight Forward algorithm. After we plug in a new disk, it simply moves the stripe units to the next available space in sequential order. Depending on the disk array configuration, maximum number of stripe units need to move in the disk array is $N * S$. The upper bound completion time for this moving is $T_{level\_0} = \sum_{1}^{N*S}(Tr + Tw)$ where variable Tw is stripe unit write time, and variable Tr is stripe unit read time. For instance, in Figure 3-2 only $N*(S-1)$ stripe units need to move with a completion time of $\sum_{1}^{N*(S-1)}(Tr + Tw)$, since we put the new disk next to disk 5 instead of before disk 0 which requires exactly $N * S$ read/write operations. Theoretically the new disk can be inserted anywhere on the disk array, but we figure out that the algorithm may perform a little bit better if we append to the end. Based on the RAID level schemes, the stripe unit size can be a bit or a several bytes as a block. In this algorithm, the assumption is that each moving operation on any single stripe unit requires exactly one read and one write without overlapping on any two operations. Comparing to the Diagonal Move algorithm in next section, this algorithm takes more time for disk array rearrangement with smaller read/write units. Due to atomic data access, we need to lock the data while moving. The Straight Forward algorithm requires minimum number of locked data units (see Chapter 4). Since it locks only one unit at a time, while the Diagonal Move algorithm usually locks whole parity stripe. Though it provides maximum data availability for users, it takes too much read and write operations for data migration.

Algorithm 1: The Straight Forward Algorithm is a capacity extendible algorithm for disk array. After a new disk is added, the disk array is subject to compaction where all the stripe units are sequentially moved to its lowest available addressing space (refer to Figure 3-1 for stripe unit addressing scheme) continuous across the disks system as illustrated in Figure 3-2.

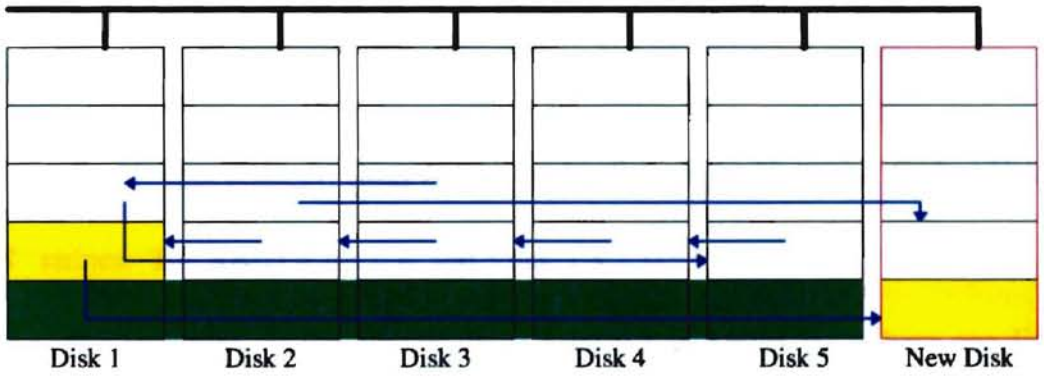


Figure 3-2 The Straight Forward Algorithm in RAID level 0.

In Straight Forward algorithm, the number of read operations is equal to the number of data stripe units on the disk (the "data stripe units" doesn't include parity stripe units). And, number of write operations is the number of stripe units in the disk array system which is $(N * S)$ at most.

Precisely, if we add the new disk as appended, the RAID level 0 requires $N*(S-1)$ reads and writes which is less than the number of reads and writes if the new disk is inserted between any two existing disks. As shown in Figure 3-2, there is one stripe at $s=1$ not involved in any read/write operations while rearranging the disk. Since the parity stripe units are only involved when we write to the disk, the RAID level 1 through RAID level 6 may have different number of read and write operations. The RAID level 1 is mirror of RAID level 0, so it takes $N*(S-1)$ reads and $2(N*(S-1))$ writes which is twice as many as of RAID level 0. Although all the RAID level 1 disks are physically parallel connected, we never access both set of disks as a stripe. We can read from one of the available disk set which only needs one stripe read operation. For the write, we need to write it twice for both set of disks. The RAID level 2 takes $\left[(N-\log_2 N-1)*S\right]$ reads and $N*S$ writes. That is because we have to have $(N-\log_2 N-1)$ data disks or $(\log_2 N+1)$ parity disks in the Hamming Code scheme with S stripes. The RAID level 3, 4, and 5 has $(N-1)*S$ reads and $(N*S)-(N-1)$ writes. In these levels (level 3, level 4, and level 5), we have one disk used for the parity information, so have only $(N-1)*S$ data stripe units in total for read operations. While we write back the data stripe units, we also need to update the parity information. So, we have $(N*S)-(N-1)$ writes on these levels. That is, we write stripe units on all disks except the $s=1$ stripe. With two parity disks, the RAID level 6 has $(N-2)*S$ reads and $(N*S)-(N-2)$ writes. Similar to RAID level 5, we read these $(N-2)*S$ data stripe units then take $(N*S)-(N-2)$ writes for data stripe units and parity information excluding these on the stripe $s=1$.

Example 2: A Straight Forward moving on a RAID level 4 disk array after adding a new disk.

In this example, we have a new disk added on a RAID level 4 disk array system as shown in Figure 3-3. The Straight Forward moving algorithm sequentially compact the stripe units to its lowest available addressing space. The Figure 3-4 illustrates the disk array after completion of the Straight Forward moving algorithm. The stripe units P1 through P5 in the parity disk are parity blocks. With special mechanism, the parity information is updated after the new stripe has completed its moving. It is that the P1 must be changed to P1' after stripe unit number 5 has been written. In this example, there are eighteen stripe units read and write operations. And no operation can be simultaneously done.

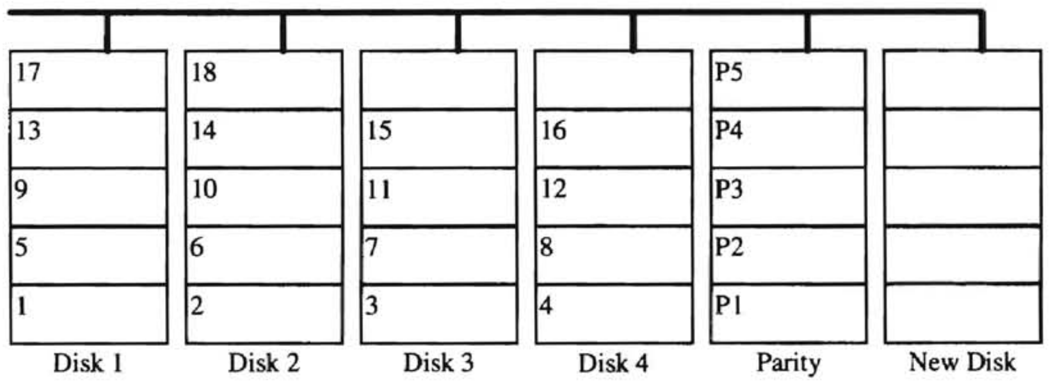


Figure 3-3 Before Straight Forward moving on RAID level 4.

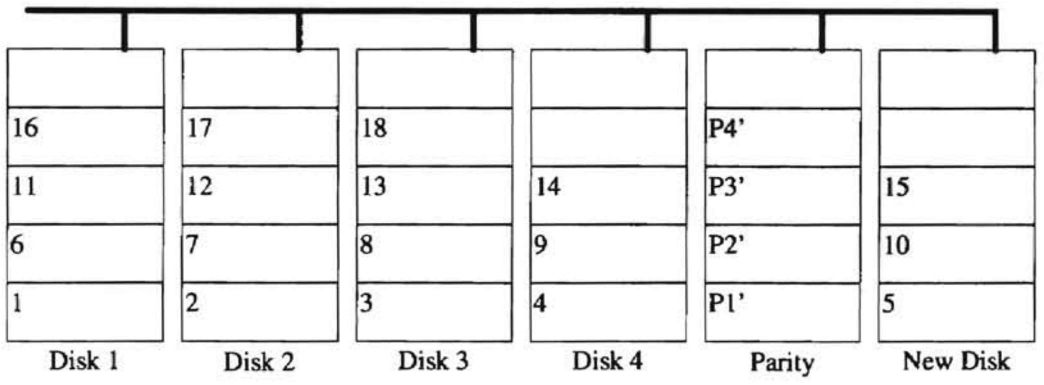


Figure 3-4 After Straight Forward moving on RAID level 4.

Completion time T for Straight Forward Algorithm for a complete disk array is total time of Tw as disk write time, Tr as disk read time for a stripe unit and time for updating parity information. Tm represents the time while some operations take place in memory such as parity computation and stripe units rearrangement if necessary in later model. Since $(N * S)$ is usually far greater than $(N-1)$, we ignore the $(N-1)$ read/write operations on the first stripe on our formula for simplicity. The Straight Forward algorithm completion time for different RAID levels are listed below.

$$T_{level\_0} = \sum_{1}^{N*S} Tr + \sum_{1}^{N*S} Tw + Tm$$

$$T_{level\_1} = \sum_{1}^{N*S} Tr + \sum_{1}^{N*S} Tw_{data} + \sum_{1}^{N*S} Tw_{mirror} + Tm$$

$$= \sum_{1}^{N*S} Tr + \sum_{1}^{2(N*S)} Tw + Tm$$

$$T_{level\_2} = \sum_{1}^{[N-(\log_2 N+1)]*S} Tr + \sum_{1}^{[N-(\log_2 N+1)]*S} Tw_{data} + \sum_{1}^{(\log_2 N+1)} Tw_{parity} + Tm$$

$$= \sum_{1}^{[N-(\log_2 N+1)]*S} Tr + \sum_{1}^{N*S} Tw + Tm$$

$$T_{level\_3} = T_{level\_4} = T_{level\_5} = \sum_{1}^{(N-1)*S} Tr + \sum_{1}^{(N-1)*S} Tw_{data} + \sum_{1}^{S} Tw_{parity} + Tm$$

$$= \sum_{1}^{(N-1)*S} Tr + \sum_{1}^{N*S} Tw + Tm$$

$$T_{level\_6} = \sum_{1}^{(N-2)*S} Tr + \sum_{1}^{(N-2)*S} Tw_{data} + \sum_{1}^{2S} Tw_{parity} + Tm$$

$$= \sum_{1}^{(N-2)*S} Tr + \sum_{1}^{N*S} Tw + Tm$$

3.2 Diagonal-Move algorithm

In this section, we demonstrate our Diagonal Move algorithm applied in all six levels of RAID structure. In our proposed algorithm, all the diagonally allocated data stripe units in the disk array will be moved to the new disk. Then we have an option to compact the disks for extra space or not compact the disk for guaranteed fast failure recovery (hot spare) disk.

The reason for diagonally moving data stripes to the new disk is that we can avoid the bottle neck on Straight Forward algorithm which is read/write at one disk at a time by simultaneously performing read/write on several disks in the disk array. And the reason for us to choose distributed hot spare disk is because we need to guarantee that the hot spare disk is always working while we need it.

In the Diagonal Move algorithm, we also have options on either compacting the disk array data or not compacting the disk array data. If we compact the disk, we will have our extended disk space at the top of the disks. If we choose not to compact, we distribute stripe units as a guaranteed hot spare disk [Wilk96] with reduced mean time to repair (MTTR) [Patt88].

The hot spare disk for disk array is not used, but physically connected with other working disks for reduced MTTR. Once there is a disk failure, the hot spare disk can replace the failed disk automatically without waiting for human manually exchanging disk, if there is a way to recover. Usually the parity information or mirrored disks provide the recovering information. It effectively reduces the mean time to repair the disk system with a little more hardware complexity. In our model with this option, we are not increasing the capacity for the existing disk array storing data after adding a new disk, but we provide a new guaranteed backup disk for any single disk failure recovery on the disk array systems except RAID level 0.

The guaranteed backup option in our model is provided by these diagonally distributed stripe units. The old fashion passive way to provide a backup disk is manually replacing the bad one when a working disk fails. An active way is to provide a hot spare disk which is an additional disk always connected on the systems for immediate failure recovery [Patt89]. The added independent extra hot spare disk may have gone bad before one of the working disks failure as in Figure 3-7. In this case, the hot spare disk does not serve its purpose. The distributed hot spare disk which is embedded in the disk array as in Figure 3-5 can be a guaranteed way for fast failure recovery. The shaded area, stripe units B1 to B5, in Figure 3-5 are these stripe units as distributed hot spare disk for better MTTR. The distributed guaranteed backup disk is not able to apply on the RAID level 0, because there is no redundant information for failure recovery. With this distributed backup mechanism, the RAID level 2 disk array can tolerate even more disk failures.

Figure 3-5 and 3-6 demonstrate the example of failure recovery of guaranteed distributed hot spare disk. The parity disk will be responsible for providing information while the disk array system needs to recover the bad disk. For example, when the disk 4 fails the parity units 2, 3, 1 and P1 will calculate the failed part and reproduce it to the hot spare at disk 1. Stripe units 5, 6, 7 and P2 will recover the fail unit on disk 4 to the disk 2 and so on. We are guaranteed the hot spare disk is always alive while we need it.

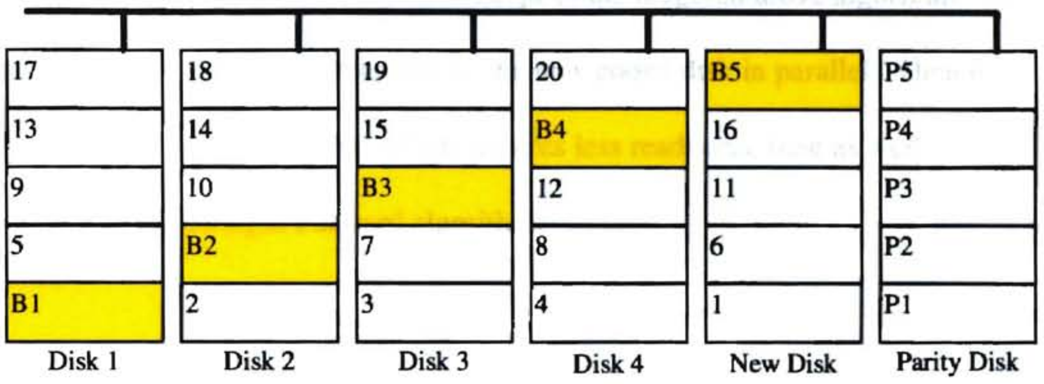


Figure 3-5 Diagonal Moving on RAID level 4 with distributed hot spare disk.

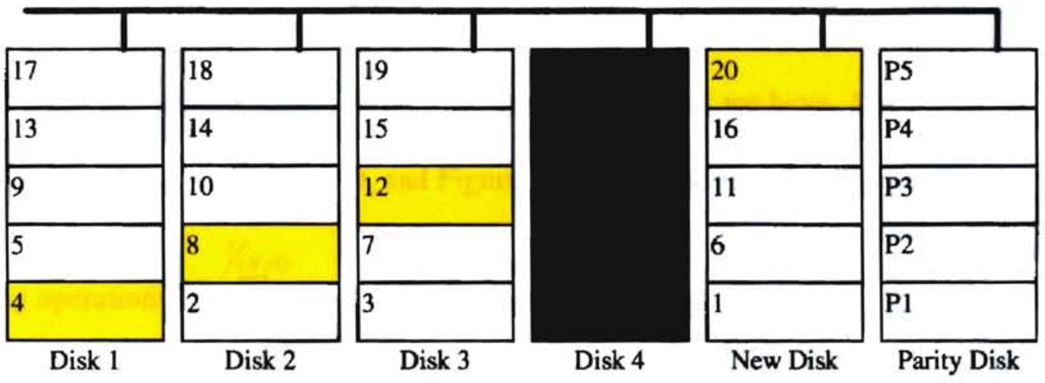


Figure 3-6 Failure recovery on distributed hot spare disk.

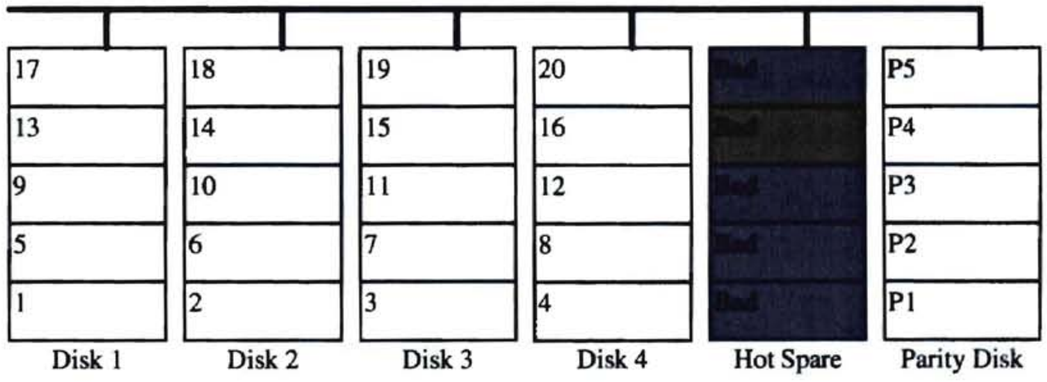


Figure 3-7 Impossible failure recovery on a bad hot spare disk.

Figures 3-8 to 3-10 show us the concept of the diagonal move algorithm. It moves the data from old disk array architecture to the new added disk in parallel. Then it compacts the disks stripe by stripe which requires less read/write time as well as operations than the Straight Forward algorithm.

In this example, we need $\frac{S}{N+1}$ stripe reads for these diagonally distributed data stripe units (N stripe units are read in parallel), and $N*\frac{S}{N+1}$ writes for these stripe units copying to the new disk. On the compacting operation, we have $S-2$ stripe reads and writes as shown in Figure 3-9, and Figure 3-10. So it will result the total time for moving operations in $\sum_{1}^{S/(N+1)} Tr + \sum_{1}^{N*S/(N+1)} Tw + \sum_{1}^{S-2} Tr + \sum_{1}^{S-2} Tw + Tm$ where the $\sum_{1}^{S/(N+1)} Tr$ is the time for diagonal reads on the disk array, the $\sum_{1}^{N*S/(N+1)} Tw$ is the time for writes on the new added disk, the $\sum_{1}^{S-2} Tr$ is the time for stripe reads while disk array compacting, and $\sum_{1}^{S-2} Tw$ is the time for stripe writes while disk array compacting. Comparing this to the Straight Forward algorithm in RAID level 0 with $T_{level\_0} = \sum_{1}^{N*S} Tr + \sum_{1}^{N*S} Tw + Tm$ operation time, we are paying less operations in less time on the reads and writes for the disk array stripe units rearrangement. To prove it, we need to have $\frac{S}{N+1} + (S-2) < (N*S)$ for the reads, and $N*\frac{S}{N+1} + (S-2) < (N*S)$ for writes.

- For the read operations, we prove $\frac{S}{N+1}+(S-2)<(N*S)$.

  Set $\frac{S}{N+1}+(S-2)<\frac{S}{N}+S<(N*S)$

  $\frac{S}{N}+S<(N*S)$

  $\frac{1}{N}+1<N$ where $N \neq \{0,1\}$

  Since both $\frac{S}{N+1}+(S-2)<\frac{S}{N}+S$ and $\frac{S}{N}+S<(N*S)$ are true, the

$\frac{S}{N+1}+(S-2)<(N*S)$ is true as well. So, we just proved that the Diagonal Move algorithm require less reads in less time than the Straight Forward algorithm.

- For the write operations, we prove $N*\frac{S}{N+1}+(S-2)<(N*S)$.

  Set $N*\frac{S}{N+1}+(S-2)<\frac{N*S}{N+1}+S<(N*S)$

  $\frac{N*S}{N+1}+S<(N*S)$

  $\frac{N}{N+1}+1<N$ where $N \neq \{0,1\}$

  Because both $N*\frac{S}{N+1}+(S-2)<\frac{N*S}{N+1}+S$ and $\frac{N*S}{N+1}+S<(N*S)$ are

true, the $N*\frac{S}{N+1}+(S-2)<(N*S)$ is true as well. Again, we just proved that the Diagonal Move algorithm require less writes in less time than the Straight Forward algorithm.

Example 3: Concept of Diagonal Move algorithm.

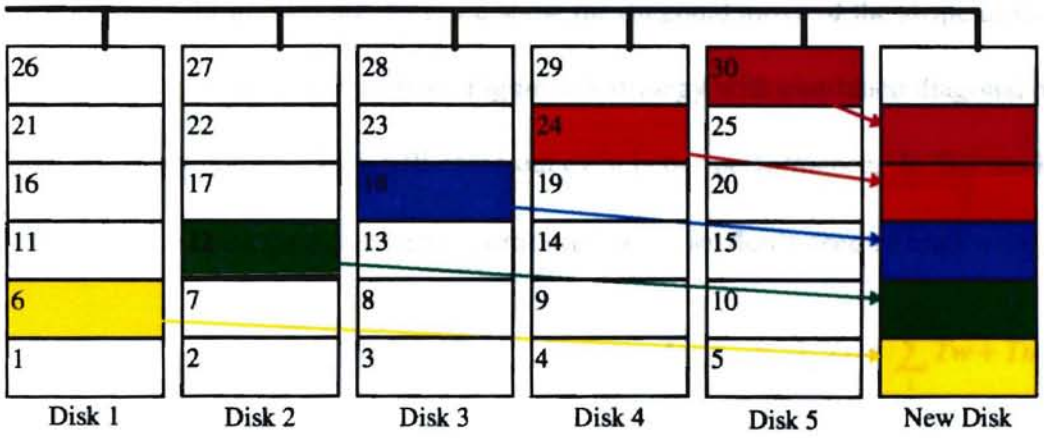


Figure 3-8 Diagonal Move on RAID level 0.

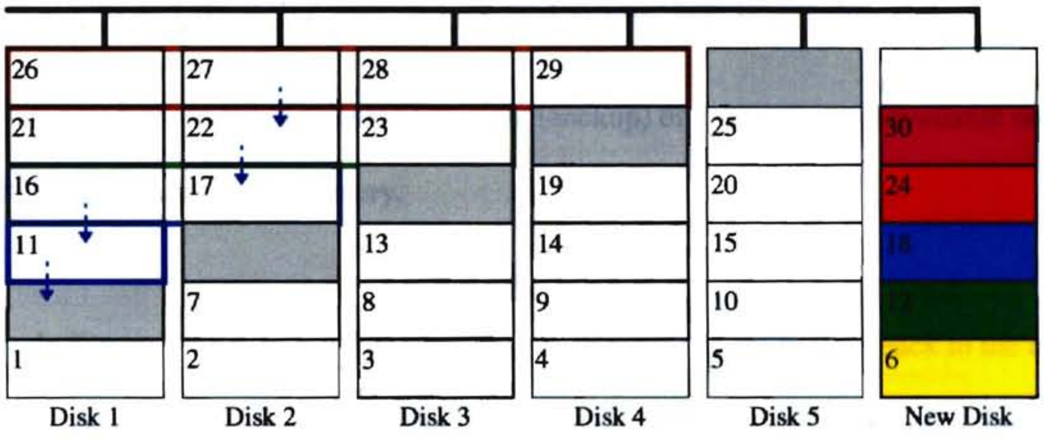


Figure 3-9 Diagonal Move on RAID level 0.

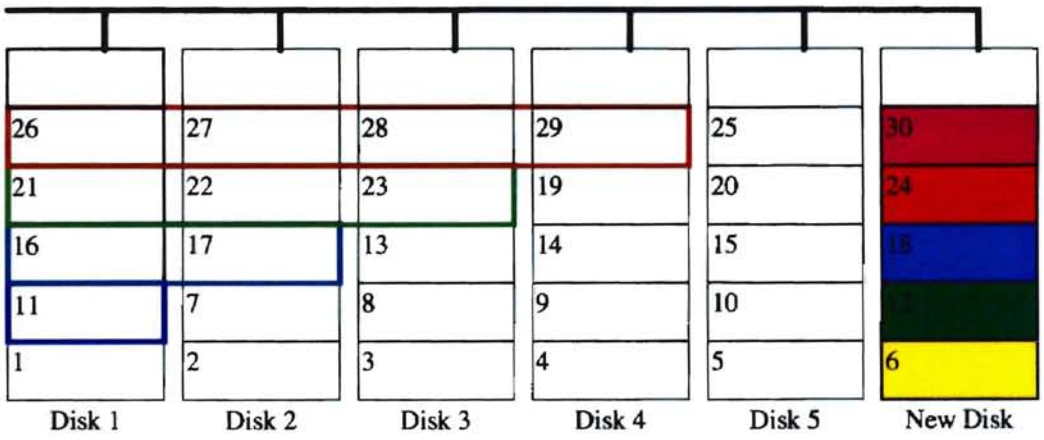


Figure 3-10 Diagonal Move on RAID level 0.

3.2.1 Non-Redundant - RAID level-0.

In Figure 3-11 and Figure 3-12, we show the diagonal move of the stripe units in the disk array system. It is derived from Figure 3-8 strategy with combined diagonal move and compaction operations which will come out even better performance. In this model, we have $(S-1)$ stripes read and write operations (since we don't need to read/write the first stripe $s=1$) which results total execution time in $T_{level\_0*} = \sum_{1}^{(S-1)} Tr + \sum_{1}^{(S-1)} Tw + Tm$. It is simply about N times less than the Straight Forward algorithm with $T_{level\_0} = \sum_{1}^{N*S} Tr + \sum_{1}^{N*S} Tw + Tm$ execution time. Because there is no redundant information for the RAID level 0, the distributed hot spare (backup) disk option is not available on this RAID level for fast failure recovery.

In Figure 3-11, we read the disk array in stripe. Then we write it back to the disk array in new format in stripe as shown in Figure 3-12. By this way, we not only compact the disk array, but we can also find that these diagonally allocated stripe units 6, 12, 18, 24 in Figure 3-11 are reallocated to the new disk in Figure 3-12. Notice there are no two stripe units from the same stripe accessing on the same disk. That is, all the stripes reads and stripes writes are in their maximum parallelisms.

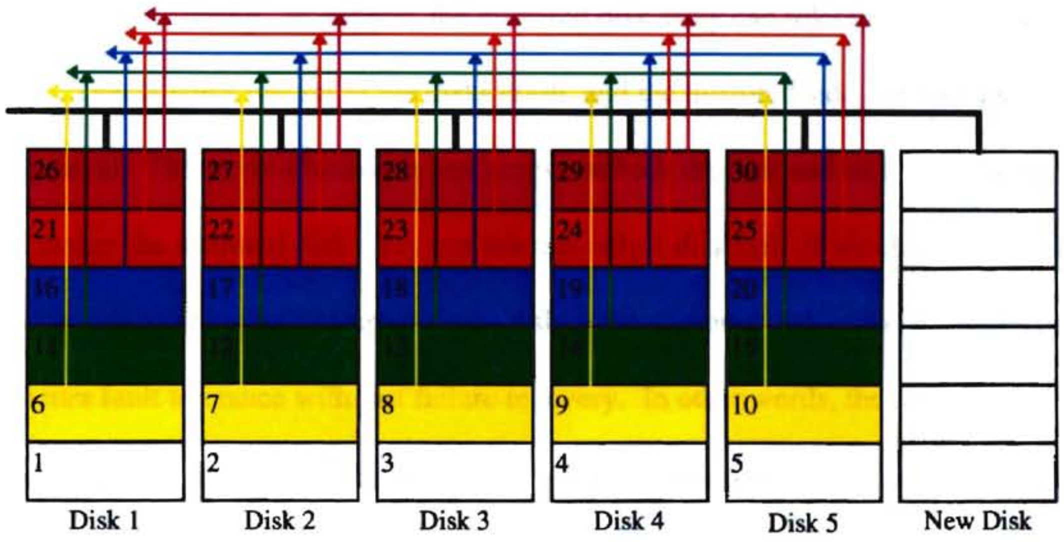


Figure 3-11 Diagonal Move on RAID level 0 with read operations.

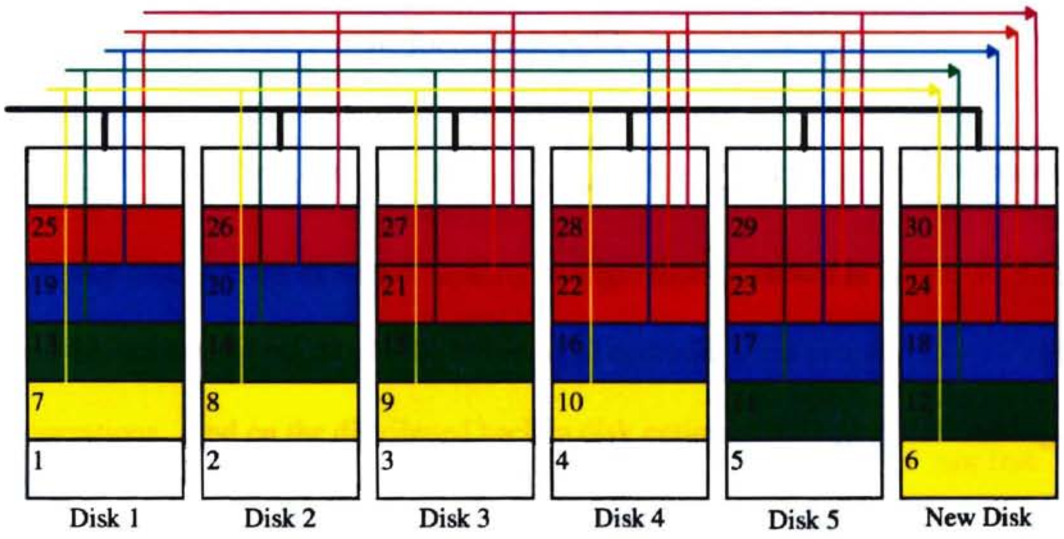


Figure 3-12 Diagonal Move on RAID level 0 with write operations.

3.2.2. Mirrored - RAID level-1.

Since all data are duplicated in this level, two new disks need to be installed at the same time for mirroring. In general, the mirrored disk array can tolerate 1 to N disks failures. That is when one set of the disks crash, and the mirrored side can take place for data retrieval. The worst situation is lost access on both the disk and its mirror image which makes the mirrored disk array can tolerate only 1 disk fail. If we choose guaranteed backup as our option after adding two new disk in the mirrored disk array, we will have even better fault tolerance with fast failure recovery. In other words, the new mirrored disk array systems can tolerate up to $(N+1)$ disks failures without replacing disk. That is because we can tolerate one disk failure from one set of disks, and complete failure of the mirror image.

The algorithm running time for extended disk system capacity is

$T_{level\_1*} = \sum_{1}^{S-1} Tr + \sum_{1}^{2(S-1)} Tw + Tm$, and $T_{level\_1*} = \sum_{1}^{S/(N+1)} Tr + \sum_{1}^{2N\left(S/(N+1)\right)} Tw + Tm$ for distributed hot spare backup disk. On the extended capacity configuration as shown in Figure 3-15 and Figure 3-16, we have at most $(S-1)$ stripes read operations and at most $(S-1)$ stripes write operations. And on the distributed backup disk option as in Figure 3-13 and Figure 3-14, we have one stripe read on every $(N+1)$ stripes (since we diagonally read) which means we have $\frac{S}{N+1}$ stripes to read. While we write back to the new disk, we are required to have $N * \frac{S}{N+1}$ stripe units to write in sequentially on two set of disks. That is simply because each stripe that we read needs N writes for all stripe units.

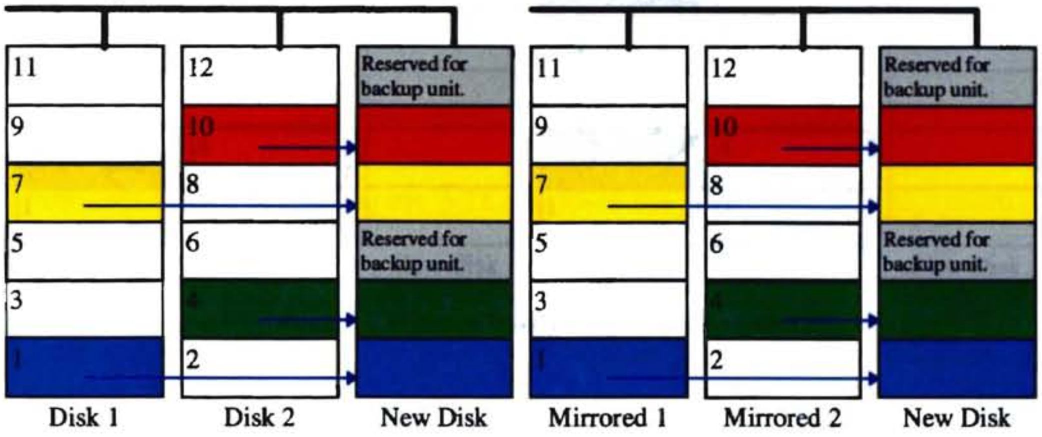


Figure 3-13 Diagonal Move on RAID level 1.

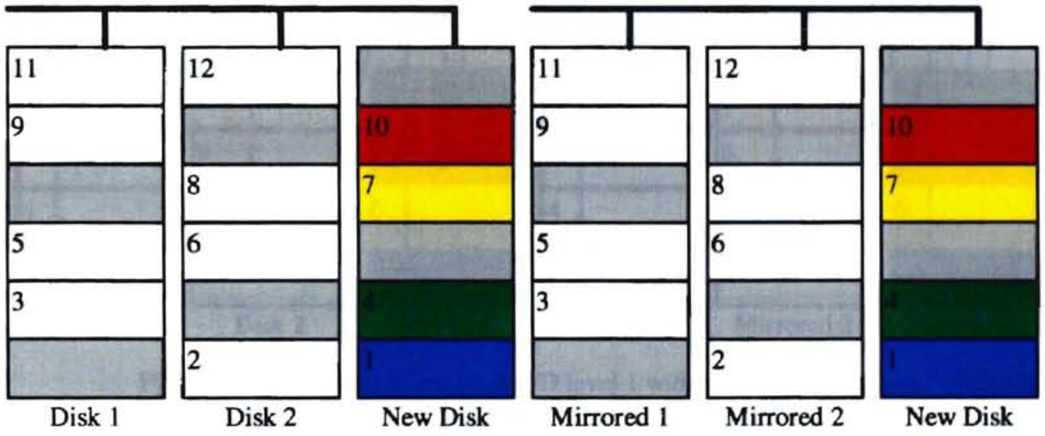


Figure 3-14 Diagonal Move on RAID level 1 with distributed backup disk.

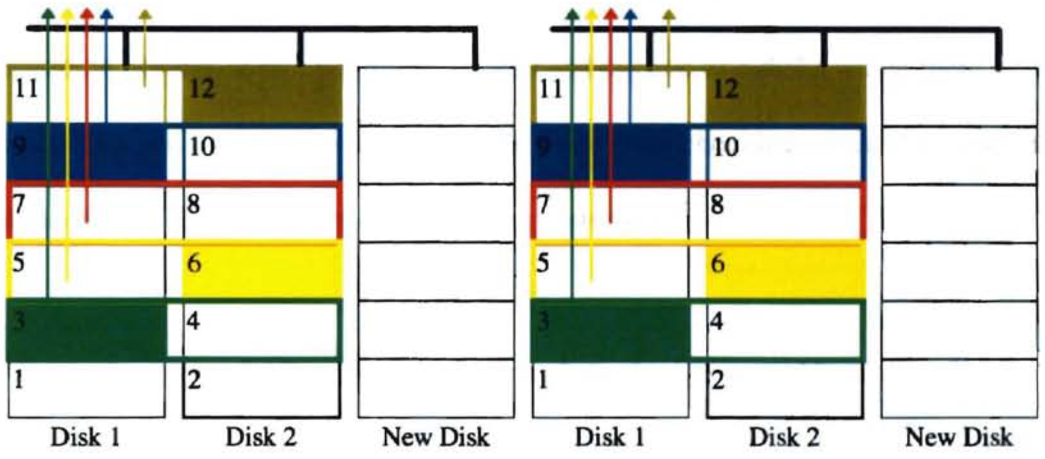


Figure 3-15 Diagonal Move on RAID level 1 with extended disk space.

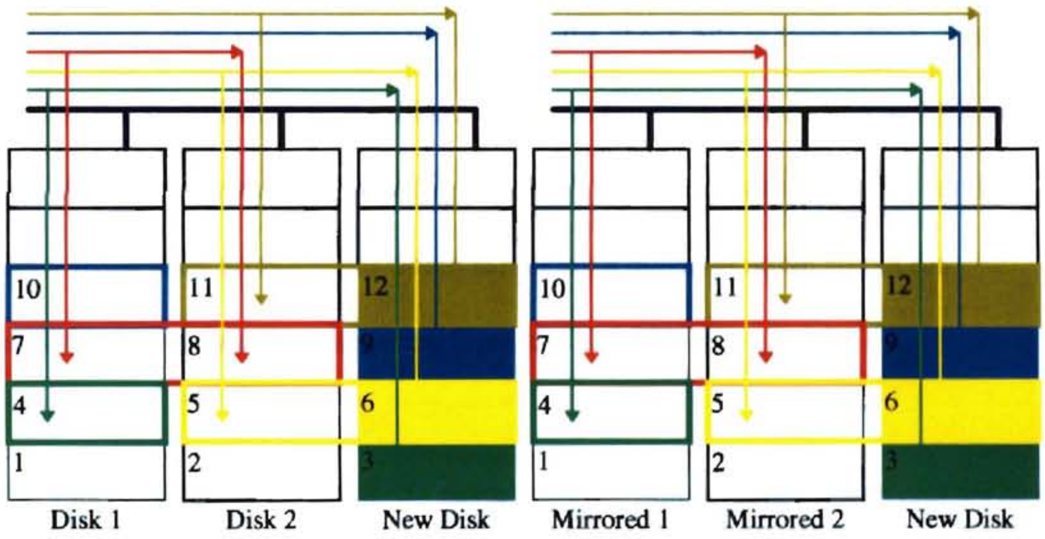


Figure 3-16 Diagonal Move on RAID level 1 with extended disk space.

3.2.2 Hamming Code - RAID level-2.

The Hamming Coded parity scheme use more than one error checking disks on a stripe based on powers of two positions, so we need to have two disks which there is one disk for the data and one disk for the Hamming Code error checking added if number of the disks in the disk array system - N reaches powers of two.

From the Figure 3-17 to Figure 3-18, we show the Diagonal Move algorithm applied on the RAID level 2 - the Hamming Code scheme. In disk array, those powers of two *check bits* in the Hamming Code are putting together in the *check disks* as disks 1, disk 2, and disk 4 in Figure 3-17. There is no necessity to move and read stripe units in these check disks, since we are going to modified the check bits in these check disks while we write the stripe units back to the disks. So in the Figure 3-17 we are actually reading the whole stripe to the memory excluding the check bits, then write it back to the disk with a new stripe units arrangement which is making a diagonal move layout for distributed hot spare disk or compaction format.

In this RAID level, we need S stripe reads as well as writes, therefore, we have the execution time as $T_{level\_2*} = \sum_{1}^{S} Tr + \sum_{1}^{S} Tw + Tm$ on both distributed hot spare disk and extended disk capacity options.

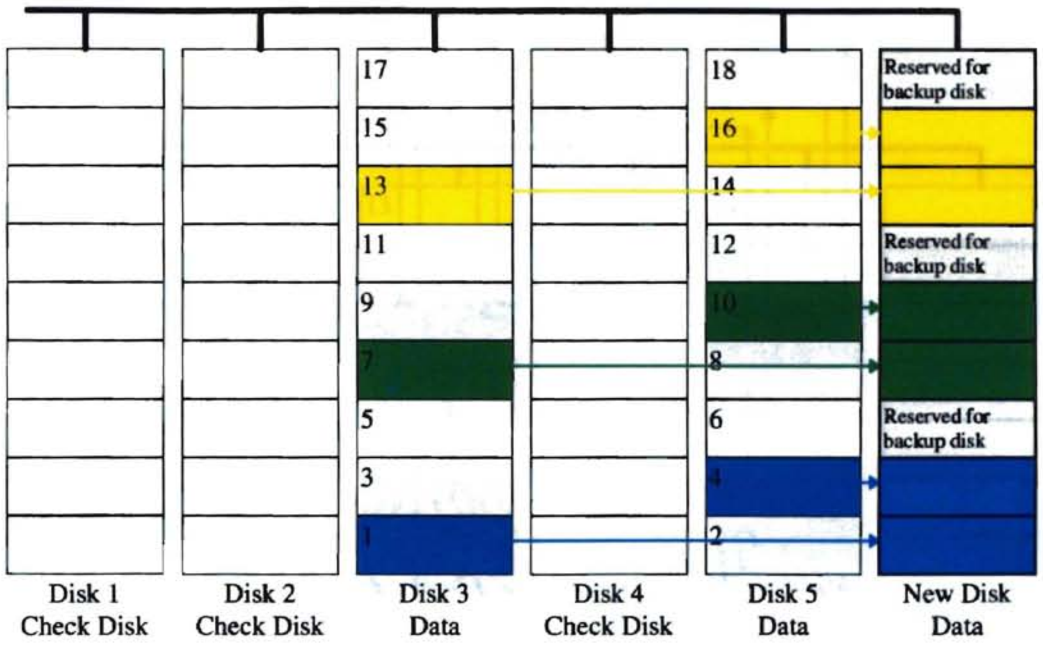


Figure 3-17 Diagonal Moving on RAID level 2.

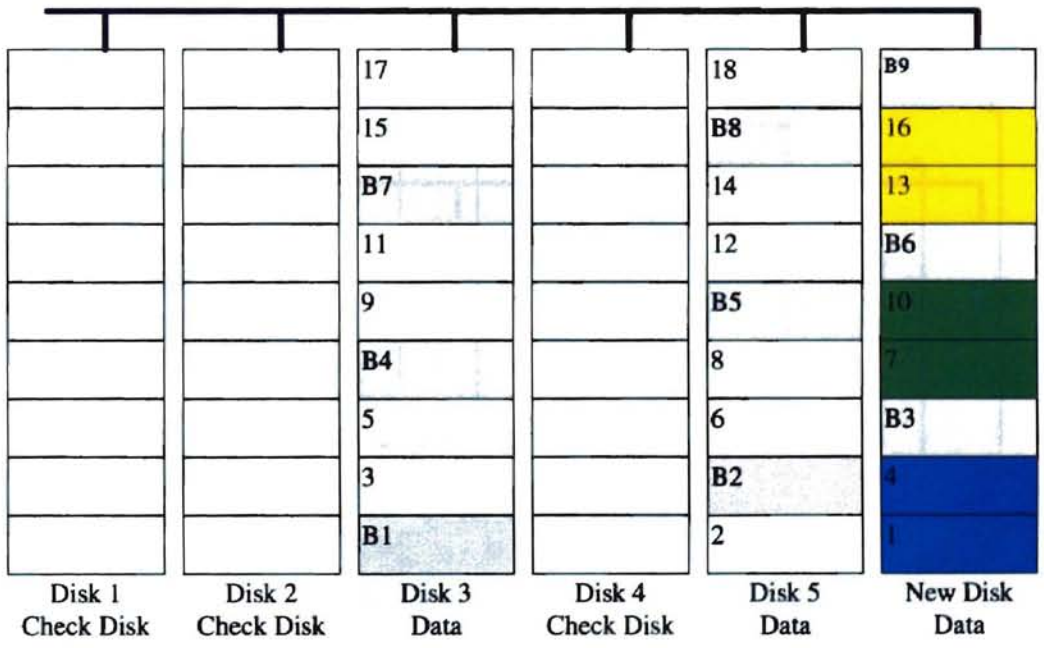


Figure 3-18 Diagonal Moving on RAID level 2.

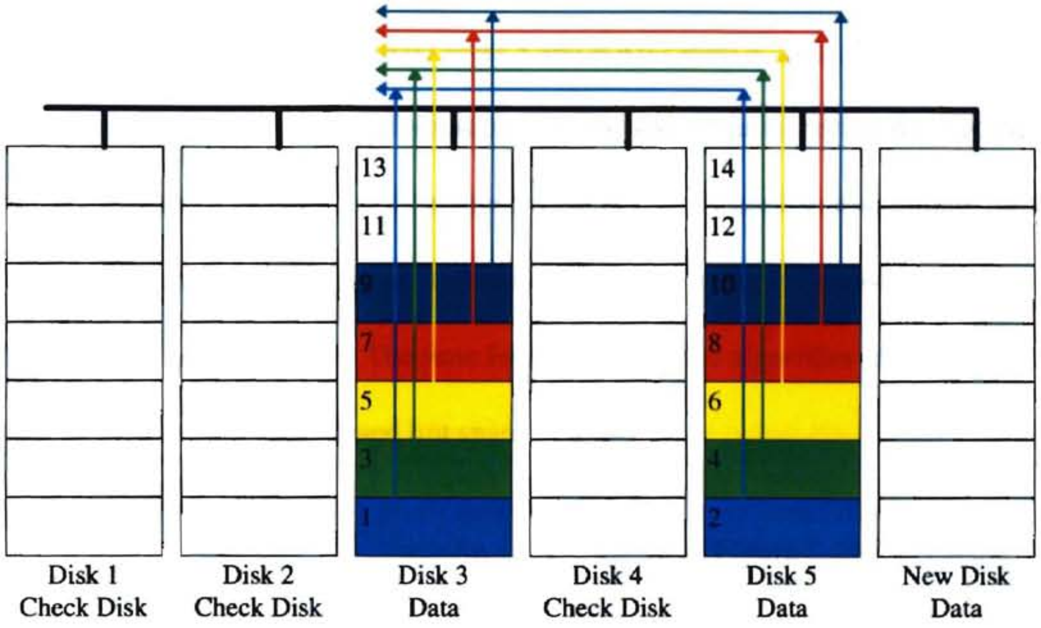


Figure 3-19 Diagonal Moving on RAID level 2.

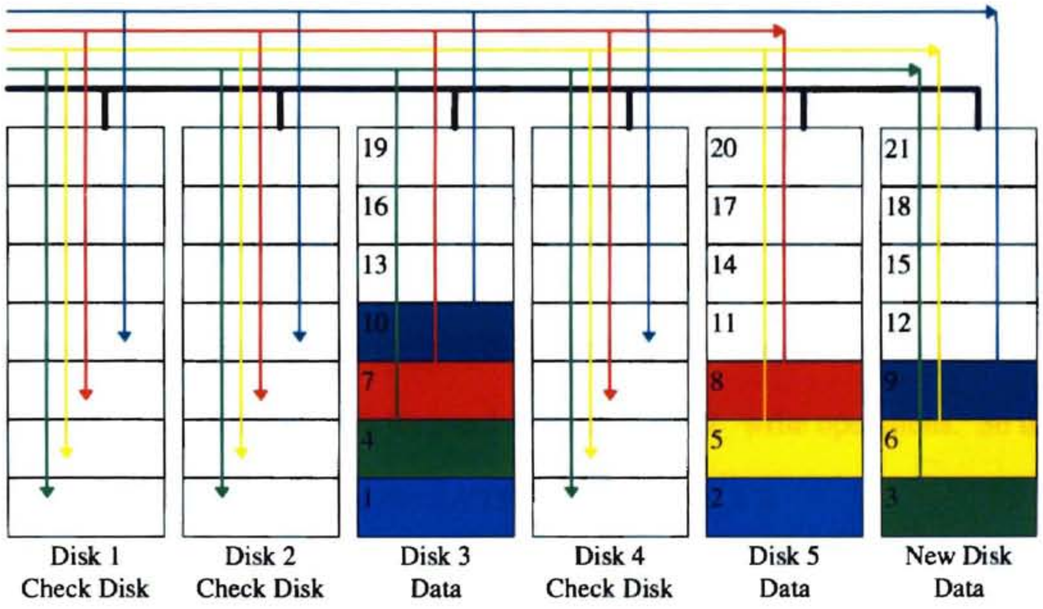


Figure 3-20 Diagonal Moving on RAID level 2.

3.2.3 Bit-Interleaved/Block-Interleaved - RAID level-3-4.

In the parity disk levels - RAID level 3 and RAID level 4, as in the RAID level 2, we need to take care these parity information while stripes require storing them at the particular place as defined in RAID. In other words, we may need to re-compute the parity information and write those parity stripe units back to the parity disk while we moving data stripe units around. The time for completing the algorithm for these levels with fast failure recovery distributed hot spare disk is

$T_{level\_3*} = T_{level\_4*} = \sum_{1}^{S/(N+1)} Tr + \sum_{1}^{N(S/N+1)} Tw + Tm$ as illustrated in Figure 3-21 and Figure 3-22.

Or $T_{level\_3*} = T_{level\_4*} = \sum_{1}^{S/(N+1)} Tr + \sum_{1}^{N(S/N+1)} Tw + \sum_{1}^{S} Tr + \sum_{1}^{S} Tw + Tm$ for increased disk system capacity as shown in Figure 3-23 and Figure 3-24.

For the distributed hot spare disk configuration, we take $\frac{S}{N+1}$ stripe read operations for the diagonal reads. Then we need $N * \left(\frac{S}{N+1} \right)$ stripe unit write operations for sequential write on the new disk. Because the reads are parallel and writes are sequential for each stripe, each diagonal read requires N write operations. So the time for diagonal hot spare disk in the RAID level 3 and level 4 is

$T_{level\_3*} = T_{level\_4*} = \sum_{1}^{S/(N+1)} Tr + \sum_{1}^{N(S/N+1)} Tw + Tm$. In Figure 3-22, the stripe units B1 to B9 represent our distributed hot spare disk units. Since we are just reordering the parity stripes and there is no new data added in any parity stripe, there is no necessary to update the parity stripe units as in Figure 3-22.

From Figure 3-23 to Figure 3-26, we are showing the extended disk array capacity configuration for the RAID level 3 and level 4. After diagonal move of the disk array as in Figure 3-23 and Figure 3-24, we compact the data together for extended disk array capacity configuration as in Figure 3-25 and Figure 3-26. As before the read requests access only the data disks, and the write operations access on all data disks including parity disk. And it is easy for we to find out that S stripe reads and S parity stripe writes are necessary for the compaction as in Figure 3-25 and Figure 3-26. In Figure 3-25 and Figure 3-26, the stripe units P1' to P8' in the parity disk are updated while the parity stripe write back to the disk array. Therefore, the compacting operation takes $\sum_{1}^{S} Tr + \sum_{1}^{S} Tw + Tm$ as execution time. Together with run time from diagonal move $\sum_{1}^{S/(N+1)} Tr + \sum_{1}^{N(S/N+1)} Tw + Tm$, the operation time for the extended disk array capacity configuration is $T_{level\_3*} = T_{level\_4*} = \sum_{1}^{S/(N+1)} Tr + \sum_{1}^{N(S/N+1)} Tw + \sum_{1}^{S} Tr + \sum_{1}^{S} Tw + Tm$.

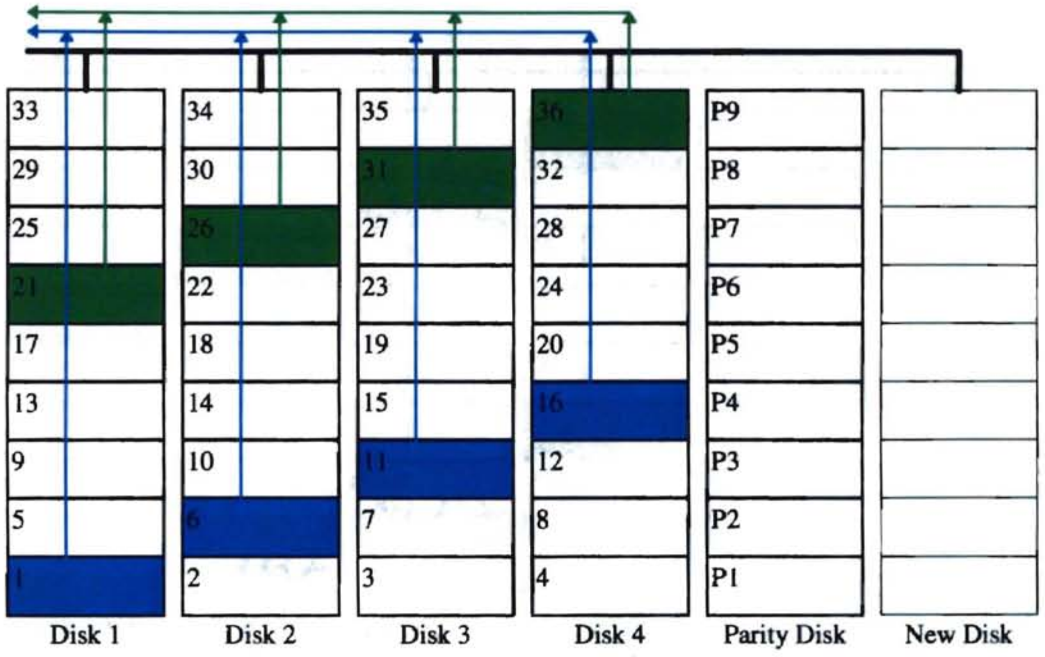


Figure 3-21 Diagonal Moving on RAID level 3 and level 4.

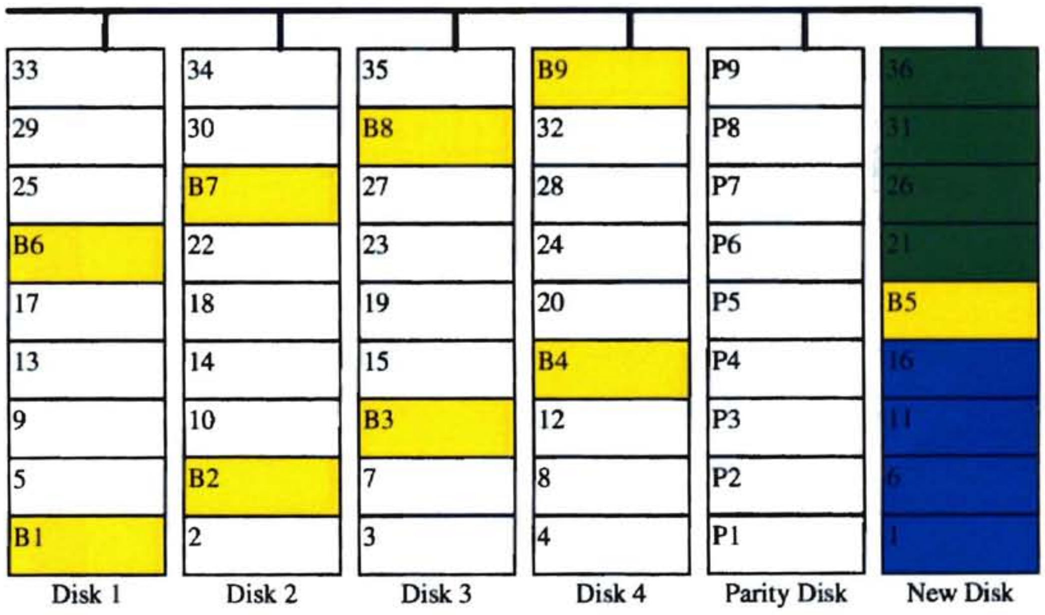


Figure 3-22 Diagonal Moving on RAID level 3 and level 4 with distributed hot spare disk.

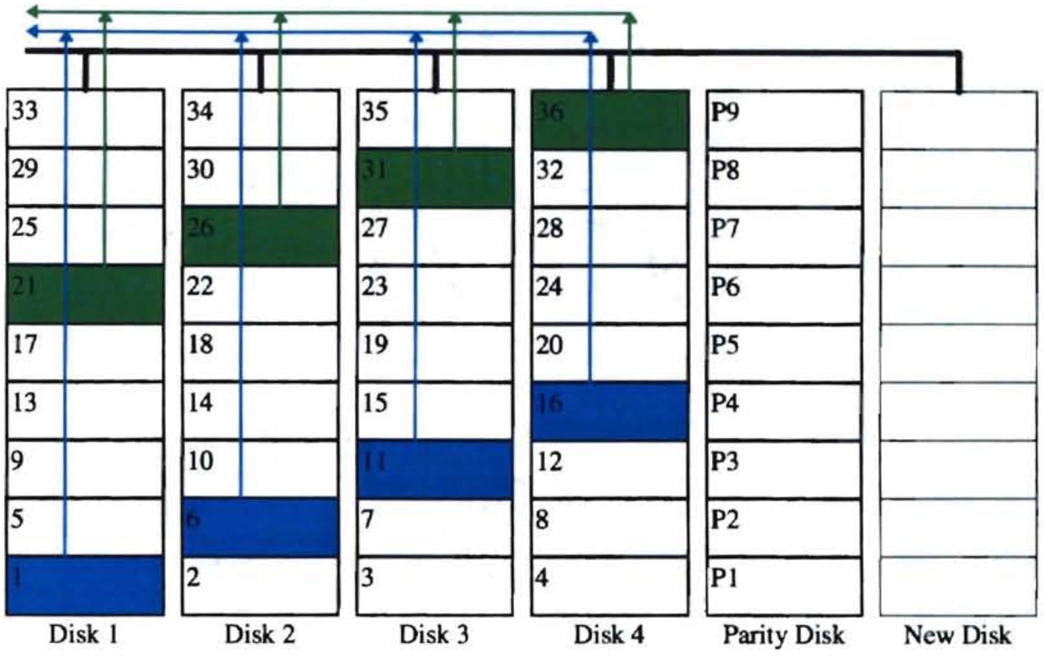


Figure 3-23 Diagonal Moving on RAID level 3 and level 4.

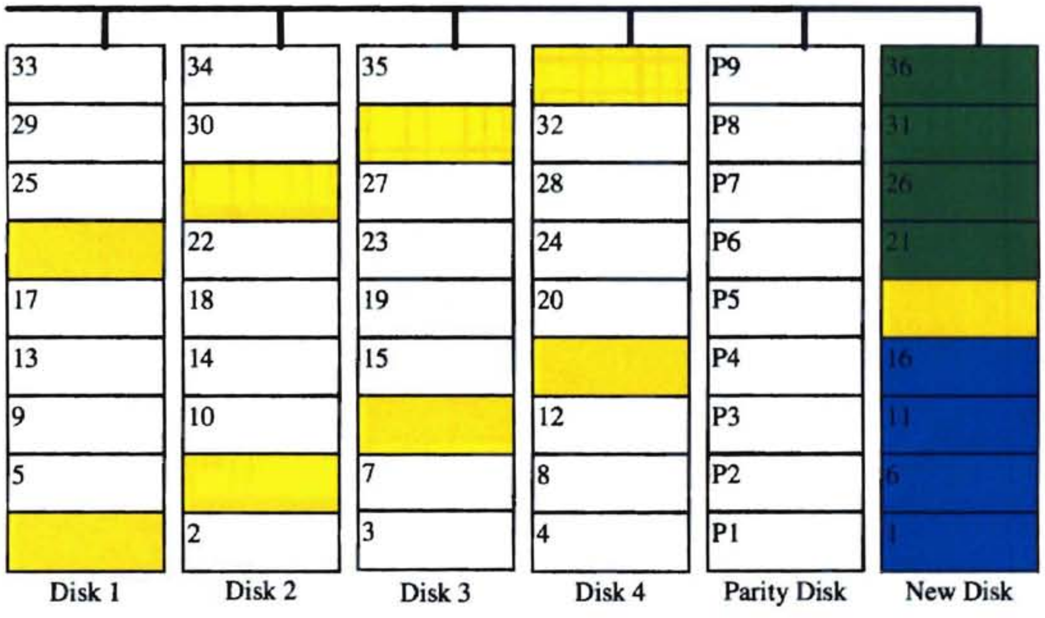


Figure 3-24 Diagonal Moving on RAID level 3 and level 4.

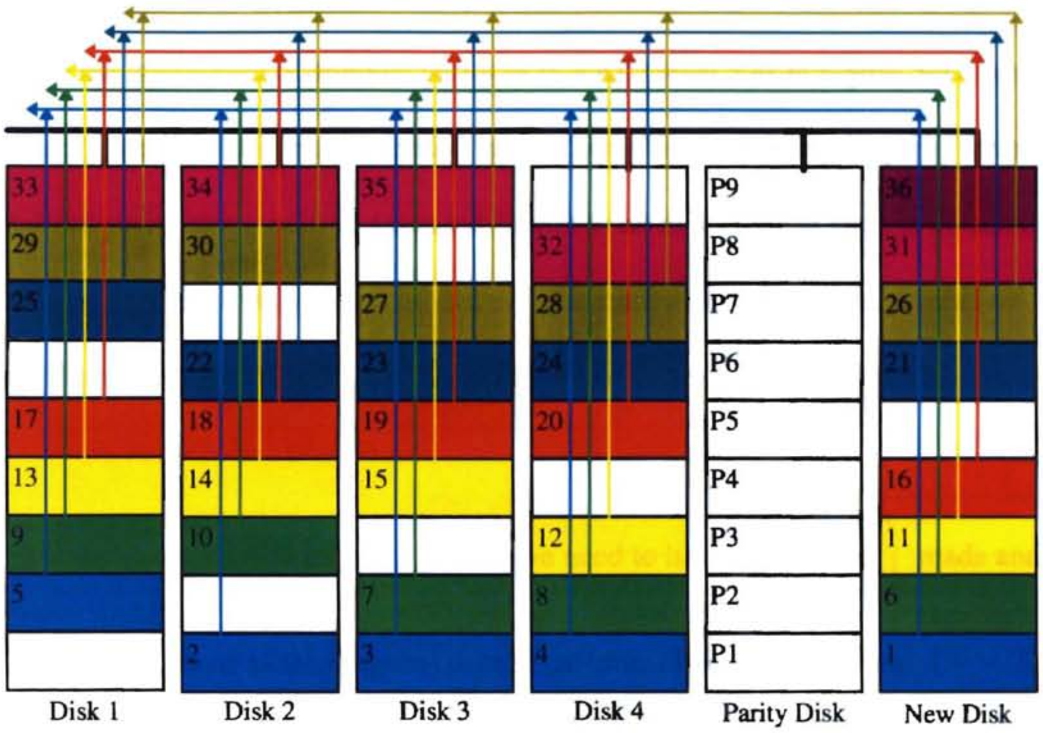


Figure 3-25 Diagonal Moving on RAID level 3 and level 4 with extended capacity.

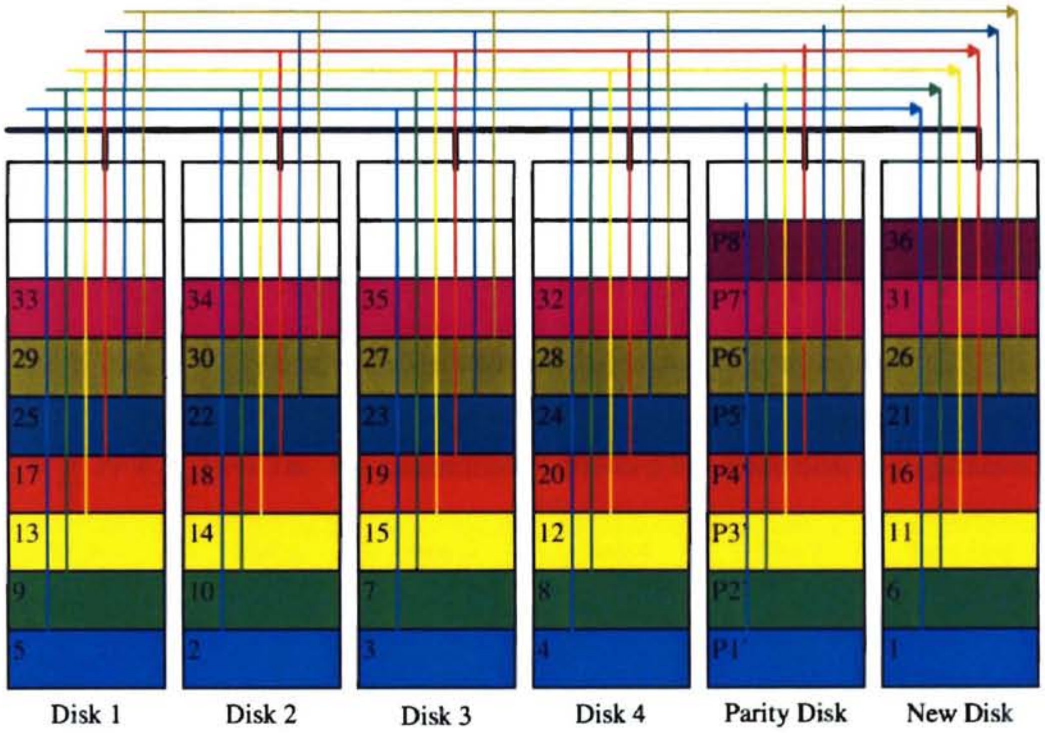


Figure 3-26 Diagonal Moving on RAID level 3 and level 4 with extended capacity.

3.2.4 Block-Interleaved Distributed-Parity - RAID level-5

For the interleaved distributed parity as in RAID level 5 as in Figure 3-27, we read the old parity stripe units in non-extended format first. Then we write the parity stripes with updated parity stripe units in extended format to all disks. Since there is no more than one stripe unit accessing from any disk, all the reads and writes are in parallel as stripes. The stripe units P1 to P7 are those distributed parity for RAID level 5.

On the extended disk capacity model, we need to have $S*\left(\frac{N-1}{N}\right)$ reads and writes. That is, because of the diagonal data stripe units allocation, there are $(N-1)$ stripe reads for every N stripes in a S stripes disk array as illustrated in Figure 3-27 and Figure 3-28. On the distributed hot spare disk configuration, there are S stripe read operations as well as write operations as in Figure 3-29 and Figure 3-30. The stripe units B1 to B7 organize the distributed hot spare disk for guaranteed backup function. The required time for completion algorithm in this level is $T_{level\_5*} = \sum_{1}^{s\left(N-1/N\right)} Tr + \sum_{1}^{s\left(N-1/N\right)} Tw + Tm$ for extended disk capacity configuration with one failed disk tolerance, and

$T_{level\_5*} = \sum_{1}^{S} Tr + \sum_{1}^{S} Tw + Tm$ for guaranteed distributed hot spare disk configuration.

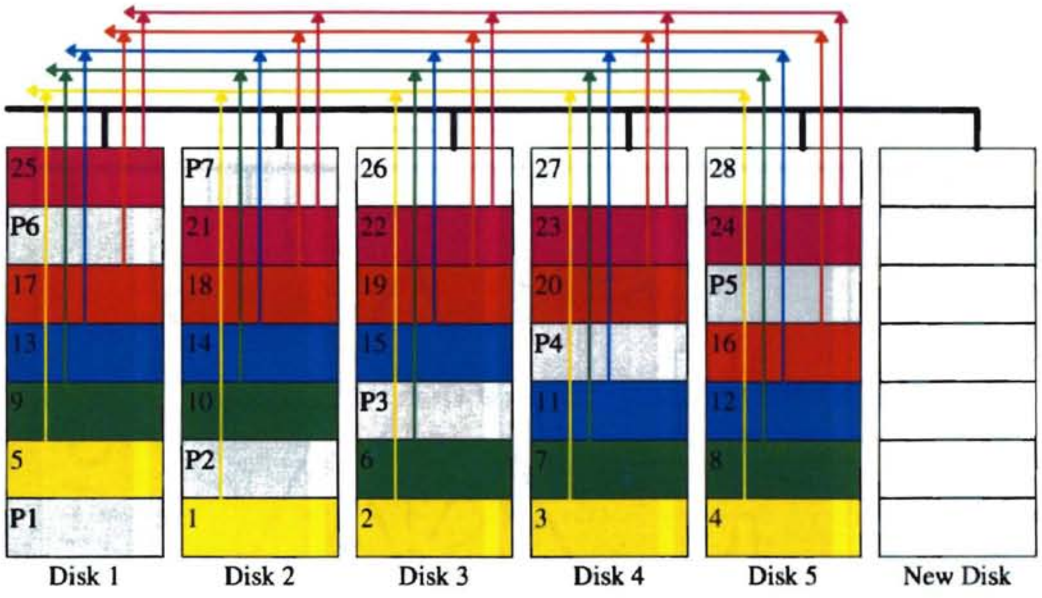


Figure 3-27 Diagonal Move on RAID level 5

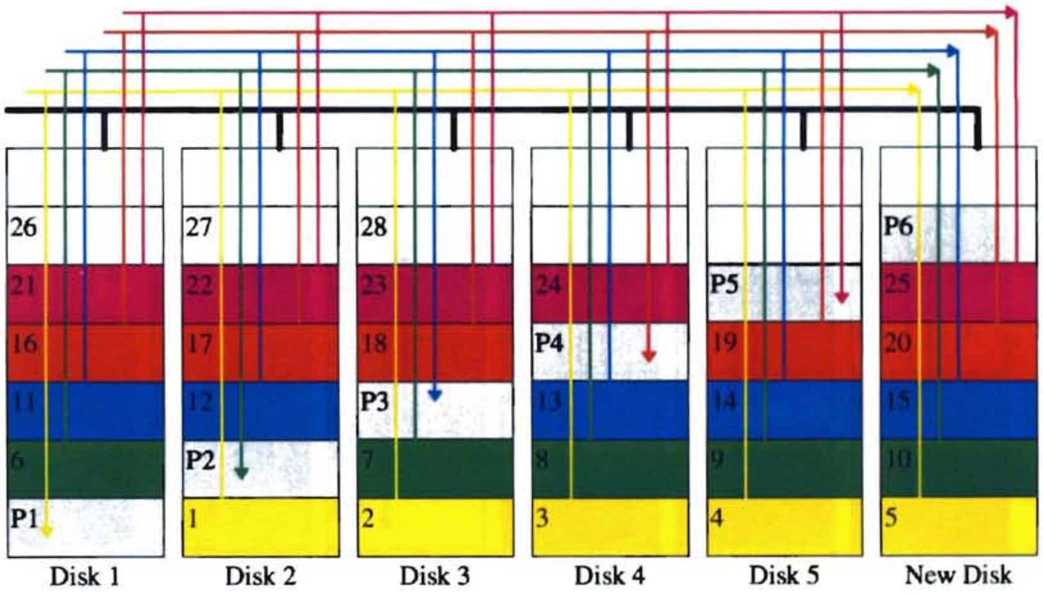


Figure 3-28 Diagonal Move on RAID level 5 with increased disk capacity

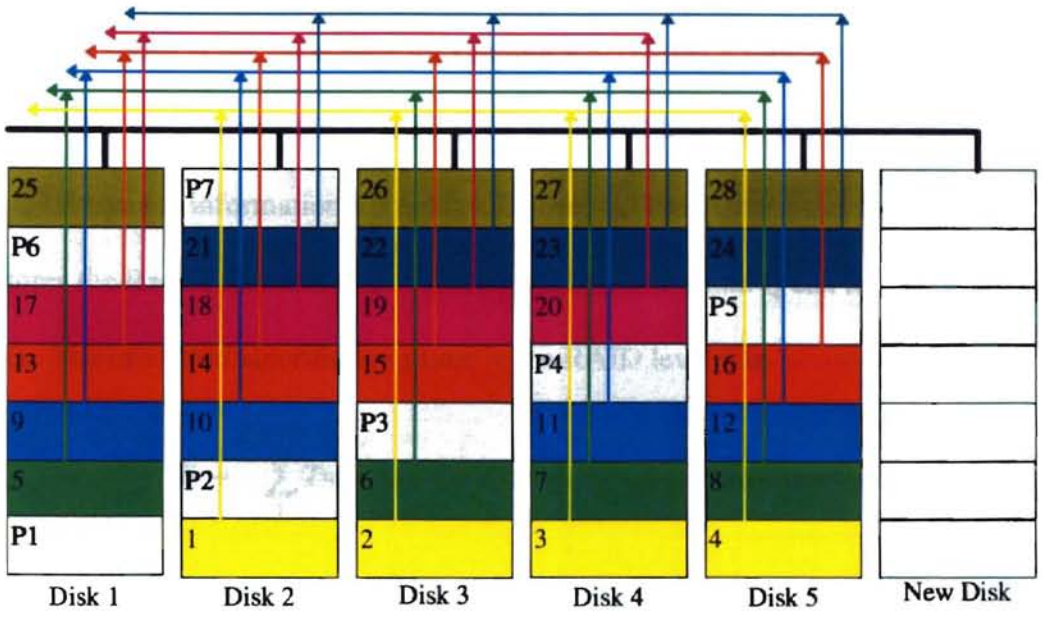


Figure 3-29 Diagonal Move on RAID level 5

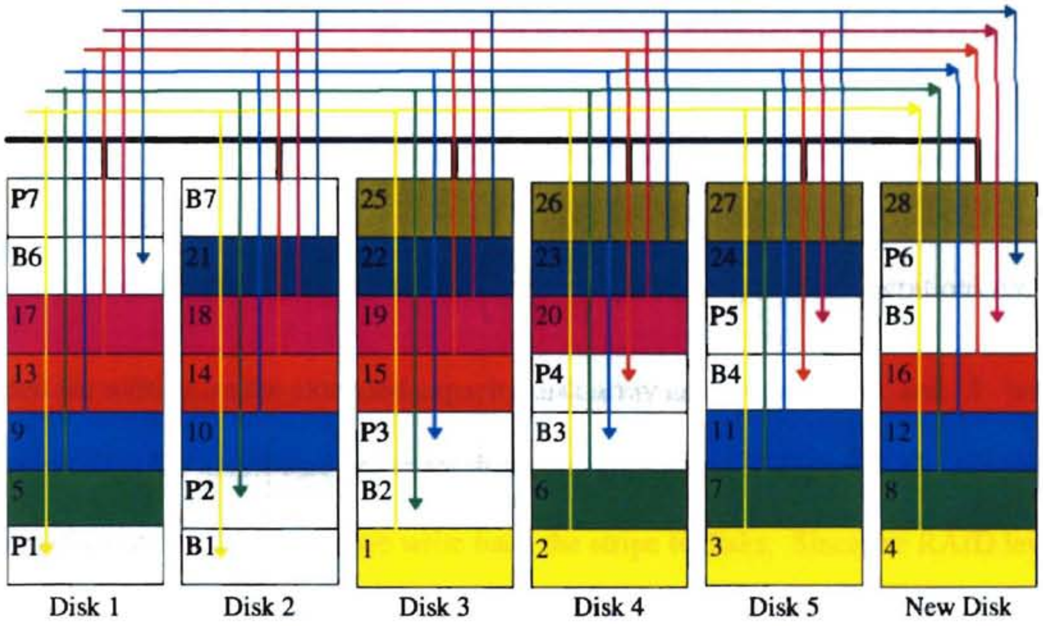


Figure 3-30 Diagonal Move on RAID level 5 with distributed backup disk

3.2.5 PQ Redundant - RAID level-6

The procedures for RAID level 6 are similar to RAID level-5, but two new disks need to be added at the same time for keeping RAID level 6 format. In Figure 3-31 to Figure 3-34, we illustrate the operations on the RAID level 6. The stripe units P1, Q1 to P7, Q7 are parity information for the RAID level 6. The P stores the parity code, and the Q stores the Reed-Solomom code. The combination of P and Q can recovery two failed disks. The estimated algorithm run time in the RAID level 6 is

$T_{level\_6^*} = \sum_{1}^{s(N-2/N)} Tr + \sum_{1}^{s(N-2/N)} Tw + Tm$ for extended disk system capacity option, and

$T_{level\_6^*} = \sum_{s=1}^{s(N-2/N)} Tr + \sum_{s=1}^{S} Tw + Tm$ for distributed hot spare disk option. Since each old parity stripe consists $\left(\frac{N-2}{N}\right)$ as partial of new parity stripe on added disk array, we need to have $S * \left(\frac{N-2}{N}\right)$ read operations to read the complete old disk array as in Figure 3-31 and Figure 3-33. Then $S * \left(\frac{N-2}{N}\right)$ parity stripe write operations are needed for writing on the extended capacity disk array as in Figure 3-34, and S writes are needed for the distributed hot spare disk configuration as in Figure 3-32. All parity information are updated while we write back the stripe to disks. Since the RAID level 6 already has the ability to recovery any two data disk fails, with this distributed hot spare disk configuration as in Figure 3-34, the RAID level 6 can tolerate up to four disks failure. In Figure 3-34, the stripe units B1 to B27 are the distributed hot spare disks.

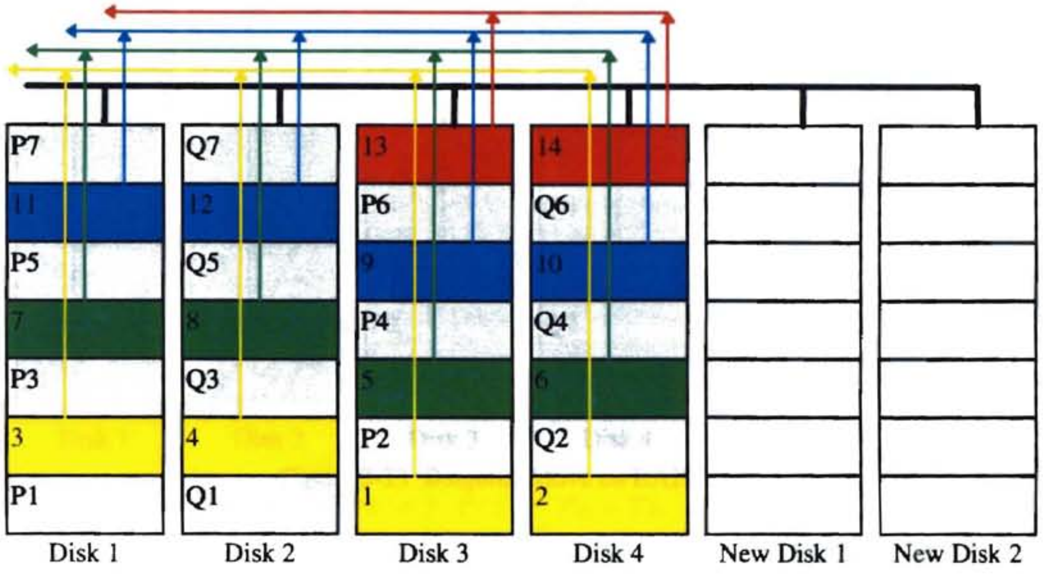


Figure 3-31 Diagonal Move on RAID level 6

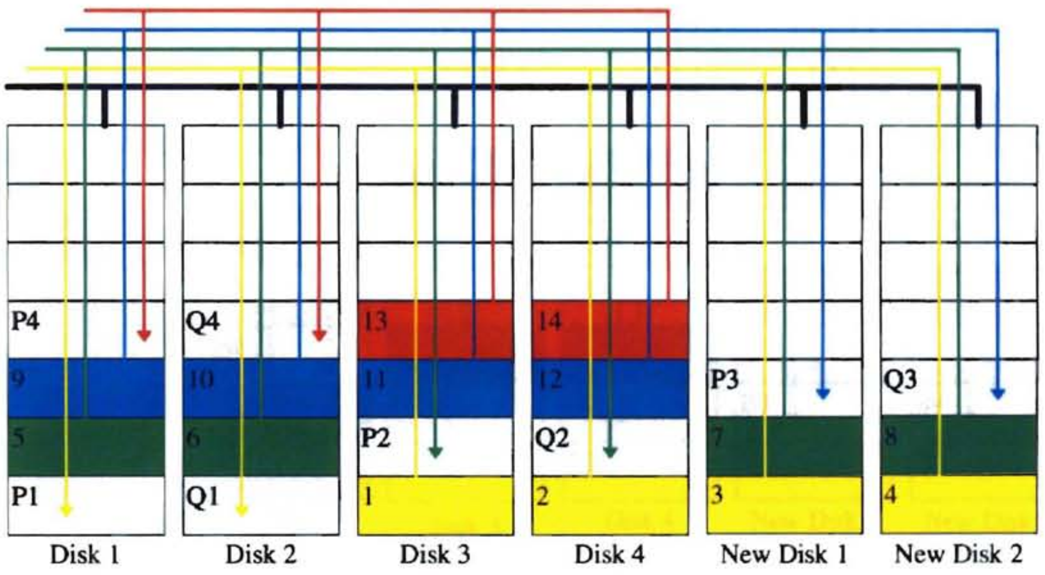


Figure 3-32 Diagonal Move on RAID level 6 with increased disk capacity.

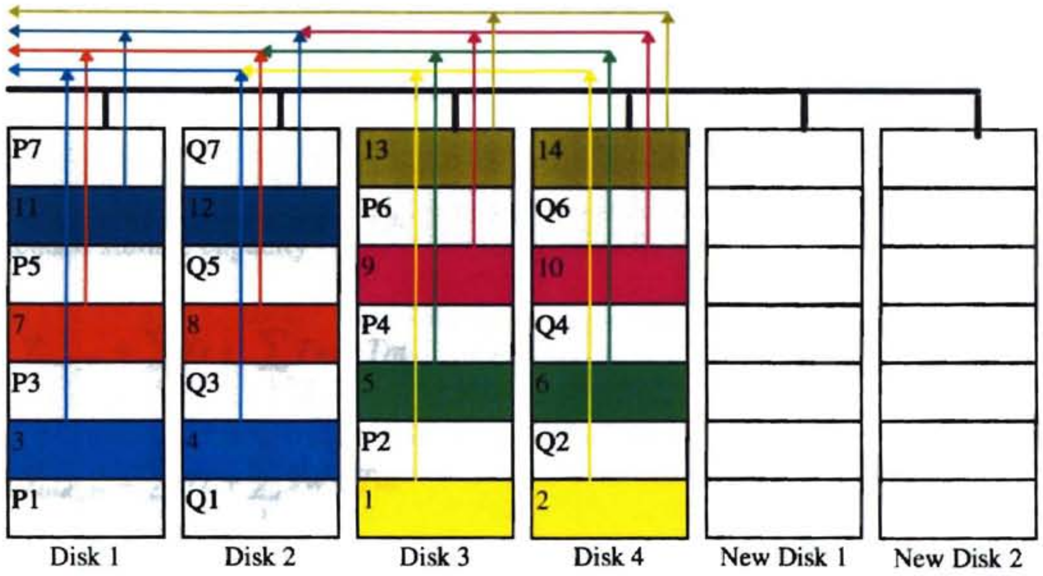


Figure 3-33 Diagonal Move on RAID level 6

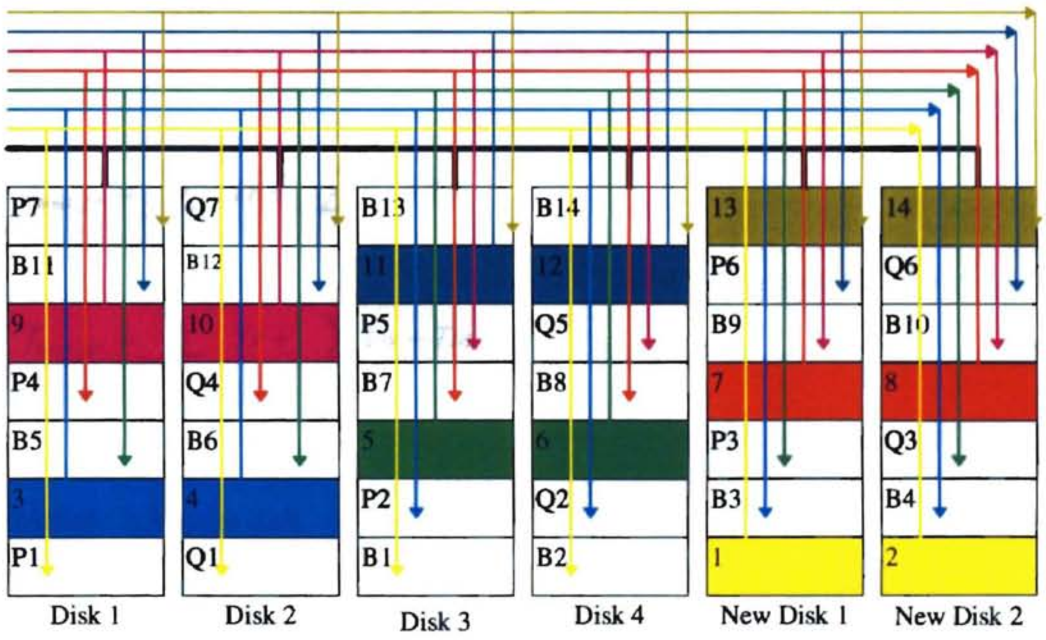


Figure 3-34 Diagonal Move on RAID level 6 with distributed backup disk

3.3 Summary of Diagonal Move

The diagonal move algorithm running time for both extended storage capacity and distributed hot spare disk are summarized below.

- Extended storage capacity

$$T_{level\_1*} = \sum_{1}^{S-1} Tr + \sum_{1}^{2(S-1)} Tw + Tm$$

$$T_{level\_2*} = \sum_{1}^{S} Tr + \sum_{1}^{S} Tw + Tm$$

$$T_{level\_3*} = \sum_{1}^{S/(N+1)} Tr + \sum_{1}^{N\left(S/_{N+1}\right)} Tw + \sum_{1}^{S} Tr + \sum_{1}^{S} Tw + Tm$$

$$T_{level\_4*} = \sum_{1}^{S/(N+1)} Tr + \sum_{1}^{N\left(S/_{N+1}\right)} Tw + \sum_{1}^{S} Tr + \sum_{1}^{S} Tw + Tm$$

$$T_{level\_5*} = \sum_{1}^{S\left(\frac{N-1}{N}\right)} Tr + \sum_{1}^{S\left(\frac{N-1}{N}\right)} Tw + Tm$$

$$T_{level\_6*} = \sum_{1}^{S\left(\frac{N-2}{N}\right)} Tr + \sum_{1}^{S\left(\frac{N-2}{N}\right)} Tw + Tm$$

- Distributed hot spare disk

$$T_{level\_1*} = \sum_{1}^{S/(N+1)} Tr + \sum_{1}^{2N\left(S/(N+1)\right)} Tw + Tm$$

$$T_{level\_2*} = \sum_{1}^{S} Tr + \sum_{1}^{S} Tw + Tm$$

$$T_{level\_3*} = \sum_{1}^{S/(N+1)} Tr + \sum_{1}^{N\left(S/_{N+1}\right)} Tw + Tm$$

$$T_{level\_4*} = \sum_{1}^{S/(N+1)} Tr + \sum_{1}^{N\left(S/_{N+1}\right)} Tw + Tm$$

$$T_{level\_5*} = \sum_{1}^{S} Tr + \sum_{1}^{S} Tw + Tm$$

$$T_{level\_6*} = \sum_{s=1}^{S\left(N-2/_{N}\right)} Tr + \sum_{s=1}^{S} Tw + Tm$$

CHAPTER 4

PERFORMANCE ANALYSIS

Unexpected write operations to the disks will happen while we are doing disk array data rearrangement if we don't terminate all users accessing the disks. Our *data locking with users priority* model can allow us running the moving algorithms without terminating users accessing the disks with minimum slowdown for users. While read operation is considered safe, locking data for atomic data write from user and algorithm is our solution in this experimental model.

4.1 Prelude

In this chapter, we analyze the interaction between the users and the moving algorithms. Our "disk data locking with users priority" methodology allows both users and algorithms operate on the disk array at the same time. It not only lets us effectively calculate the slowdown ratio, but also minimize the delay for users by giving users higher priority and allowing users to interrupt the disk array data moving algorithm at almost any moment. In the other words, the disk array data moving algorithm can utilize the users idle time for the disk data moving operations, and the users can interrupt the data moving operations unless the data is locked by the algorithm. Since the moving algorithm is running while users are still online, both moving algorithms and users will slow each other because they share the disks resources. In this chapter, we give a procedure which implement the "disk data locking with users priority" methodology. The effect of the data moving algorithm on the performance of user programs is also discussed. Simulation is used for performance analysis.

4.2 Simulation

Procedure given in the next page calculates the performance degradation for users versus algorithm. The ExecutionTimer is the overall execution time for the moving algorithm finishing the data migration operations on the disks system. The UserRunTime is a summation of UserTimer which is the duration of users' read and write operations on sharable data. The sharable (unlocked) data is the data which is not locked by the moving algorithm. By adding AlgorithmTimer, the AlgRunTime represents the actual disk data moving operation time for the whole disk array systems.

The moving algorithm needs to lock the users from writing data to avoid data inconsistency. Write on unlocked data and read on all disk data are allowed. To allow moving algorithm run without forcing users off-line, there are three rules to follow:

- If there is any read request from users on either unlocked or locked data, we simply delay the disk array data moving algorithm operations and let the users finish reading on those locked or unlocked data first.

- For the write requests from users on unlocked data set, we also simply delay the disk array data moving algorithm operations and let the user finish writing those unlocked data to the disk.
- For the write request from users on locked data which is currently used by disk array data moving algorithm, we are required to block the write access from users until disk array data moving algorithm completes data reallocation on that locked data.

Procedure to calculate algorithm runtime and users runtime is given below:

```
PROCEDURE BEGIN;
    /* set all timers to zero */
    InitializeTimer( AllTimers );
    ExecutionTimer( Start );
    REPEAT
        /* users can read all disk data without wait */
        IF ( UserReadRequest == TRUE ) THEN
            UserTimer( Start );
            { Users Read Operations; }
            UserTimer( Pause );
        ENDIF;
        /* users can write on unlocked disk data without wait */
        IF ( DataLocked == FALSE )&& ( UserWriteRequest == TRUE ) THEN
            UserTimer( Start );
```

```
                    { Users Write Operations; }

                    UserTimer( Pause );

                ENDIF;

                /* users have to wait while trying to write on locked disk data */

                IF ( DataLocked == TRUE ) && ( UserWriteRequest == TRUE ) THEN

                    UserWaitTimer( Start );

                ENDIF;

                /* algorithm run while user is not accessing disk */

                AlgorithmTimer( Start );

                { Moving Algorithm Operations; }

                AlgorithmTimer( Pause );

                UserWaitTimer( Pause );

                /* update all timers */

UserRunTime = UserRunTime + ( UserTimer( Pause ) - UserTimer( Start ) );

UserWaitTime = UserWaitTime + ( UserWaitTimer( Pause ) - UserWaitTimer( Start ) );

AlgRunTime = AlgRunTime + ( AlgorithmTimer( Pause ) - AlgorithmTimer( Start ) );

UNTIL ( MovingOperationFinished == TRUE );

ExecutionTimer( Stop );

/* get total execution time for rearranging disk array */

TotalExecTime  = ExecutionTimer( Stop ) - ExecutionTimer( Start );

ENDPROCEDURE;
```

4.3 Analysis

- Definition: ExecutionTime Te is the overall execution time for completing the disk array data moving algorithm reallocating all disk data. It begins when the disk array data moving algorithm starts running, and ends when the algorithm finished moving the last disk data set.

  $$\begin{aligned}\text{ExecutionTime } Te &= \text{AlgRunTime} + \text{UserRunTime} \\ &= Tar + Tur\end{aligned}$$

- Definition: UserWaitTime Tuw is the total waiting time that users are blocked by accessing locked data.

- Definition: UserRunTime Tur is the time interval that users were accessing on the disks. It is also the time that users stole from moving algorithm by interruption.

  $$\begin{aligned}\text{UserRunTime } Tur &= \text{ExecutionTime} - \text{AlgRunTime} \\ &= Te - Tar\end{aligned}$$

- Definition: AlgRunTime Tar is the moving algorithm running time when the disks are not used by users.

  $$\begin{aligned}\text{AlgRunTime } Tar &= \text{ExecutionTime} - \text{UserRunTime} \\ &= Te - Tur\end{aligned}$$

We define the Slowdown as the ratio of total operation time divided by actual operation time where $\text{Slowdown} = \dfrac{\text{Total Operation Time}}{\text{Actual Operation Time}}$

- Definition: User Slowdown is the ratio of the request round trip time to the request run time.

$$\text{User Slowdown} = \frac{\text{user request round trip time}}{\text{request running time}}$$

$$= \frac{\text{UserWaitTime} + \text{UserRunTime}}{\text{UserRunTime}}$$

$$= \frac{Tuw + Tur}{Tur}$$

The performance degradation range is between 1 as the best case in lower bound and no upper bound for the worst case.

$$1 \leq \text{User Slowdown} \equiv \frac{Tuw + Tur}{Tur}$$

- Definition: Algorithm Slowdown is the ratio of the overall execution time for completion to the actual data moving time.

$$\text{Algorithm Slowdown} = \frac{\text{time for completion data moving}}{\text{time for moving data}}$$

$$= \frac{\text{ExecutionTime}}{\text{AlgRunTime}}$$

$$= \frac{\text{UserRunTime} + \text{AlgRunTime}}{\text{AlgRunTime}}$$

$$= \frac{Te}{Tar}$$

$$= \frac{Tur + Tar}{Tar}$$

Such that the performance degradation has lower bound 1 as the best case and no upper bound as the worst case.

$$1 \leq \text{Algorithm Slowdown} \equiv \frac{Tur + Tar}{Tar}$$

- Definition: Probability of Locking Data Pl is the probability of disk data blocks being locked by moving algorithms. With locking data size Sl and total disk systems data size St, we have $Pl = \frac{Sl}{St}$ as the probability of disk data blocks being locked by moving algorithms.

- Definition: Probability of Hitting Locked Data Ph is the probability of users' write request blocks locked by moving algorithm. With users total write request size Su, total disk systems data size St, and data locked probability Pl. We have $Ph = \frac{Su}{St} * Pl$ or $Ph = \frac{Su * Sl}{St^2}$ as the probability of hitting locked data.

$$\begin{aligned} Ph &= P * \frac{Sl}{St} \\ &= \frac{Su}{St} * \frac{Sl}{St} \\ &= \frac{Su * Sl}{St^2} \end{aligned}$$

- Definition: Average Time Disk Access Time for User Ta is defined as $Ta = Ph * Tw + Th$ where the Tw is the time duration which users are locked and the Th is the time for users' data access.

When the disk size St has larger capacity, there is less chance to lock the data which users need. The performance of the moving algorithm improves when St increased and Su decreased. It also implies that the Straight Forward algorithm with less locked data could provide better accessing on the disk data than Diagonal Move algorithm , since the Su in the first algorithm is always smaller than the Su in the second algorithm. The simulation result of algorithms and users slowdown are shown as below in Figure 4-1. The Figure 4-2 shows the probability of hitting locked data. Simulated results of the Diagonal Move algorithm under different locking probabilities are shown in Figure 4-3 through Figure 4-13. In those figures, we can find the Average Time Disk Access Time for User, Ta , increases when the Tw, or Th increases. And, Ta decreases when Ph decreases.

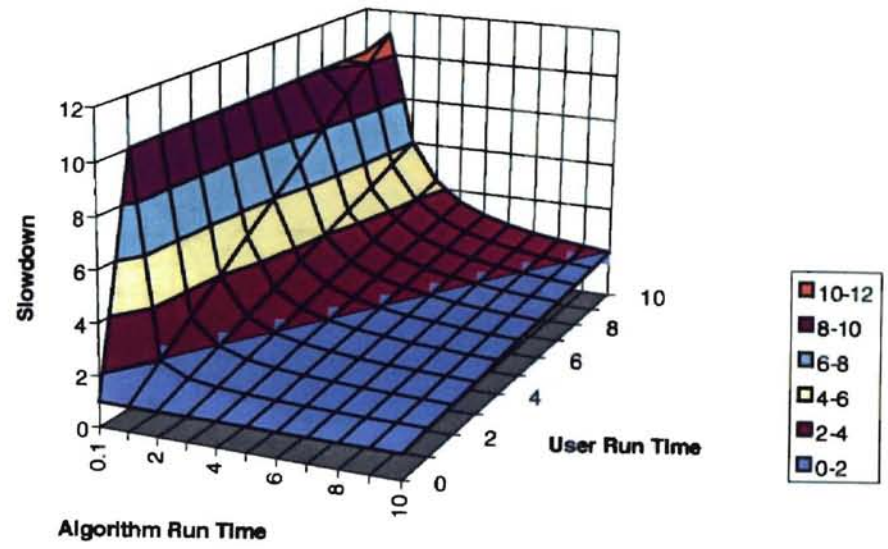


Figure 4-1 Algorithm slowdown.

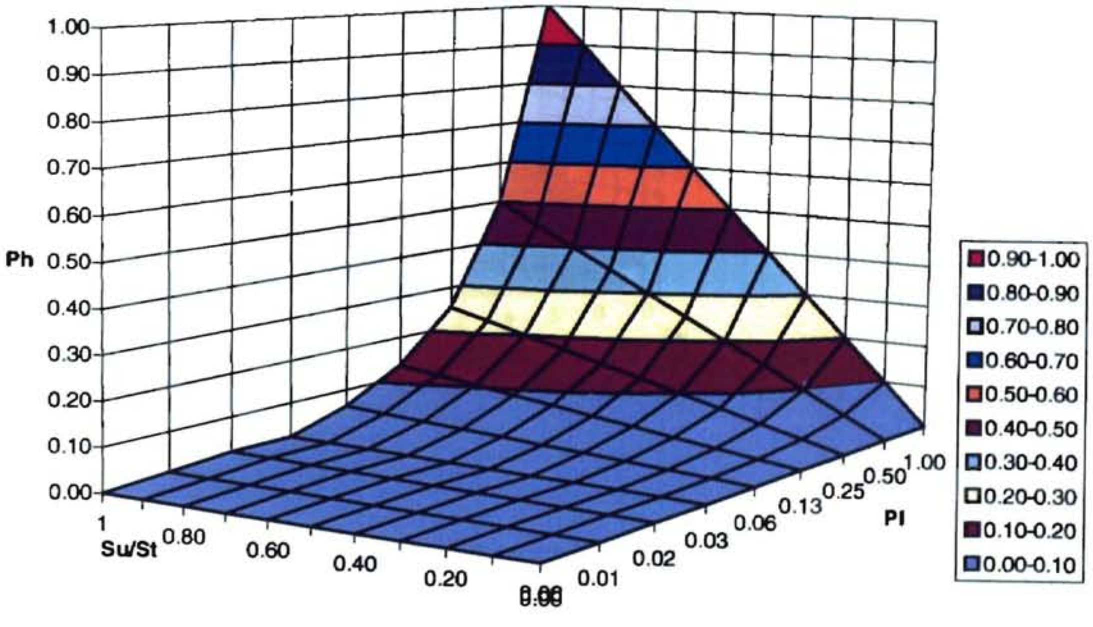


Figure 4-2 Probability of hitting locked data.

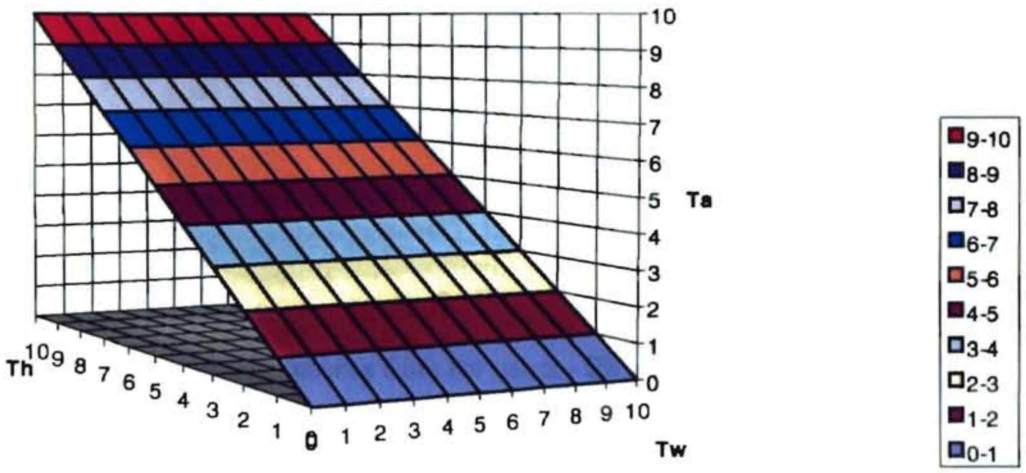


Figure 4-3 Average disk access time for users - Ph = 0.

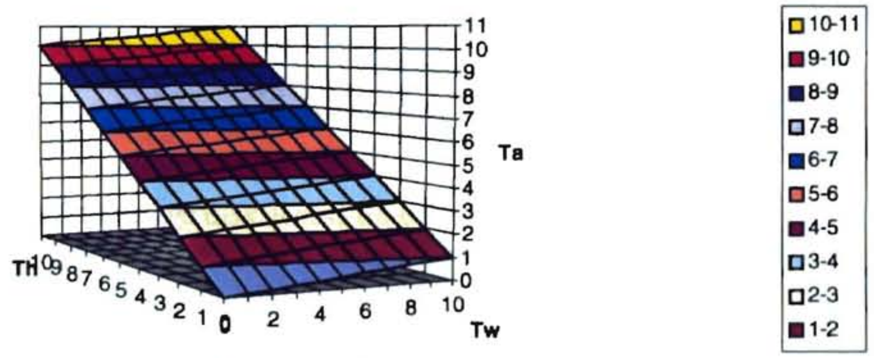


Figure 4-4 Average disk access time for users - Ph = 0.1.

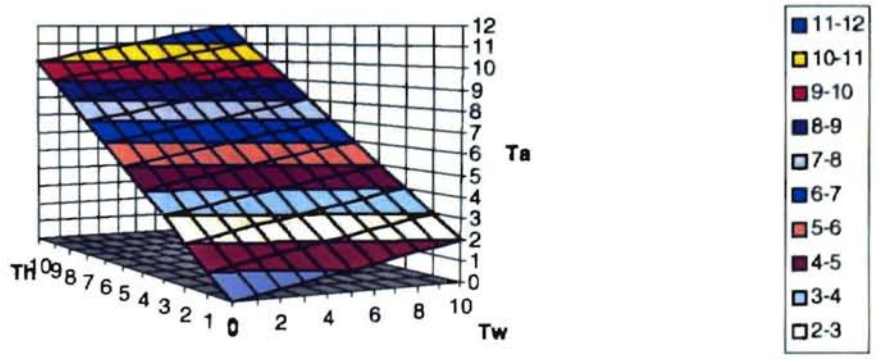


Figure 4-5 Average disk access time for users - Ph = 0.2.

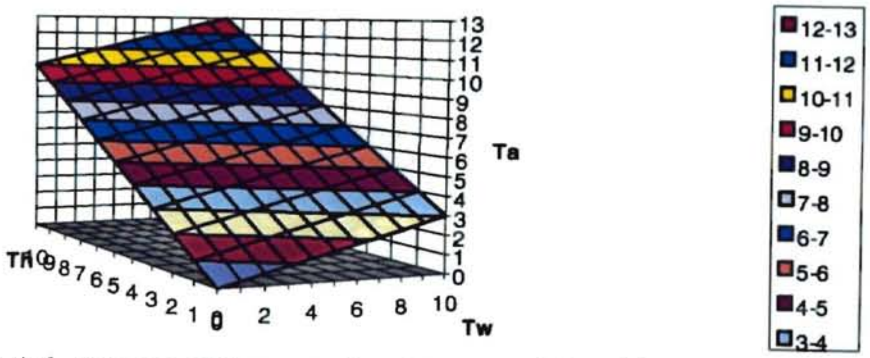


Figure 4-6 Average disk access time for users - Ph = 0.3.

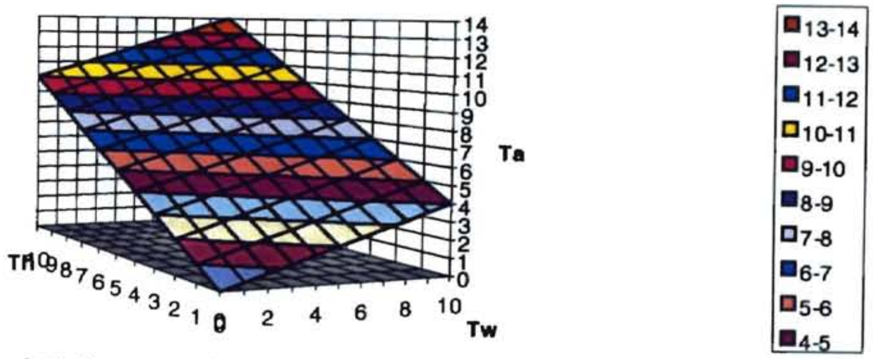


Figure 4-7 Average disk access time for users - Ph = 0.4.

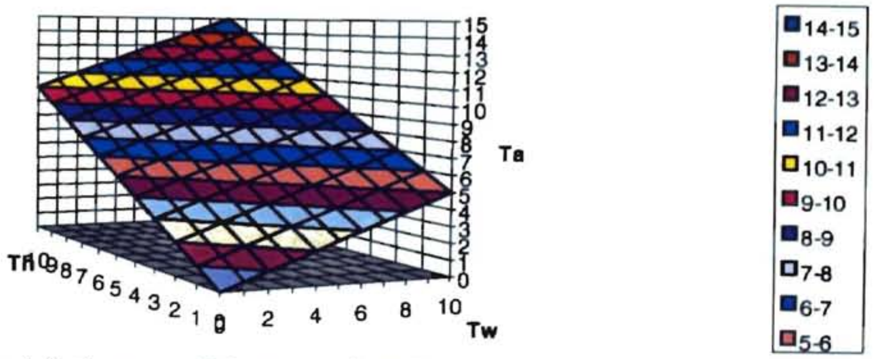


Figure 4-8 Average disk access time for users - Ph = 0.5.

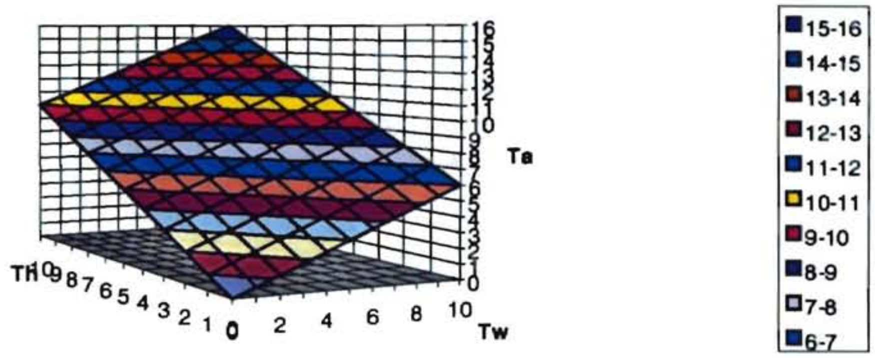


Figure 4-9 Average disk access time for users - Ph = 0.6.

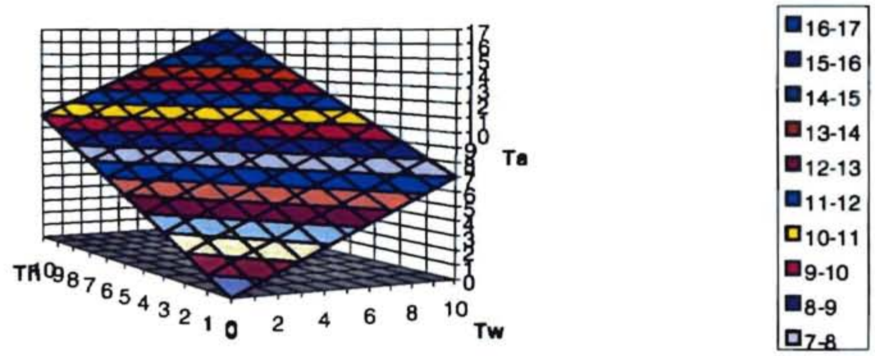


Figure 4-10 Average disk access time for users - Ph = 0.7.

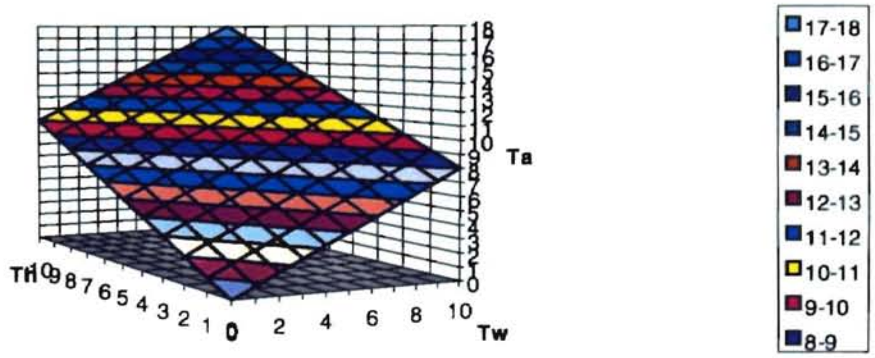


Figure 4-11 Average disk access time for users - Ph = 0.8.

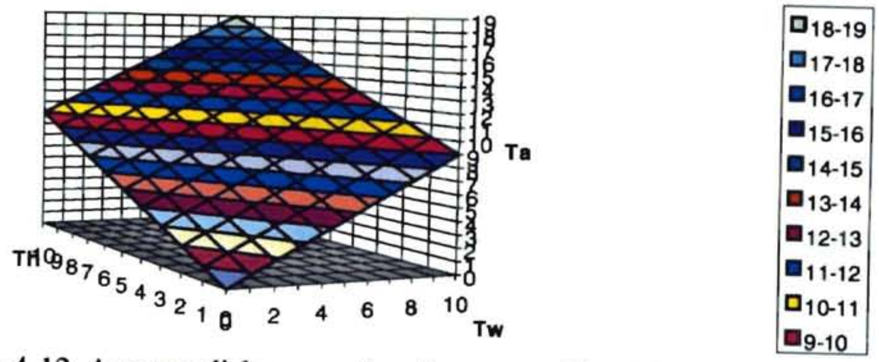


Figure 4-12 Average disk access time for users - Ph = 0.9.

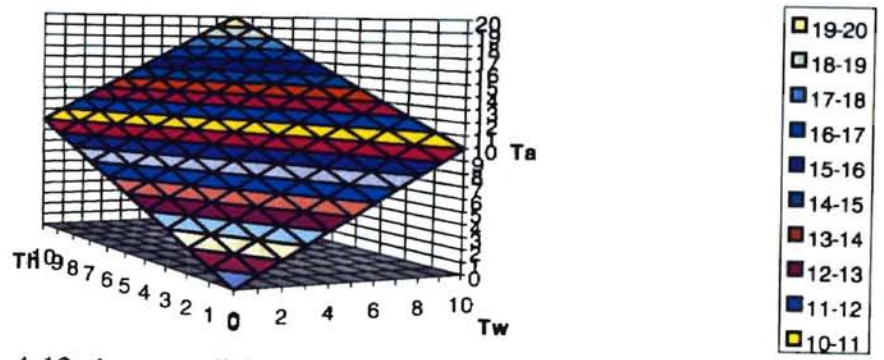


Figure 4-13 Average disk access time for users - Ph = 1.0.

CHAPTER 5

CONCLUSION

5.1 Summary

In this study, we build a capacity extendible disk array model with distributed hot spare disk option for all seven RAID levels. Two algorithms - the Straight Forward algorithm and the Diagonal Move algorithm - are developed. We also analyze performance of the algorithms. For different probabilities of hitting locked data, algorithm/user slowdown, and algorithm/user running time while rearranging the disk array data are examined.

5.2 Conclusion

Our model has the advantages of making disk array capacity extendible, taking shorter time for reallocating data to the disks, providing a solution for guaranteed hot spare disk problem, improving parallel I/O performance and migrating data while users are still on line. Without any doubt, parallel I/O processing can improve computer system performance dramatically. Though the straight forward algorithm takes longer time to complete the data migration task, it has the advantage of locking least number of data. Less locked disk data will let users use data from disks with less access restrictions(blocking). Users always have higher priority than moving algorithm to use disk data which will benefit the users because of reduced waiting time. With relatively low cost, the disk array can provide satisfactory high bandwidth for data transfer with robust reliability for important data. Our extendible capacity and distributed hot spare disk ideas can make the disk array even better in performance and reliability. Users have the choice of using the newly added disk for increase data storage space or using it as a hot spare (backup) disk.

GLOSSARY

Capacity. The maximum amount of data that a disk array can hold measured in bytes.

Capacity Extendible Disk Array. A capacity extendible disk array is a disk array model in which we can enlarge maximum amount of data in the disk array system by adding new disks without changing the data layout scheme for any RAID level.

Disk Array. A set of disks organized as an array for parallel access.

Data Redundancy. Redundant information stored in disks. Some of the schemes used for redundancy are mirroring, Hamming Code, and parity.

Data Transfer Rate. Amount of data transferred through the bus or network per second such as bits per second (BPS).

Degree of Parallelism. The maximum possible number of simultaneously running (disk) I/Os in a system.

Hamming Code. Hamming Code is a error detection and correction model. Hamming Code information are stored at disk positions 2^n where $0 \le n \le \lfloor \log_2 N \rfloor$ and N is the number of disks in the disk array for error detection and error correction.

Hot Spare Disk. A hot spare disk for a disk array system is a disk physically connected with other working disks for reduced MTTR, and not used. It replaces a failed disk automatically without manual intervention.

I/O Operating Rate. The number of I/O operations per second.

MTTR. Mean Time To Repair.

Read-Modify-Write. The disk array uses the *read-modify-write* if majority of the stripe units in a parity stripe is not changed. The old data stripe unit(s) (those data stripe units that need to be updated) and old parity stripe unit are read first, then the new parity stripe unit is computed and new data stripe unit(s), and the new parity stripe unit are written back to the disk array.

Reconstruct-Write. The *reconstruct-write* is used if the majority of the stripe units in a parity stripe need update. It will first read the rest of the old data (those data stripe units that we are not updating) and old parity stripe unit, then compute the new parity stripe unit and write the new data stripe units with new parity stripe unit back to disk array.

RAID. Redundant Arrays of Inexpensive Disks.

SLED. Single Large Expensive Disk.

Straight Forward Algorithm. It is a capacity extendible algorithm for disk array system. After a new disk is added, the disk array system is subject to compaction where all the stripe units are sequentially moved to its lowest available addressing space continuously across the disks system.

Stripe Unit. It is the interleaved component of a *stripe*. A group of *stripe units* can organize a *parity stripe* for failure recovery. A *stripe unit* which stores data is a *data stripe unit*. A *stripe unit* which stores parity information is called a *parity stripe unit*.

Stripe. It is the maximum number of *stripe units* collection in a row. Accessing the disk array in a *stripe* can maximize the parallelism of the disk array.

Parity Stripe. A group of *stripe units* in a row with common computed parity information.

BIBLIOGRAPHY

[Ande94] T.E. Anderson, D.E. Culler, D.A. Patterson. A Case for NOW. *IEEE Micro*, Vol.15, pages 54-64, February 1995.

[Ande95] T.E. Anderson, M.D. Dahlin, J.M. Neefe, D.A. Patterson, D.S. Roselli, R.Y. Wang. Serverless Network File Systems. *ACM Transactions on Computer Systems*, Vol.14, No.1, pages 41-79, February 1996.

[Arpa95] R.H. Arpaci, A.C. Dusseau, A.M. Vahdat, L.T. Liu, T.E. Anderson, D.A. Patterson. The Interaction of Parallel and Sequential Workloads on a Network of Workstations. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1995.

[Asam96] S. Asami, N. Talagala, T. Anderson, K. Lutz, D. Patterson. The Design of Large Scale, Do It Yourself RAIDs. *Technical report, University of California at Berkeley*, pages 1-30, 1996.

[Cao94] P. Cao, S.B. Lim, S. Venkataraman, J.Wilkes. The TickerTAIP Parallel RAID Architecture. *ACM Transactions on Computer Systems*, Vol.12, No.3, pages 237-269, August 1994.

[Chen93] P.M. Chen, E.K. Lee. Striping in a RAID Level 5 Disk Array. *Technical Report CSE-TR-181-93, Computer Science Department, University of California at Berkeley*, 1993.

[Chen93] P.M Chen, E.K. Lee, G.A. Gibson, R.H. Katz, T.E. Anderson. RAID: Hign-Performance, Reliable Secondary Storage. *ACM Computing Surveys*, pages 1-62, 1993.

[Chen90] P.M. Chen, G.A. Gibson, R.H. Katz, D.A. Patterson. An Evaluation of Redundant Arrays of Disks using an Amdahl 5890. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1990.

[Cher91] A.L. Chervenak, R.H. Katz. Performance of a Disk Array Prototype. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1991.

[del94] J.M. del Rosario, A.N. Choudhary. High-Performance I/O for Massively Parallel Computers. *IEEE Computer*, pages 58-68, March 1994.

[Doug89] F. Douglis, J. Ousterhout. Log-Structures File Systems. *IEEE COMPCON89 34th IEEE Computer Society International Conference.* Pages 124-129, February 1989.

[Feit95] D.G. Feitelson, P.F. Corbett, S.J. Baylor, Y. Hsu. Parallel I/O Subsystems in Massively Parallel Supercomputers. *IEEE Parallel and Distributed Technology.* Vol.3, No.3, pages 33-45, Fall 1995.

[Frie96] M.B. Friedman. RAID Keeps Going and Going. *IEEE Spectrum*, pages 73-79, April 1996.

[Gray90] J. Gray, B. Horst, M. Walker. Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput. *Proc. of 16th VLDB*, pages 148-159, 1990.

[Hart94] J.H. Hartman. The Zebra Striped Network File System. *Ph.D. dissertation, Computer Science Department, University of California at Berkeley*, 1994.

[Hart92] J.H. Hartman, J.K. Ousterhout. Zebra: A Striped Network File System. *USENIX Workshop on File Systems*, May 1992.

[Hart95] J.H. Hartman, J.K. Ousterhout. The Zebra Striped Network File System. *ACM Transactions on Computer Systems*, Vol.13, No.3, pages 274-310, August 1995.

[Katz89] R.H. Katz, G.A. Gibson, D.A. Patterson. Disk System Architecture for High Performance Computing. *IEEE Computer*, pages 1842-1858, December 1989.

[Keet93] K. Keeton, R.H. Katz. The Evaluation of Video Layout Strategies on a High-Bandwidth File Server. *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video*, November 1993.

[Kim86] M.Y. Kim. Synchronized Disk Interleaving. *IEEE Transactions on Computers*, Vol.35, No.11, pages 978-988, November 1986.

[Lee92] E.K. Lee. Software and Performance Issues in the Implementation of a RAID prototype. *Technical Report UCB/CSD 90/573, Computer Science Department, University of California at Berkeley,* March 1992.

[Lee93] E.K. Lee. Performance Modeling and Analysis of Disk Arrays. *Ph.D. dissertation, Computer Science Department, University of California at Berkeley*, 1993.

[Lee92] E.K. Lee, P.M. Chen, J.H. Hartman, A.L. Chervenak Drapeau, E.L. Miller, R.H. Katz, G.A. Gibson, D.A. Patterson. RAID-II: A scaleable Storage Architecture for High-Bandwidth Network File Service. *Proceedings of 21st International Symposium on Computer Architecture*, 1992.

[Lee93] E.K. Lee, R.H. Katz. The Performance of Parity Placement in Disk Arrays. *IEEE Transactions on Computers*, Vol.42, No.6, pages 651-664, June 1993.

[Mead89] W.E. Meador. Disk Array Systems. *IEEE COMPCON89 34th IEEE Computer Society International Conference.* pages 143-146, February 1989.

[Ng89] S. Ng. Some Design Issues of Disk Arrays. *IEEE COMPCON89 34th IEEE Computer Society International Conference.* pages 137-142, February 1989.

[Patt88] D.A. Patterson, G. Gibson, R.H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). *ACM SIGMOD Conference on Management of Data*, pages 109-116, June 1988.

[Rose91] M. Rosenblum, J.K. Outsterhout. The Design and Implementation of a Log-Structures File System. *ACM Transactions on Computer Systems*, Vol.10, No.1, pages 26-52, February 1992.

[Schu89] M. Schulze, G. Gibson, R. Katz, D. Patterson. How Reliable is a RAID? *IEEE COMPCON89 34th IEEE Computer Society International Conference.* Pages 118-123, February 1989.

[Skie95] T.A. Skeie, M.R. Rusnack. HP Disk Array: Mass Storage Fault Tolerance for PC Servers. *Hewlett-Packard Journal*, Vol.46, No.3, pages 71-78, June 1995.

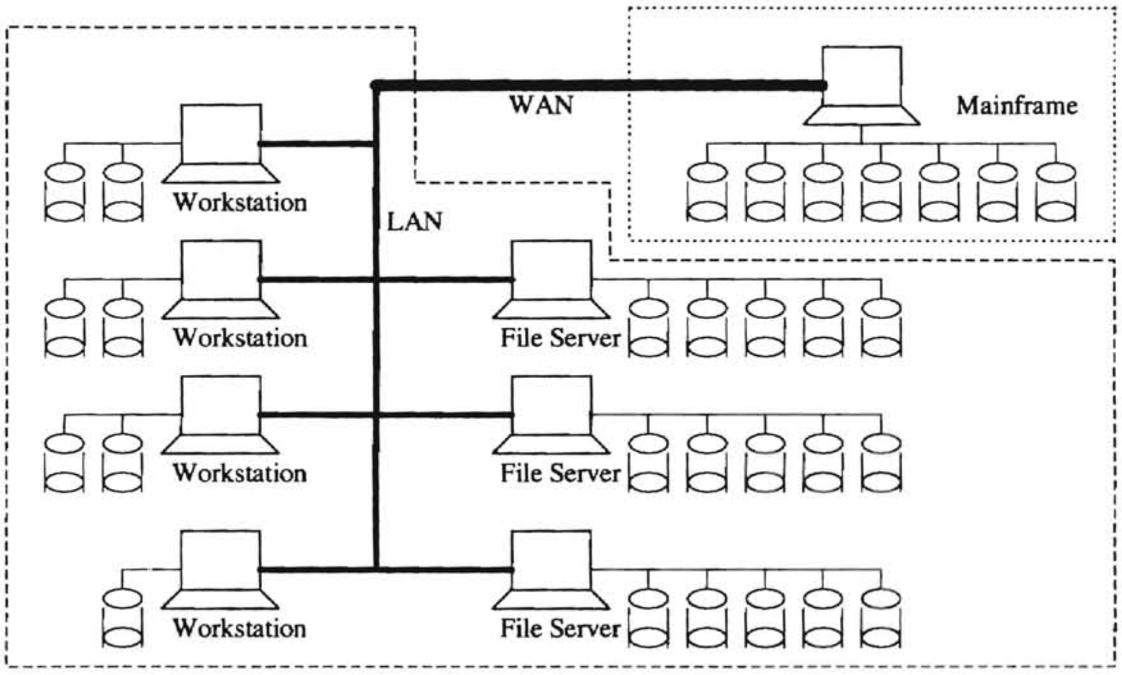
[Wilk96] J. Wilkes, R. Golding, C. Staelin, T. Sullivan. The HP AutoRAID Hierarchical Storage System. *ACM Transactions on Computer Systems*, Vol.14, No.1, pages 108-136, February 1996.

[Wang93] R.Y. Wang, T.E. Anderson. xFS: A Wide Area Mass Storage File System. *Proc.4th Workshop on Workstation Operating Systems*, pages 71-78, October 1993.

APPENDIX

APPENDIX I

- RAID applied in local workstations and across networks.



VITA

Tseng, Cheng-Yuan

Candidate for the Degree of

Master of Science

Thesis: THE DESIGN AND ANALYSIS OF CAPACITY EXTENDIBLE DISK ARRAY SYSTEM: THE DIAGONAL-MOVE ALGORITHM

Major Field: Computer Science

Biographical Data:

Personal Data: Born in Taipei, Taiwan on February 23, 1969.

Education: Received Bachelor of Science in Computer Science from the Oklahoma State University, Stillwater, Oklahoma in 1994. Completed the requirements for the Master of Science degree in Computer Science at the Oklahoma State University in May 1997.

Experience: Network System Administrator and Analyst, Oklahoma State University, Department of Financial Aid, 1996 to present.