ABSTRACT

PERFORMANCE EVALUATION OF VARIOUS DATA ALLOCATION METHODS IN A HETEROGENEOUS DISK ARRAY ARCHITECTURE

by
Bogdan Alexandru Branzoi

Dataset attributes, such as data availability levels and access patterns, make their mapping to certain RAID levels more desirable than others. On the other hand, it is not economically viable for an installation to acquire multiple disk arrays to satisfy diverse data storage requirements. A Heterogeneous Disk Array (HDA) architecture is proposed, which allows device heterogeneity as well as RAID level heterogeneity. In other words, various disks of different types can be incorporated in a single HDA and multiple RAID schemes can coexist in the same array. The goal of this architecture is to utilize the resources of all its disks to the maximum possible extent by using appropriate RAID levels to meet the varying availability requirements for different applications. An improved best-fit allocation algorithm is proposed and various data allocation methods are tested against it.

In an HDA system, each new object is associated with an appropriate RAID level and the allocation is carried out in a way to keep disk bandwidth and capacity utilizations balanced. The data structures of the HDA architecture are described and the flowcharts for the most frequent operations are depicted. Then a data allocation algorithm is formulized and a possible solution is given. Finally, the HDA architecture is prototyped based on the DASim simulation toolkit developed at NJIT and comparison results of various data allocation algorithms are presented.

PERFORMANCE EVALUATION OF VARIOUS DATA ALLOCATION METHODS IN A HETEROGENEOUS DISK ARRAY ARCHITECTURE

by
Bogdan Alexandru Branzoi

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Masters of Science in Computer Science

Department of Computer Science

January 2006

APPROVAL PAGE

PERFORMANCE EVALUATION OF VARIOUS DATA ALLOCATION METHODS IN A HETEROGENEOUS DISK ARRAY ARCHITECTURE

Bogdan Alexandru Branzoi

Dr. Alexander Thomasian, Dissertation Advisor — Date
Professor of Computer Science, NJIT

Dr. David Nassimi, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Alexandros Gerbessiotis, Committee Member — Date
Associate Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Bogdan Alexandru Branzoi

Degree: Masters of Science

Date: January 2006

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2006

- Bachelor of Science in Computer Science,
  Rutgers, The State University of New Jersey, New Brunswick, NJ, 1999

Major: Computer Science

To my wife

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Alexander Thomasian, who not only served as my research supervisor, providing valuable and countless resources, insight, and intuition, but also constantly gave me support, encouragement, and reassurance. Special thanks are given to Dr. David Nassimi and Dr. Alexandros Gerbessiotis for participating in my committee.

Many of my fellow graduate students in the Integrated Systems Research Laboratory are deserving of recognition for their support. Special thanks go to Gang Fu and Chunqi Han for their great previous work and help with the DASim simulation toolkit.

I also wish to thank my brother for his advice and encouragement throughout the years.

Finally, I would like to thank my wife, Jennifer, for her advice, consistent love, and support throughout my graduate work.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Objective

The objectives of this thesis are to propose a Heterogeneous Disk Array Architecture, to explain the motivation behind it, and to evaluate the performance of various data allocation methods on this type of architecture, ultimately choosing the best one for further experiments.

The Heterogeneous Disk Array architecture (HDA) was proposed as an example of a self-managed storage system with a very valuable characteristic, *heterogeneity*. This characteristic differentiates the proposed HDA architecture from Redundant Arrays of Independent Disks (RAID)[1] architectures. The heterogeneity of the proposed architecture comes from allowing various disks of different capacity, bandwidth and make, as well as various different RAID levels, to coexist on a disk array. The main purpose of this type of architecture is to automatically and efficiently utilize the resources of all the disks to the maximum extent possible, while choosing the appropriate RAID level that would meet the requirements of various applications.

1.2 Background Information

This section describes the structure and organization of modern hard drives and it gives a brief overview of the Redundant Arrays of Independent Disks. Knowledge of the structures and layouts of these architectures is assumed in the following chapters.

1.2.1 Hard Disk Technology

In Figure 1.1, the structure of a typical hard drive is depicted.[2] A disk drive consists of a stack of *platters* mounted on a common *spindle*. Each platter has 2 sides, both coated with a magnetic material. The platters rotate at a constant velocity, and the rotational speed or velocity is typically measured in revolutions per minute or RPM*s*. Each platter has a corresponding read/write head for each of its surfaces mounted at the end of a disk arm. Disk arms are mounted to a common shaft called an *actuator*. Applying a directional current to a positioning motor causes the actuator to rotate small distances in either direction. Rotating the actuator causes the disk heads to move together in a radial motion along the platters, thus allowing access to a radius spanning most of the coated surface of each platter. However, only one head is active at any given time. The reason for this is that it is nearly impossible to position two heads at the exact same time on corresponding tracks, due mostly to thermal variations of the disk arms and platters.

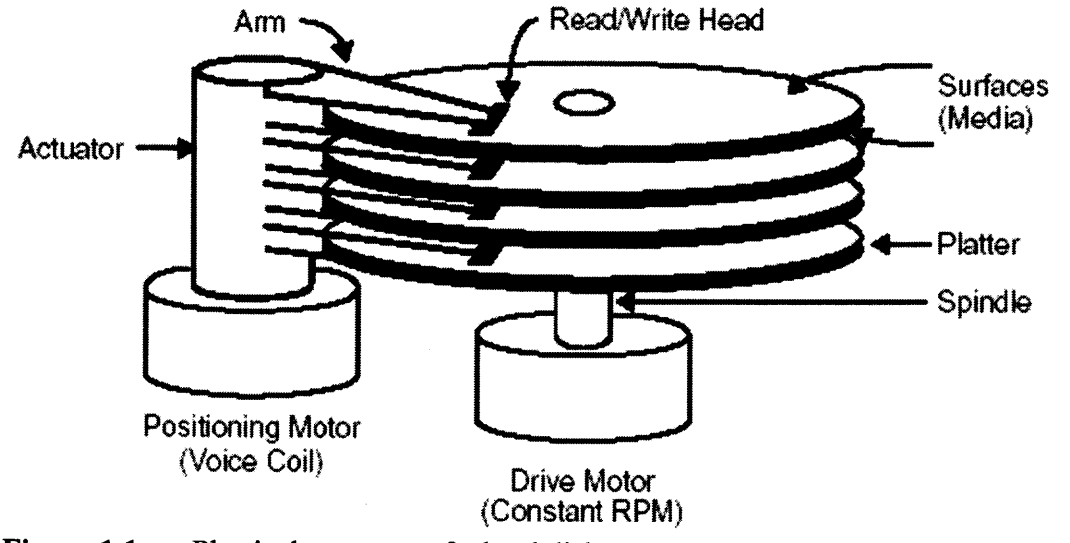

Figure 1.1 Physical structure of a hard disk.

The data stored on the hard disk is organized into the following components: *sectors, tracks, and cylinders*. A *sector* is a block of sequential user data (almost always 512 bytes) and is considered to be the smallest unit that can be read from or written to the disk. A header area in front of each sector contains sector identification and clock synchronization information, and a trailer area contains an error correcting code computed over the header and data. A *track* is made up of a set of sectors on a single platter surface at a constant radial distance from the spindle. A set of tracks with the same radius constitutes a *cylinder*. All the sectors in a hard disk are numbered sequentially starting from 0 as block addresses and constitute a linear address space to the user. These concepts are illustrated in Figure 1.2.[2]

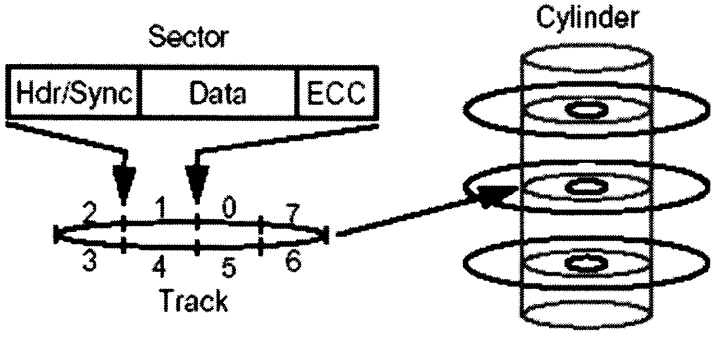

Figure 1.2 Grouping data into sectors, tracks, and cylinders.

When trying to access a block of data, the control mechanics of a hard disk move the actuator such that the disk heads are correctly placed over the right cylinder. After waiting for the desired data to rotate under the read/write heads, the sought after sectors are read or written. The time that it takes to move the actuator is called *seeking time* and it is usually around 1 to 15 ms depending on the seek distance. For each user request, the actuator must first *seek* to the indicated cylinder and then the disk must *rotate* to the start of the requested data. The combination of these two operations is referred to as

positioning the disk heads. Sometimes the rotational latency can be completely eliminated. This special case occurs when a user request needs to access an entire track of data and, instead of waiting until the first sector rotates under the heads to begin the operation, it starts by reading or writing the data in the order the sectors pass under the heads. This special case is called *zero-latency* operation and can be extended to include the case where the access spans only part of a track.

It should be noted that the tracks near the outside of each surface have greater circumference than those closer to the spindle. Therefore, a technique called *zoned bit recording (ZBR)* takes advantage of such a placement and stores more sectors per track in the outer cylinders. This technique groups sets of 50 to 200 adjacent cylinders into zones with the number of sectors per track being constant within each zone but successively larger in the outer zones than the inner.

Figure 1.3 illustrates the assignment of sequential data to sectors, tracks, and cylinders.[2] As shown in the figure, sequential data starts at sector zero and proceeds around to the end of the track. It then moves to the next track and continues in this manner until it reaches the end of the cylinder, at which point it moves to the next cylinder and starts again. As shown in Figure 1.3, a rotational distance equal to one sector is skipped when crossing a track boundary (in this example, moving from sector 7 to 8). Also it should be noted that two sectors are skipped when crossing a cylinder boundary (moving from sector 23 to 24 in this example). The skipped distances are called the *track skew* and *cylinder skew*. This method of laying out the data is necessary in order to allow the hard disk control electronics to have time to reposition the actuator when a user access spans a track or cylinder boundary. The track skew is shorter than the cylinder

skew because only fine adjustments are necessary when switching to a new track within one cylinder, whereas switching to a new cylinder requires the actuator to be moved one full cylinder width and then fine-adjusted over the new track. Typical values for track and cylinder skew in current technology are about 0.5 and 1.5 ms, respectively.

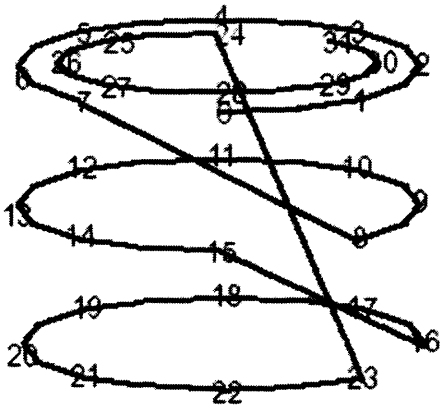

Figure 1.3 Sequential sector layout.

1.2.2 Brief Overview of RAID Architecture

There has been a steady and steep increase in the performance of computer processors in the past ten to twenty years. And with such a performance increase, comes the need for higher I/O bandwidth availability. However, the performance of disk drives has not been able to keep up with the fast pace at which processors have improved. According to Amdahl's law[3], the overall performance of the computer systems is limited by the performance of the I/O subsystem. Hence, the need to use parallelism in the storage subsystem to meet the increasing demands for I/O bandwidth.

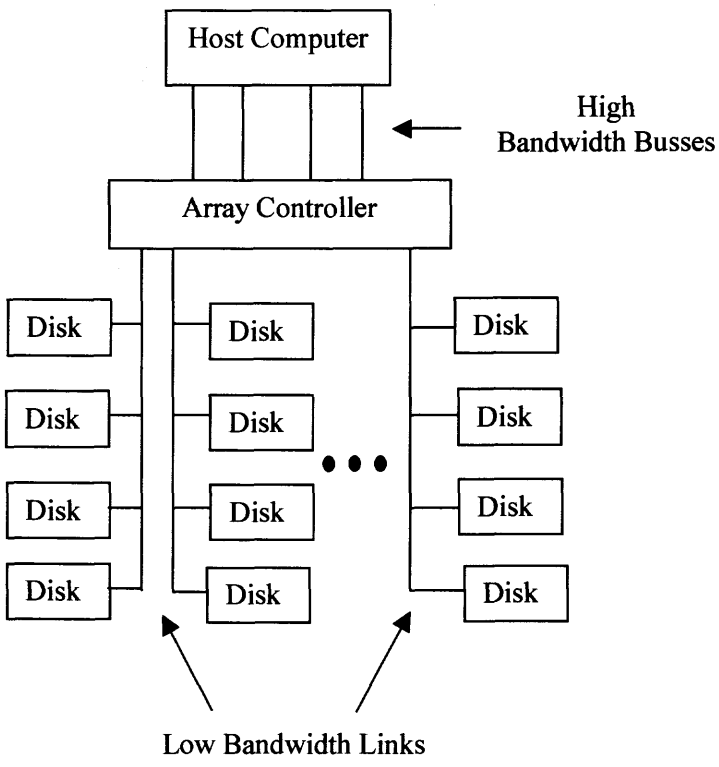

Figure 1.4 Disk-array architecture.

The disk-array architecture used in today's systems connects the disks via low-bandwidth links to an array controller,[4] which in turn is connected via high-bandwidth parallel buses to the host computer.[5] The array controller is responsible for the majority of all system-related activity, including maintaining address mapping and redundant information, controlling individual disks, and recovering from disk failures and executing transfers. It also provides a very convenient linear address space to the host computer. The mapping of the linear addresses to the individual disk addresses is performed by the array controller and is referred to as the *data layout*.

The core concept of disk arrays is *striping* or breaking up consecutive units of user data across the disks that make up the array.[1,6,7] The array controller exports a linear

address space to the host computer and striping is responsible for breaking up this linear space into *striping units* of a constant size. Consecutive units are then assigned to consecutive disks. There are some great benefits with striping, such as automatic load balancing in concurrent workloads and high bandwidth for large sequential transfers by a single process.

Disk arrays are usually classified into five types, RAID levels 1 through 5, based on the organization of redundant information and the layout of user data on the disks.[1] This terminology has gained wide acceptance in the storage community and will be used throughout this work. Although not part of the original RAID levels classification, RAID level 0 is often used to indicate a non-redundant disk array. RAID level 1, also termed mirroring or shadowing, is achieved by grouping the disks into mirror pairs and storing one copy of each data block or striping unit on each of the disks in the pair. In RAID level 2, the array disks are divided into data disks and check disks. Bit or byte striping is used across the data disks while the Hamming error correcting code[8] is used in the check disks. RAID level 3 uses bit or byte interleaved parity, where the data is stripped across the data disks while a single parity disk stores the cumulative exclusive-or over the corresponding bits on the data disks. RAID level 4 is very similar to RAID level 3 except that it uses block interleaved parity instead of bit or byte interleaved parity. The size of the block could be 32 KB or larger. RAID5 uses a rotated block interleaved parity, with the parity blocks distributed over all disks. Figure 1.5 illustrates RAID level concepts.

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
0	D_0	D_1	D_2	D_3	D_4	D_5
1	D_6	D_7	D_8	D_9	D_{10}	D_{11}
2	D_{12}	D_{13}	D_{14}	D_{15}	D_{16}	D_{17}
3	D_{18}	D_{19}	D_{20}	D_{21}	D_{22}	D_{23}

(a) RAID level 0: Non-redundant.

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
0	D_0	D_0	D_1	D_1	D_2	D_2
1	D_3	D_3	D_4	D_4	D_5	D_5
2	D_6	D_6	D_7	D_7	D_8	D_8
3	D_9	D_9	D_{10}	D_{10}	D_{11}	D_{11}

(b) RAID level 1: Mirroring.

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4 Disk 5 Disk 6
0	d_0	d_1	d_2	d_3	h_{0-3}
1	d_4	d_5	d_6	d_7	h_{4-7}
2	d_8	d_9	d_{10}	d_{11}	h_{8-11}
3	d_{12}	d_{13}	d_{14}	d_{15}	h_{12-15}

(c) RAID level 2: Hamming-code ECC.

Figure 1.5 RAID levels 0 through 5. Parity blocks have been highlighted while the data blocks make up the rest. "D" represents a block of user data, "d" a bit or byte of user data, "h_{x-y}" a Hamming code computed over user data bits/bytes x through y, "p_{x-y}" a parity (exclusive-or) bit/byte computed over data blocks x through y, and "P_{x-y}" a parity block over user data blocks x through y.

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
0	D_0	D_1	D_2	D_3	D_4	p_{0-4}
1	D_5	D_6	D_7	D_8	D_9	p_{5-9}
2	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	p_{10-14}
3	D_{15}	D_{16}	D_{17}	D_{18}	D_{19}	p_{15-19}

(d) RAID level 3: Byte-Interleaved Parity.

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
0	D_0	D_1	D_2	D_3	D_4	P_{0-4}
1	D_5	D_6	D_7	D_8	D_9	P_{5-9}
2	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	P_{10-14}
3	D_{15}	D_{16}	D_{17}	D_{18}	D_{19}	P_{15-19}
4	D_{20}	D_{21}	D_{22}	D_{23}	D_{24}	P_{20-24}
5	D_{25}	D_{26}	D_{27}	D_{28}	D_{29}	P_{25-29}

(e) RAID level 4: Dedicated Block Parity.

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
0	D_0	D_1	D_2	D_3	D_4	P_{0-4}
1	D_6	D_7	D_8	D_9	P_{5-9}	D_5
2	D_{12}	D_{13}	D_{14}	P_{10-14}	D_{10}	D_{11}
3	D_{18}	D_{19}	P_{15-19}	D_{15}	D_{16}	D_{17}
4	D_{24}	P_{20-24}	D_{20}	D_{21}	D_{22}	D_{23}
5	P_{25-29}	D_{25}	D_{26}	D_{27}	D_{28}	D_{29}

(f) RAID level 5: Rotated Block Parity

Figure 1.5 RAID levels 0 through 5. Parity blocks have been highlighted while the data blocks make up the rest. "D" represents a block of user data, "d" a bit or byte of user data, "h_{x-y}" a Hamming code computed over user data bits/bytes x through y, "p_{x-y}" a parity (exclusive-or) bit/byte computed over data blocks x through y, and "P_{x-y}" a parity block over user data blocks x through y. (Continued)

The RAID level 5 design shown in Figure 1.5 uses a *left-symmetric* organization,[9] which is formed by first placing the parity units along the diagonal and then placing consecutive data stripe units on consecutive disks at the lowest available offset on each disk. The parity is computed over a group of disks that is called a *parity group*. In the examples illustrated in Figure 1.5, there is only one parity group for each RAID level. It is possible that more than one parity group exist in a RAID level. This technique is called *declustering* and the RAID scheme is called *clustered RAID.*[10,11] It is also known as RAID level 6 and is an extension of RAID level 5. It uses Reed-Solomon coding with P and Q as group parities to protect against two disk failures.

The reliability of single disk tolerant disk arrays can be measured in the form of *mean time to data loss (MTTDL)*. A simple expression for the MTTDL for a redundant disk array that can tolerate one disk failure is given by[1]

$$MTTDL = MTTF_{RAID} = \frac{(MTTF_{disk})^2}{N(G-1)MTTR_{disk}} \tag{1.1}$$

In the above equation, N is the total number of disks in the array, G is the number of disks in a RAID group (i.e. a set of disks over which a parity is computed), $MTTF_{disk}$ is the mean time to failure (mean failure time) of a component disk, typically one million hours, and $MTTR_{disk}$ is the mean time to repair (mean repair time) of a component disk, typically a few hours.

1.3 Trends and Issues in Data Storage Technology

This section provides an overview of the issues and trends in data storage technology that led to designing a Heterogeneous Disk Array architecture.

In the past ten years, dramatic improvements have been made in magnetic disk technology. Disk density has been improving by more than 100 % per year, quadrupling in three years. Prior to 1990, density increased by 30 % per year, doubling in three years. Looking into the future, it appears that disk technology will continue the fast density growth rate for some time to come. Access time has also been improving by one-third in ten years.

Such drastic improvements in disk capacity have numerous benefits, but that is beyond the scope of this thesis. However, some implications must be mentioned, as they motivate the introduction of the proposed Heterogeneous Disk Array (HAD) architecture.

Consider a computer system for which storage needs are fulfilled by a couple of disks. After operating for several years, one of the disks fails. The computer system administrator is faced with the option of replacing the failed disk with a newer model, given that the old model might not be available or cost effective anymore. Since the new disk model most likely has a higher capacity, a lower access time and a higher transfer rate, the extra resources will be wasted. The explanation behind such wastefulness is not being able to use up more than the capacity or bandwidth of the old disk model. The computer system administrator will be faced with a similar problem when a disk array is filled to capacity after years of use, due to constant increases of data being stored and/or installation of new computer applications onto the system. There is the possibility that an identical disk model might not be available at the time, or worse, not even manufactured.

The choices facing the system administrator are simple: either add new disks to the disk array if the architecture allows it, or replace the entire array with a new one. Selection of the first option results in a large waste of resources (capacity and bandwidth) of the newly added disks. However, the second approach might cause the whole operation to become too costly.

An optimal solution to this problem will be one that allows new disks of different models to be added to a disk array when there is a need for extra storage or bandwidth while simultaneously utilizing the added resources to the full extent. This way the benefits of adding a faster and better storage technology combined with a better performance/cost ratio are fully utilized. The possibility of this optimal solution demands that the storage system be comprised of heterogeneous disks.

Another aspect worth mentioning is that disk arrays are usually shared by multiple computer applications, where each application tries to access a multitude of datasets. Each dataset tends to have a number of different requirements, such as capacity, throughput, and reliability. The RAID architecture can meet a lot of requirements, but there is not one RAID level that can meet all of them. Each RAID level performs well for a certain range of workloads with specific characteristics. RAID level 0 is used when there is a need to store a high volume of temporary data, since this level has the lowest storage overhead and does not incur the small write penalty associated with writing to the parity disk. RAID level 1 is a highly reliable system that can tolerate up to N/2 disk failures without losing data; hence it should be used when the cost per megabyte of storage is not as important as the reliability of the storage system. It is also desirable to use RAID level 6 over RAID level 5 for critical data since RAID level 6 provides

increased protection. RAID level 4 is rarely used. RAID level 5 is best used by on-line transaction processing (OLTP), in which a large number of independent processes concurrently request relatively small units of data from the array. RAID level 3 is best suited for applications such as scientific calculations, where a single process requests a large amount of sequential data from the disk array.

An optimal solution to solving the RAID level dilemma will be to design a storage system on which various RAID levels are allowed to coexist at the same time, thus being able to satisfy the requirements of various applications optimally and concomitantly. In other words, for a viable solution, the storage system will have to be heterogeneous regarding the RAID levels.

One of the very important factors that originally prompted the discussion about designing Heterogeneous Disk Array architectures is the cost of maintaining storage systems. Multiple studies have been done over the years that indicate that the cost of maintaining large storage systems over time tends to be relatively high and driven mainly by the high cost of storage management.[12,13]

To solve the high cost problem, the Heterogeneous Disk Array architecture that will be described later in this work will be able to self-manage itself. It will be able to balance the data load automatically by constantly monitoring the system's performance.

In summary, the Heterogeneous Disk Array architecture will allow for different disks of various models, as well as multiple RAID levels, to coexist simultaneously in a single disk array. It will all also utilize the resources (disk capacity and disk bandwidth) of its disks to the maximum possible extent. Lastly, it will automatically balance the data loads by constantly monitoring the performance of the storage system.

CHAPTER 2

HETEROGENEOUS DISK ARRAY ARCHITECTURE: STRUCTURE AND DESIGN

After reviewing the trends in data storage technology and the motivation related to the issue of heterogeneity in disk arrays, it is only natural to introduce the Heterogeneous Disk Array (HDA) architecture, an ongoing concept proposed and researched by the Integrated Systems Lab at NJIT.[14] In the following sections the Heterogeneous Disk Array features, design and function of the essential components of the architecture, metadata structures, and flow charts depicting the processing of requests are described.

2.1 HDA Architecture: Features

The proposed HDA architecture is so unique because the disk array may be comprised of various disk models of different bandwidth and capacity; multiple RAID levels are allowed to coexist in a disk array; disk resources, such as capacity and bandwidth, are utilized to the maximum possible extent; load balancing occurs automatically through constant monitoring of the system performance and improvement in the data allocation decisions. The first two features are concomitantly satisfied, thus ensuring that the heterogeneity aspect is met.

2.2 HDA Architecture: Design and Functionality

HDA architecture consists of an array controller and multiple disks of various models. The array controller is the primary component and is responsible for maintaining the functionality of the system. The components that make up the array controller are the *scheme selector*, the *splitter*, the *distributor*, the *system directory*, the *performance monitor*, and *system tuner*. The design and architecture of an HDA system is depicted in Figure 2.1. Before discussing the array controller components, it is necessary to present the request types.

The HDA system is capable of handling three types of user data requests: *allocation requests*, *update requests*, and *read requests*. An allocation request creates a new object by allocating available space from the disk array to a new file. An allocation request is sometimes followed by an update of the file. An update request updates the object that has been previously allocated. A read request performs reading of data from the disk.

The parameters of the reads and updates are the logical address and the size of the request. The logical address is translated into a physical address by the system directory. Once the physical address is obtained, the controller performs the required operation. If the operation is an update request, the system directory also updates the parity blocks if it is necessary.

Allocation requests, on the other hand, are more complex and require more parameters such as desired availability rating, expected access rate, expected read/write ratio, and size of the request. The desired availability parameter is used in deciding the RAID level, and the expected access rate and read/write ratio are used in the allocation

process. The availability parameter is specified either quantitatively by using Mean Time To Data Loss, MTTDL, or qualitatively (e.g. very high availability). A simpler approach is to tag the allocation request with the desired RAID level.

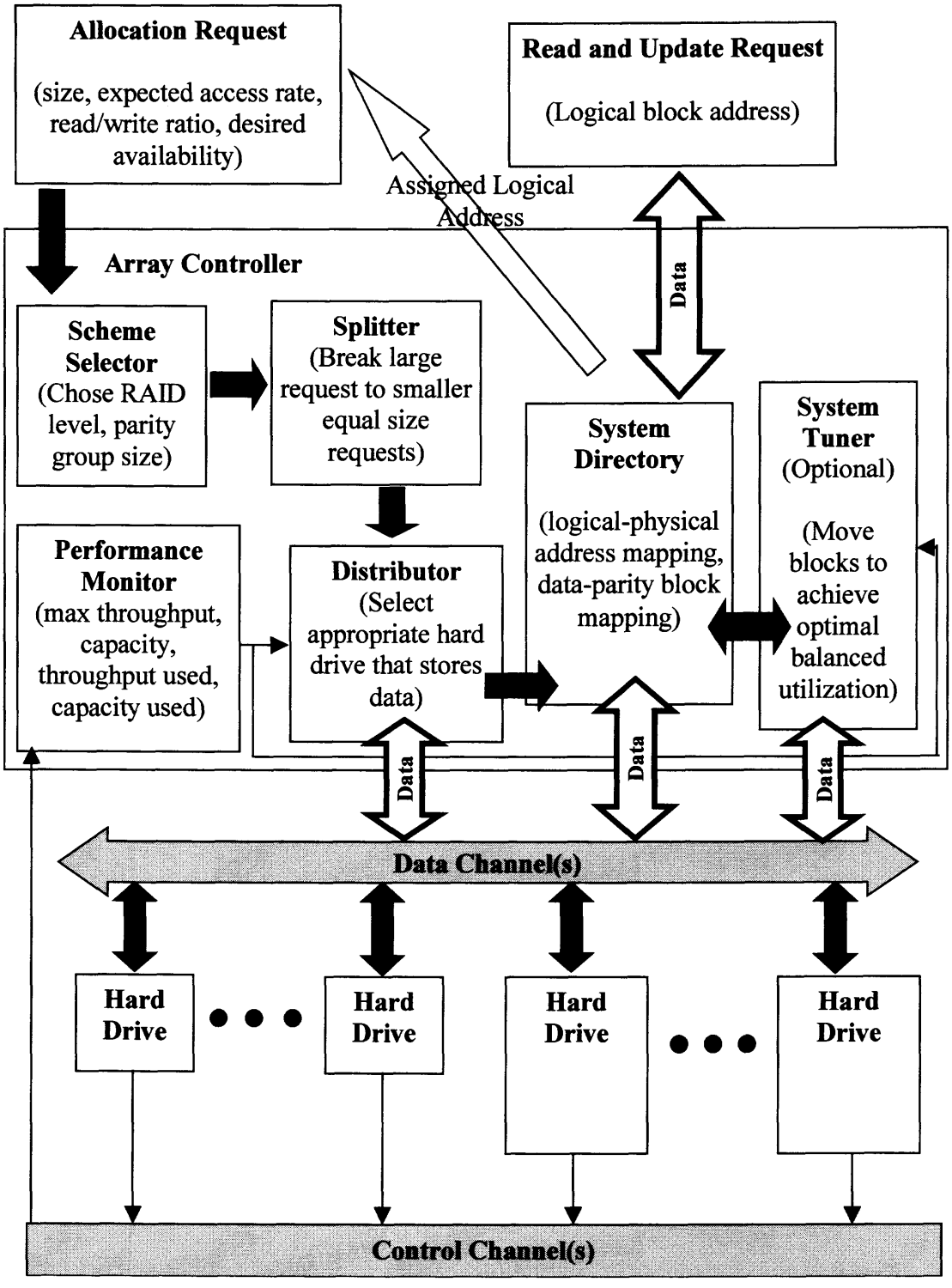

Figure 2.1 Architecture of the Heterogeneous Disk Array (HDA) system.[19]

Knowledge of the request types, encompassing allocation requests, update requests, and read requests, permits description of the array controller components: scheme selector, splitter, distributor, system directory, performance monitor, and system tuner.

- **Scheme Selector**

  The scheme selector is the first module of the array controller that deals with the incoming allocation requests. If the availability parameter of the request is specified in terms of MTTDL, the scheme selector selects a suitable RAID level through the help of a reliability model. If the request is tagged with a RAID level the Scheme Selector just passes the request onto the next module. The reliability model is described in the Appendix.

- **The Splitter**

  The role of this module is to split big allocation requests into smaller size pieces called sub-allocation requests. Each sub-allocation request can be handled in a batch or separately and additional constraints may apply to it. For example, if the data is stored using the RAID level 5 scheme, the sub-allocation requests should be sent to different disks.

- **The Distributor**

  The distributor, or allocator, is the key component of the array controller. Its function is to take a batch of equal-size blocks from the splitter and assign them to disks in a way such that the bandwidth and capacity utilizations of all disks are roughly equal, and that any required constraints are met. The allocator also handles all of the allocation requests, manages the free available space, and decides which device the space is allocated from for the allocation request. If the current virtual array does not have enough free space for a new allocation, a new virtual array of the desired RAID scheme is created. In this case, the allocator must determine the subset of devices from which the new virtual array is created.

- **System Directory**

  After deciding the placement of each sub-allocation request, the distributor outputs its decision to the system directory. This model is composed of a set of data structures and procedures that store and retrieve the logical-to-physical address mapping information in terms of space and time, store and retrieve the data parity relations between blocks, keep track of the hotness of blocks for performance tuning purposes, and manage the free space. It provides a layer between the logical addresses that is visible to the Operating System and the physical addresses that is visible to devices. This layer is implemented using fully associative mapping which makes the data migration transparent to the user. To make this mapping possible and overcome size

concerns, relocation blocks, RBs, are used as the smallest unit of data migration. Each RB has an entry in the system directory and a corresponding logical address termed RB number.

Another important task of the system directory is keeping track of the actual access rate to data objects. The actual access rate is the input to the system performance tuner component. Due to the large volume of requests, only the aggregated access rate to a RB is recorded.

The information stored in the system directory is crucial to the HDA system. It is comprised of all the information about the format and data organization of the array and some statistical data. The aggregation of this information is called *metadata*, and the corresponding data structure is called *metadata structure.* The entities of the metadata are described in section 2.3.

- **System Performance Tuner**

The potential exists for the system to run at a low utilization. Allocations are based on predicted access rates that require the extension of operation system functionality to record the mean access rate and pattern (i.e. read/write ratio) for data generated by a certain application. In a multiprogramming environment, accurate predictions are very difficult even with the aid of the operating system. Moreover, the situation is compounded by several constraints that must be satisfied when making allocations that cause load balancing to be more difficult, such as data and parity blocks that are in the same stripe must be placed on different disks. When the system is running at a low utilization, it would be beneficial to have a background process that tunes the system.

The system performance tuner, the system tuner component in Figure 2.1, obtains disk utilization statistics, including throughput and space utilization, and the access frequency for data as input and balances the utilization of all disk drives by swapping data blocks between disk drives. If there is an improvement involving "n" disks, there exists at least one improvement involving just two disks.[15] Thus, load balancing only requires the swap operation, allowing the system to reduce the neighborhood-searching amount.

Load balancing is performed according to a greedy algorithm called disk cooling.[16,17] Disk cooling tracks the heat associated with data blocks, computes the temperature of each disk, and relocates the hottest block from the hot disks so that the number of blocks to be moved is minimized. The cooling process is triggered only when the temperature of hottest drive is higher than $1 + \delta$ the average temperature, for which δ is a system parameter. The dynamic tracking of the block temperature is implemented based on a moving average of the inter-arrival time of requests on the same block. However, shortcomings of the disk cooling algorithm include an assumption of homogeneous disk drives, and the algorithm does not consider the

constraints introduced by redundancy schemes, for example, data and parity cannot reside on the same disk.

The system performance tuner runs in the background and only when the system is idle or has a light workload since data migrations are usually expensive operations.

- **Performance Monitor**

  The performance monitor module is constantly keeping track of the device utilization while the system directory records the access rate for data blocks in order to collect current data access rate and device utilization information. This information is passed along to the system performance tuner so that load balancing on disks may occur.

2.3 Data Structures and Procedures of the System Directory

In this section, the entities of the metadata are described and the frequent operations are analyzed, the address translation diagram optimized according to the frequent operations is depicted, a list of the data structures used by the HDA system is presented, and the read/write operation flowcharts, based on these data structures, are discussed.

2.3.1 Addressable Entities in the HDA System

The addressable entities in the HDA system include device, relocation block, virtual disk, virtual array, and data/parity block. Their relationships are depicted in Figure 2.2 and each is described below.

- **Device**

  A device is an array-controller-visible physical device but can also be a disk array. Each device is identified by a unique *Logical Unit Number* (LUN) and has its own capacity and maximum throughput (bandwidth). All blocks in the device are addressable by using a *device_offset.*

- **Relocation Block, RB**

  A relocation block, RB, is an entity comprised of a set of contiguous sectors, and its size is a predetermined number.

- **Virtual Disk, VD**

  A virtual disk, VD, is an entity comprised of a set of a predetermined number of contiguous RBs on a device.

- **Virtual Array, VA**

  A virtual array, VA, is an entity comprised of one or more VDs from different devices. A virtual array is formatted to use a certain RAID level.

- **Data/Parity Block**

  Data blocks are accessible by user applications, while parity blocks store redundant information, and are usually transparent to the users. Both data and parity blocks are addressable.

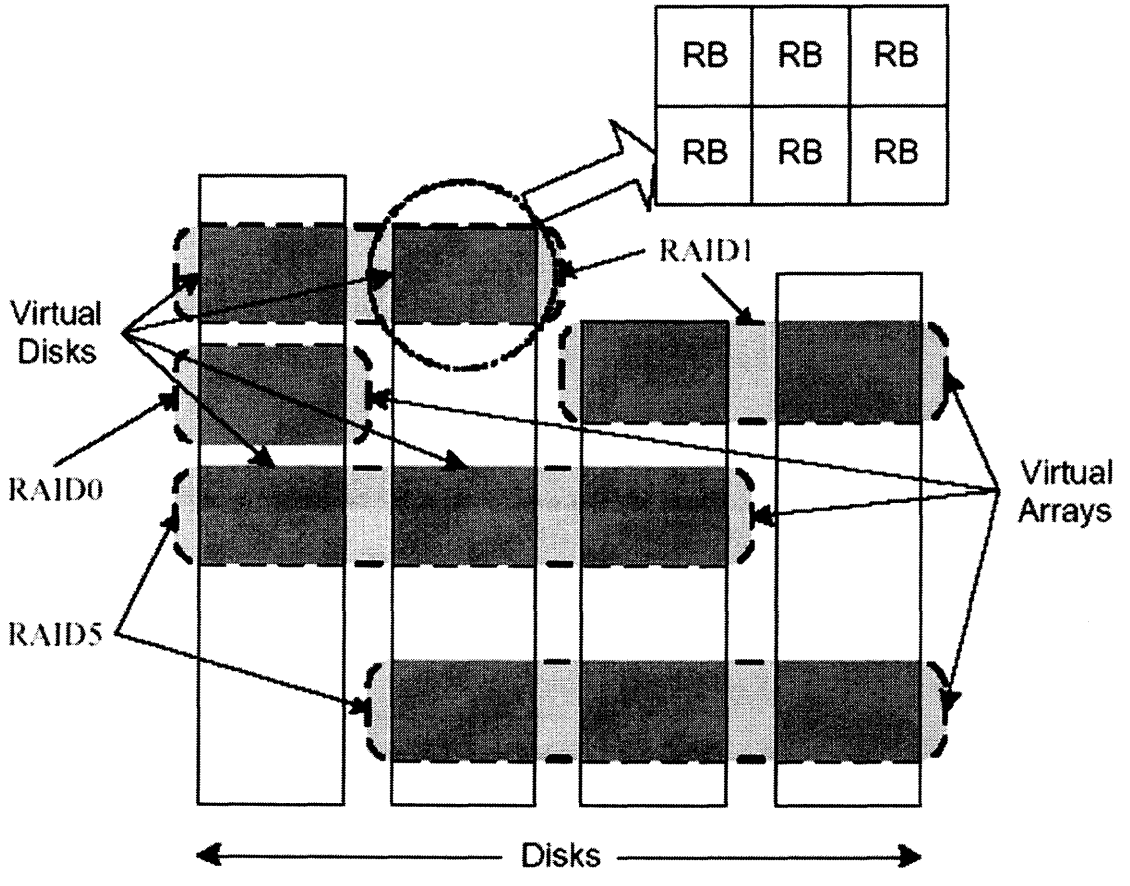

Figure 2.2 Entities in the HDA system and their relationship.[19]

The device, RB, VD, and VA entities each have an address associated with them. The addresses are consecutive integers starting from zero, and for different entities they belong in different dimensional address spaces (e.g. both RB and device number can be 0). The addresses belonging to an address spaces are not necessarily consecutive, which means that holes can exist in the address space. Consecutive addresses in the device, RB, VD and VA address space simplify and greatly speedup the table lookup procedure.

The fifth entity, data/parity blocks, uses a combination of two addresses to identify each block: a physical address that is understood only by devices and an HDA address that is visible to the users.

The physical address is comprised of a logical unit number and a device_offset pair: (LUN, device_offset). The LUN is the device number of a disk, and every physical address uniquely identifies a block in the storage space. It is possible for the identified block to store user data or redundancy information, in which case it will not be addressable in user storage space.

The HDA address is comprised of a relocation block number and a relocation block offset (RB#, RB_offset), where the RB_offset is the offset considered from the beginning of the RB. The HDA address is visible to the file system and is the counterpart of the physical address in user storage space.

2.3.2 Address Translations in the HDA System

The handling operations of the read, update, and allocation requests require the following procedures to be performed by the array controller: translation from an HDA address to a physical address, scheme lookup, buddy lookup, physical address to VA information

translation, creation of a new VA from the available free storage space, count the number of VAs of a given RAID level, and relocate the RBs through the load balancing process. Figure 2.3 illustrates the address-mapping diagram. The addresses of the HDA system entities can be translated following the arrows in the diagram. The ratios on the links depict the mapping relationships: one-to-one, one-to-many or many-to-many.

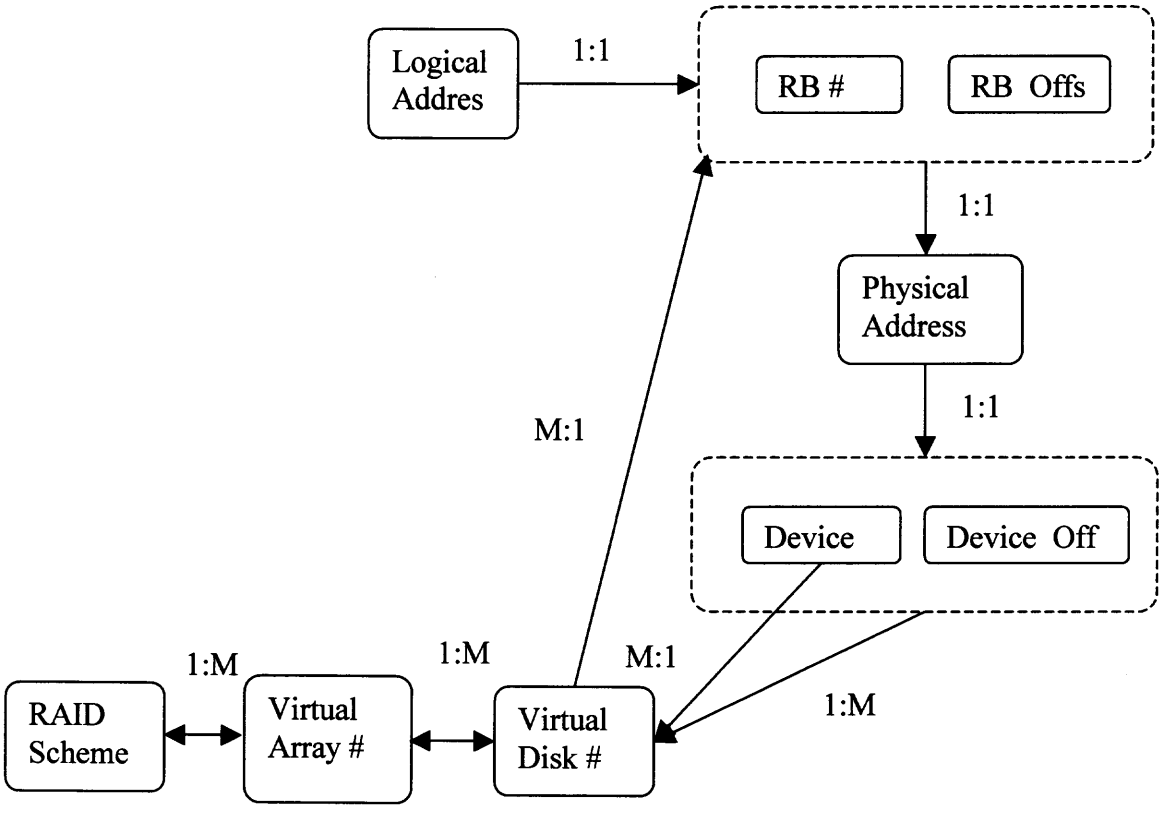

Figure 2.3 Address mapping diagram.

2.3.3 The Data Structures of the Meta Information

The HDA system has to maintain a set of tables in order for the array controller to be able to perform the address translation procedures. A simplified depiction of the tables maintained by the HDA system is shown in Figure 2.4.

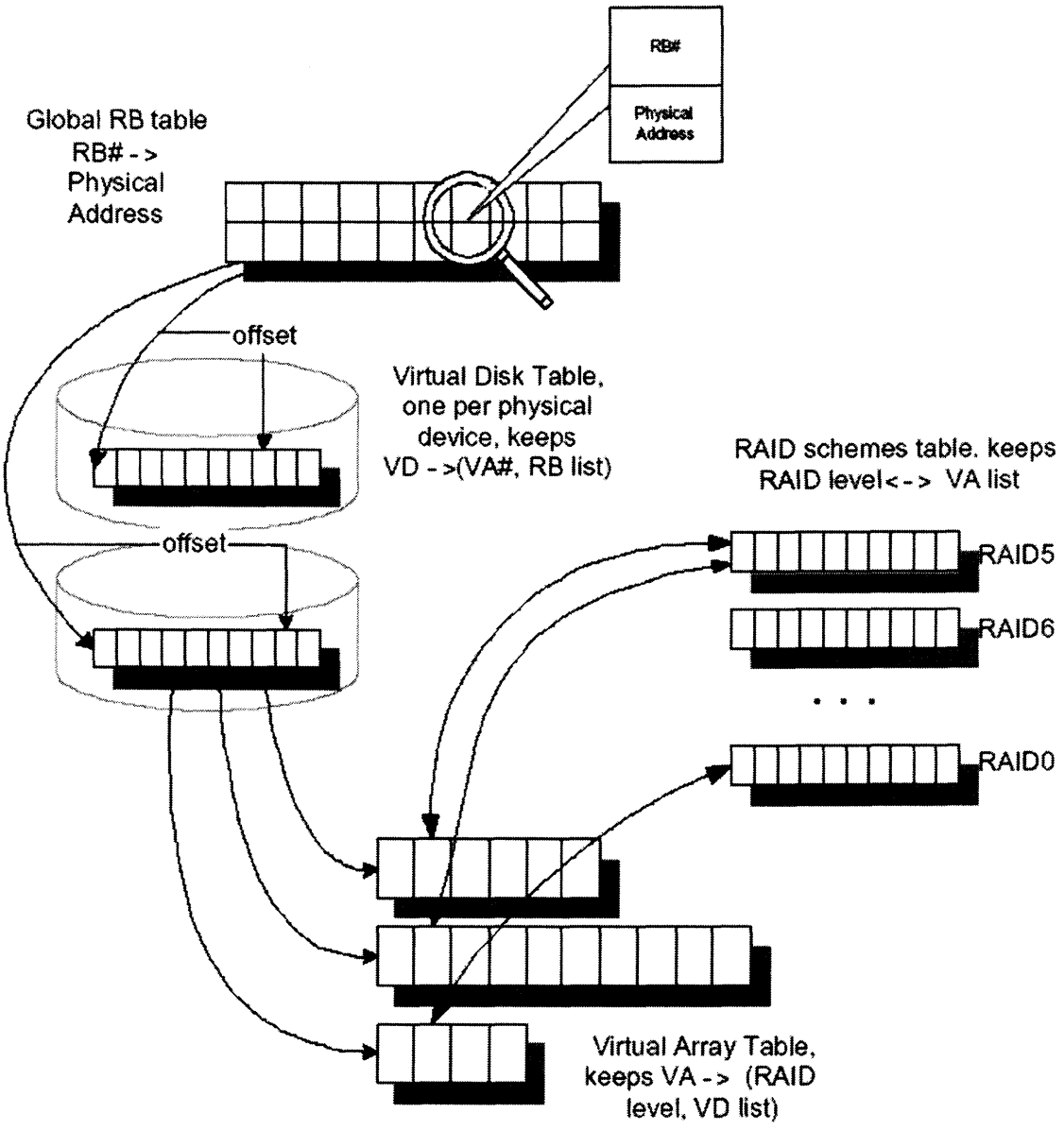

Figure 2.4 The tables maintained by the HDA system.[19]

There are four tables illustrated in Figure 2.4 above. The data structures and relationships found in these tables are described below:

- **Global RB Table**

  This global table stores the mapping of RB numbers to physical addresses and the heat index. The data structure that represents a row in the table is depicted in Figure 2.5. The RB# is used to index the global table, the Device# and Device_offset make up a physical address pair, and the Heat index records the access rate for the blocks of this RB. The RB# can be omitted, if it grows sequentially without interruption.

RB#	Device#	Device_offset	Heat Index

Figure 2.5 The row structure in the global RB table.

- **VD Table**

  The virtual disk, VD, table consists of records that have their structure depicted in Figure 2.6. Each physical device has a VD table. VDs contain a fixed number of RBs (RB_PER_VD), have a fixed size, and can be described as consecutive disk spaces on physical devices. The RB numbers that are in a VD at one time may not be consecutive, as a result of background workload balancing. Each VD can be marked as unformatted space before it is assigned to a virtual array, VA. The virtual disk number, VD#, consists of a Device# and an index of the VD in that device, and it can be directly mapped to a physical address by multiplying the VD size with the index. The virtual array number, VA#, indicates the VA to which the VD belongs. RB_1, RB_2, ..., $RB_{RB\_PER\_VD}$ represent the RBs in this VD. The VD# is implied and may be omitted if the sequence of numbers is uninterrupted.

VD# = (Device#, Index in device)	VA #	RB_1	RB_2	...	$RB_{RB\_PER\_VD}$

Figure 2.6 VD table record.

- **VA Table**

  The virtual array (VA) table is also a global table. A field in the VA table is depicted in Figure 2.7. The VA table stores information regarding the organization of the virtual arrays, such as the RAID scheme, the components VDs, the parity location etc. VD_1 through VD_k represent the virtual disks that constitute the respective virtual array, with the parity VDs listed first.

VA#	RAID Scheme	Num VDs (=k)	VD_1	VD_2	...	VD_k

Figure 2.7 The structure of a VA table field.

- **RAID Scheme Table**

  The RAID scheme table is a global table that stores all the VAs for a certain RAID level in the HDA. Figure 2.8 illustrates the structure of a RAID scheme table record. The records are of variable length.

RAID Scheme	Num VAs	VA_1	VA_2	VA_3	...

Figure 2.8 Structure of a record in the RAID scheme table.

2.3.4 The Read and Write Operations

The read and write operations require certain steps to be taken while performing address translation. Figures 2.8 and 2.9 depict the flowcharts that illustrate the address translation process in both cases. Operations in normal mode only are discussed here, and the write operation assumes a RAID level 5 array organization. Operations in degraded mode and using different RAID level schemes are similar.

The read operation, as depicted in Figure 2.3, consists of two parameters, the target HDA address and the request size, and translates the HDA address to the physical address by simply identifying an integer in a table and performing one addition step. The HDA address is comprised of the *RB number*, identified by the pair (*device number, device_offset*) that gives the physical address in the RB table, and the *offset within the RB*. The device_offset is the starting point of the RB. Summation of the device_offset and the offset within the RB provides the physical address of the target block. Then, the request is passed down to the device without any change. Note that identifying the RB number in the RB table is equivalent to obtaining an element from a large array that

requires only one multiplication to get the element address and one memory access because the RB number is a sequential integer starting from zero.

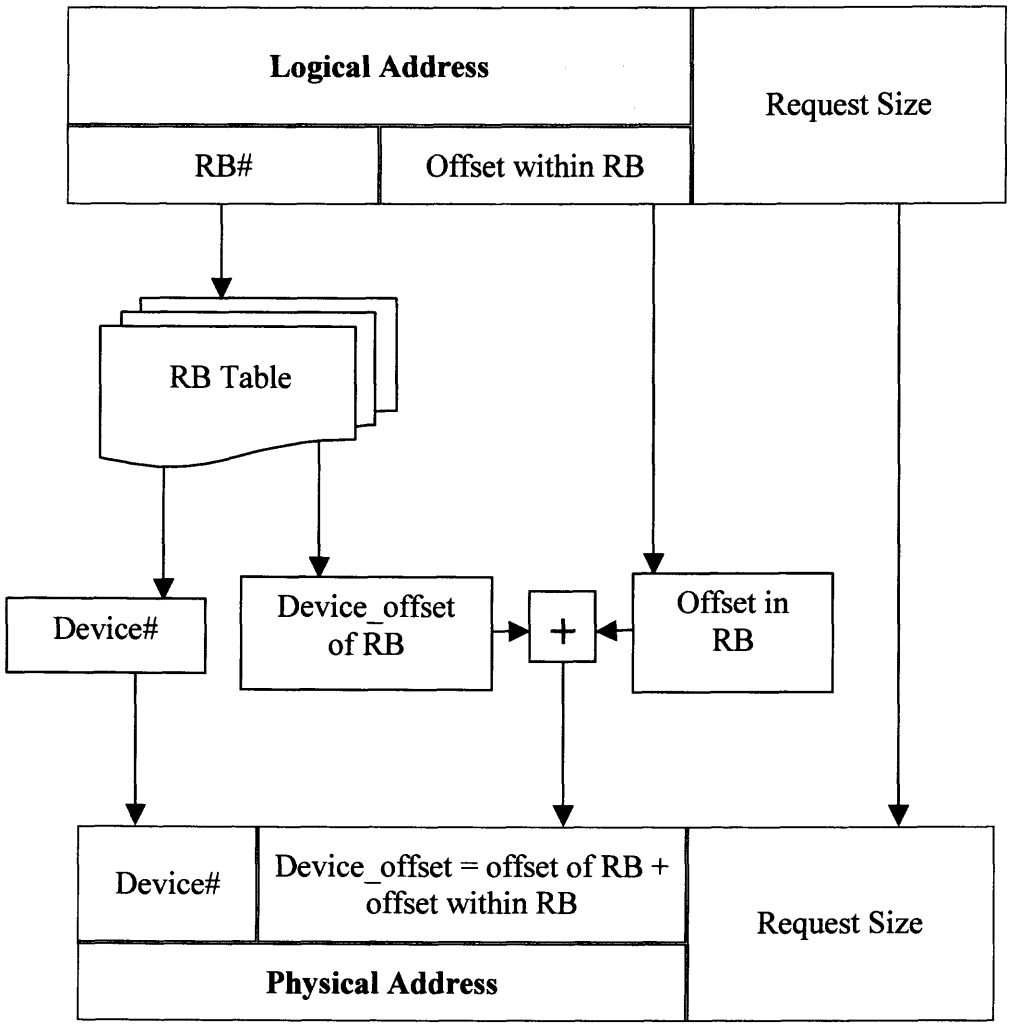

Figure 2.9 The read operation address translation flowchart.[19]

CHAPTER 3

DATA ALLOCATION METHODS IN HETEROGENEOUS DISK ARRAYS

One of the main challenges in heterogeneous disk arrays is finding a way to utilize both capacity and bandwidth to the maximum possible extent. This chapter formalizes this problem and describes a solution to it in the form of a data allocation algorithm. To verify its effectiveness, an HDA simulation environment was designed and implemented at the Integrated Systems Lab at NJIT[19], and several data allocation algorithms were tested against the proposed solution.

3.1 Problem Formulation

The hard disks that make up the storage space of a heterogeneous disk array can be modeled as two-dimensional vectors, as illustrated in Figure 3.1. The two dimensions are maximum throughput, measured in accesses per second, on the X-axis, and storage capacity on the Y-axis.

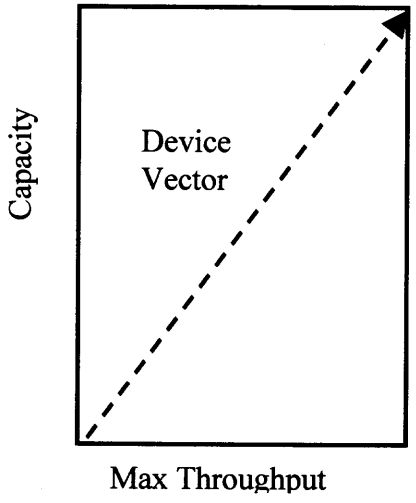

Figure 3.1 Disk modeled as device vector.

Allocation requests that come to the disks can also be modeled as two-dimensional request vectors. The two dimensions are access rate on the X-axis and allocation size on the Y-axis. Figure 3.2 illustrates this concept.

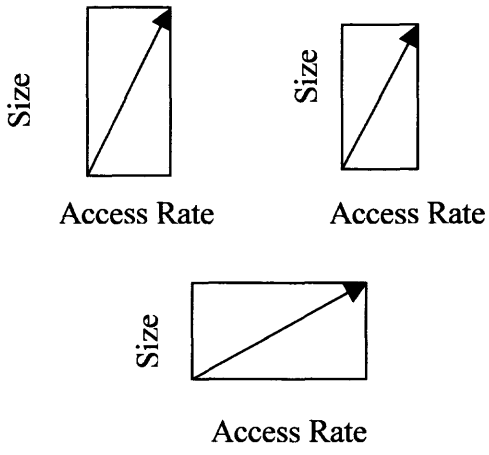

Figure 3.2 Allocation requests modeled as a two-dimensional vectors.

The disk allocation process can be modeled by adding the request vectors, and ensuring that their sum, on either coordinate, is less than the dimensions of the disk vector. Figure 3.3 illustrates this concept.

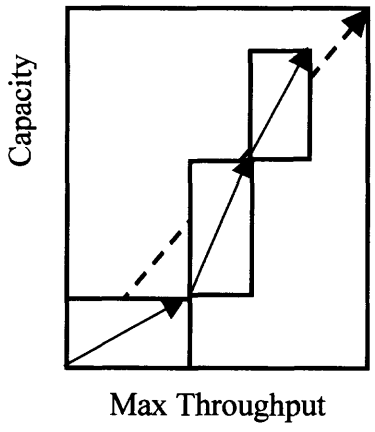

Figure 3.3 Disk allocation process modeled as vector sum.

Based on the above concepts, the problem of balancing the utilization of disks in terms of both throughput and capacity is formulized as follows:

- **Problem Definition**

  Considering a set D of n disks, the i^{th} disk is represented by a vector $d_i = (X_i, C_i)$, where X_i denotes the maximum throughput and C_i denotes the capacity of the i^{th} disk. Given a set J of allocation requests, each allocation request is represented by a two-dimensional vector $p_j = (x_j, c_j)$, where x_j is its anticipated access rate and c_j is the size of data.

- **Problem Solution**

  A partition of the set J into n subsets $J_1, \ldots, J_n$ such that the sum of both dimensions of subsets does not exceed the corresponding limit set by the dimensions X_i and C_i, as in Equations (3.1) and (3.2):

$$\sum_{j \in J_i} x_j \leq X_i \quad \text{, for all } 1 \leq i \leq n \tag{3.1}$$

$$\sum_{j \in J_i} c_j \leq C_i \quad \text{, for all } 1 \leq i \leq n \tag{3.2}$$

Offline and *online* algorithms are used to implement data allocation methods. In an offline algorithm, complete knowledge of all the items in the system is required before the starting of the algorithm. An online algorithm, on the other hand, assigns an item using only the item's own information and the system's statistical data.

Both offline and online algorithms are used in an HDA system. The *system tuner* uses the offline algorithm, while the *distributor* uses the online algorithm in the initial data allocation phase. The online algorithm used by the distributor processes allocation requests in the order of their arrival (FCFS policy), until no more requests need to be allocated. Given that Equation (3.3) denotes the utilization (of throughput) of device i,

$$U_i^x = \frac{\sum_{j \in J_i} x_j}{X_i} \tag{3.3}$$

and Equation (3.4) denotes the utilization capacity of device i,

$$U_i^c = \frac{\sum_{j \in J_i} c_j}{C_i} \tag{3.4}$$

the two possible objective functions to minimize are F_1 and F_2, as in Equations (3.5) and (3.6), respectively:

$$F_1 = \max_{1 \le i \le n} \left\{ U_i^x, \alpha U_i^c \right\} \tag{3.5}$$

and

$$F_2 = Var\left\{U_i^x\right\} + \alpha Var\left\{U_i^c\right\} \tag{3.6}$$

In Equation (3.6), ***Var{x_i}*** is the variance over a set of numbers $\left\{x_i \middle| 1 \le i \le n\right\}$ and is defined in Equation (3.7):

$$Var\{x_i\} = \frac{\sum_{1 \le i \le n} x_i^2}{n} - \left(\frac{\sum_{1 \le i \le n} x_i}{n} \right)^2 \tag{3.7}$$

To put more emphasis on the throughput rather than capacity, in Equation (3.6), α is chosen between 0 and 1. Balanced throughputs are more important than balanced disk capacities, as more often the system bottleneck tends to be the throughput. Another reason is that mean disk response time is proportional to $(1-\rho)^{-1}$ since the disk throughput is directly proportional to the disk utilization ρ, which is a product of the arrival rate of requests and the mean service time per request.[18] It is assumed that the mean access times of the various disks of the HDA system do not vary significantly, in spite of their different sizes and models. A computational expensive optimization algorithm to improve response time is unnecessary and impractical in the allocation phase, since access rates and read/write ratios are only approximately known. In a typical storage system, each allocation block takes a very small fraction of the throughput and capacity of a disk. A bin packing with small items type of problems may be used to illustrate the allocation of blocks to disks. A good asymptotic ratio is possible when using a greedy algorithm for finding an approximate solution to the two-dimensional vector-scheduling problem. The best-fit heuristic algorithm is adapted to the two-dimensional vector-scheduling problem by assigning an incoming request to the $\mathbf{k}^{th}$ device to minimize the target function. The algorithm is illustrated in Figure 3.4.

Algorithm: **BEST_FIT_SCHEDULE** $\left(U_i^x, U_i^c, p\right)$

Input: The utilization of throughput $\left(U_i^x\right)$ and capacity $\left(U_i^c\right)$ for the device i, $1 \le i \le n$; The expected throughput (x) and capacity (c) consumption of the new allocation request, represented as a vector p = (x,c).

Output: The k^{th} device, $1 \le k \le n$, to which the new allocation request should be assigned.

Steps:

1. Consider the target functions $F(i)$, $1 \le i \le n$, where the new allocation request is assigned to disk i.
2. Compute $F(i)$ for all $1 \le i \le n$, using a throughput. If a device does not have enough space or bandwidth for the new item, it is excluded from consideration.
3. Select k such that $F(k) \le F(i)$; $1 \le i \le n$. If there are multiple choices, select one randomly.
4. Return k.

Figure 3.4 Best-Fit scheduling algorithm.[19]

3.2 Other Data Allocation Methods

To verify the effectiveness of the best-fit allocation algorithm, other data allocation methods were considered in this study[19]:

- **Round-robin:**

  It places the allocation requests in a round robin manner on the disks.

- **Random:**

  It places the allocation requests randomly on the disks.

- **Proportional to throughput:**

  It places the allocation requests on the disks with a probability proportional to their throughput (bandwidth). In mathematical terms, the probability of placing a request on disk *i* is:

- **Proportional to capacity:**

  It places the allocation requests on the disks with a probability proportional to their capacity. In mathematical terms, the probability of placing a request on disk *i* is:

3.3 Constraints on the Allocation Process

There are several constrains that must be considered during the allocation process, such as:

- The data and corresponding parity stripe units in a RAIDx parity group must not be placed on the same device. This constraint is essential to the correctness of the scheme.

- Blocks from the same allocation request should be placed on a single device up to the size of a stripe unit. This constraint is beneficial from the viewpoint of sequential access.

- The stripe units of a stripe should be placed on disks with similar characteristics.

The first constraint is required for data recovery after a device failure, while the other two constraints aim at better performance.

3.4 Allocation Algorithms Used in the Simulation

The simulation works by assigning consecutive numbers to allocation requests. Their attributes are recorded in a table, which is used by the simulator to determine the characteristics of requests. For example, after ***i*** requests are allocated, the total arrival rate of read/write requests is $\Lambda_I = \sum_{i=1}^{I} \lambda_i$. Once an arrival occurs, the probability that allocation ***i*** will be accessed is proportional to λ_i / Λ_I . The logical or the HDA address of the allocated requests is also recorded to the table. As time elapses, more allocation requests are processed, more space is allocated, and the access rates to disks increase. Allocation requests are infrequent compared to read/write requests and require the same disk access time as the latter requests, so the processing required by them is negligible.

The HDA system allocates at two levels, at the VA level and at the file level. New VAs are allocated on demand, when there is not enough space left on the current VA for a new allocation. The VDs for the new VA are selected from the disks with the most available free space, percentage-wise. The virtual bandwidth associated with each VD is based on the remaining access bandwidth divided by the number of free VDs on the device. Thus VDs in a particular VA may have different target bandwidths. The expected bandwidth per VD is simply the disk bandwidth divided by the number of VDs that can fit on that disk. As requests are allocated to VDs of a RAID1 or RAID5 array, the residual bandwidth that remains to be allocated may be lower or higher than expected.

The allocation of bandwidth is accelerated or decelerated based on the residual bandwidth on the disk.

3.5 Configurations for an HDA Simulation

Two configurations are considered in this study. First, a heterogeneous disk array with 6 disks is considered. Secondly, the storage capacity is doubled. The disk capacities range from 2GB to 9GB, and the disks are of various models. The specifications for the disks considered in this simulation study are given in Table 3.1 and Table 3.2.

Table 3.1 Specifications of the Disks Used in the HDA Simulation Study 1

Disk #	Model	Capacity	Bandwidth (access/sec)	Bandwidth – Capacity Ratio	RPM	Sectors/Track
0	IBM18ES	8.6G	88.01	9.779	7200	247-390
1	IBM18ES	8.6G	88.01	9.779	7200	247-390
2	Atlas10k	8.5G	116.89	13.13	10000	229-334
3	Barracuda	2.0G	74.53	35.49	7200	119-186
4	Barracuda	2.0G	74.53	35.49	7200	119-186
5	Cheetah4LP	4.2G	91.63	20.36	10000	131-195

Table 3.2 Specifications of the Disks Used in the HDA Simulation Study 2

Disk #	Model	Capacity	Bandwidth (access/sec)	Bandwidth – Capacity Ratio	RPM	Sectors/Track
0	IBM18ES	8.6G	88.01	9.779	7200	247-390
1	IBM18ES	8.6G	88.01	9.779	7200	247-390
2	Atlas10k	8.5G	116.89	13.13	10000	229-334
3	Barracuda	2.0G	74.53	35.49	7200	119-186
4	Barracuda	2.0G	74.53	35.49	7200	119-186
5	Cheetah4LP	4.2G	91.63	20.36	10000	131-195
6	IBM18ES	8.6G	88.01	9.779	7200	247-390
7	IBM18ES	8.6G	88.01	9.779	7200	247-390
8	Atlas10k	8.5G	116.89	13.13	10000	229-334
9	Barracuda	2.0G	74.53	35.49	7200	119-186
10	Barracuda	2.0G	74.53	35.49	7200	119-186
11	Cheetah4LP	4.2G	91.63	20.36	10000	131-195

The arrival process is Poisson for both allocation and access (i.e. read/update) requests, and the request size follows the exponential distribution. The mean request size is 100 sectors or 50 KB, with a cutoff threshold of 4096 sectors and a minimum of one sector. The access rate is also exponentially distributed, with mean access rate of 8×10^{-7} accesses per second. The maximum access rate is 10 accesses per second. Two RAID levels, RAID1 and RAID5, coexist in the HDA. Each allocation request is tagged as either RAID1 or RAID5, with 30% of them tagged as RAID1. Multiples runs of simulations are executed with various read/write ratios for data blocks, but only the results for the ratio read/write = 3/1 are reported here. The arrival rate for the allocation requests remains constant throughout the simulation. The arrival rate for accessing the

data blocks depends on how many data objects have been allocated on the disk. The simulator keeps track of all the data objects that have been allocated and generates read/write requests according to the actual access rate of each data object after it is allocated. As time elapses, more allocation requests are processed and more space is allocated. Therefore, as the arrival rate for the data objects increases with time, so do the utilizations and response times for disk accesses. The size of a relocation block is 10 megabytes, and each virtual disk has 2 RBs. A RAID1 virtual array consists of two virtual disks. A RAID5 virtual array consists of five virtual disks, one of which stores the parity.

3.6 Studies of Data Allocation Methods

Based on the HDA disk configurations described in Section 3.4, two studies were conducted to test the effectiveness of the proposed best-fit allocation algorithm described in Section 3.1. Six data allocation methods were compared against each other on the two configurations.

In the first study, the simulation uses an HDA system with 6 disks. Their specifications are given in Table 3.1. The simulation stops when either bandwidth or capacity utilization of any disk exceeds 95%. In other words, the stopping criterion of the simulation, in this first study, is chosen as the point when one of the resources of a disk is close to maximum usage. This is done as a first step in observing the behavior of the system. The results of the simulation are given in Table 3.3.

Table 3.3 Simulation Results for an HDA System with Six Disks

Placement Strategy	Alloc Req	Ux(0) Uc(0)	Ux(1) Uc(1)	Ux(2) Uc(2)	Ux(3) Uc(3)	Ux(4) Uc(4)	Ux(5) Uc(5)	σ_x
Best Fit with objective F1	295044	0.9308 0.7926	0.9500 0.7903	0.8198 0.7922	0.8695 0.8014	0.8271 0.8014	0.8436 0.7966	0.0549
Best Fit with objective F2	307441	0.9210 0.8198	0.9501 0.8198	0.8901 0.8219	0.8873 0.8300	0.8851 0.8300	0.9180 0.8241	0.0257
Round-Robin	245483	0.9505 0.6472	0.9267 0.6472	0.6936 0.6461	0.3806 0.6487	0.4200 0.6583	0.5840 0.6501	0.2442
Random	244829	0.9271 0.6472	0.9501 0.6472	0.6849 0.6461	0.4023 0.6487	0.4217 0.6583	0.6203 0.6501	0.2369
Proportional to Max xput	245472	0.9500 0.6472	0.9170 0.6472	0.7063 0.6461	0.4329 0.6487	0.4004 0.6583	0.6082 0.6501	0.2339
Proportional to Capacity	240319	0.9130 0.6359	0.9502 0.6359	0.6934 0.6369	0.3932 0.6392	0.3827 0.6487	0.5864 0.6409	0.2460

It is apparent that the *best-fit with objective* F_2 allocation method can hold more allocation requests than all the other allocation methods. Therefore it is considered more cost-effective than the other strategies. In terms of utilization of both bandwidth and capacity, the *best-fit* allocation algorithm outperforms the other strategies considerably. The utilizations of capacity for the 6 disks used in the simulation are higher, on the average and individually, when the *best-fit with objective F2* strategy is used. The utilizations of bandwidth are also higher, and the standard deviation of bandwidth (σ_x) is smaller than the others. This shows that the resources of the system are used at a higher extent when using this method, while also better balancing the workloads over the system's disks.

A second, more involved study was conducted, with an HDA configuration of 12 disks. The disks specifications are given in Table 3.2. In this case, the simulation runs until the system exhausts all of the resources necessary to meet the allocation requirements. The results are illustrated in Table 3.4 and 3.5.

Again the *best-fit with objective* F_2 allocation algorithm can hold more allocation requests and is therefore more cost-effective than the other strategies. It can be observed that when the program stops, all 12 disks have almost evenly utilized around 74% of their storage capacity. The value of the standard deviation of capacity (σ_c) is very small, suggesting that the utilization of capacity is very evenly distributed amongst the array disks. The utilization of bandwidth is also well distributed within the disks limits, and its values over the disks are higher than those given by the other allocation methods. The fact that the value of the standard deviation of bandwidth (σ_x) is smaller than that obtained using the other allocation methods, also suggests a better load balancing among the disks. It is apparent that the *best-fit with objective* F_2 allocation algorithm outperforms all the other methods not only in terms of allocations serviced, but also in terms of utilization of the disks resources to the maximum extent possible and in terms of workload balancing. Table 3.4 and 3.5 show that, when the data allocation algorithm with objective function F_2 is used, the best use of the array's resources in terms of bandwidth and capacity is achieved.

Table 3.4 Simulation Results for an HDA System with 12 Disks (Bandwidth)

Placement Strategy	Best Fit with objective F1	Best Fit with objective F2	Round-Robin	Random	Proportional to Max xput	Proportional to Capacity
TotalReq	40640200	41940600	31152400	30985300	31775700	31720300
Ux(0)	0.999474	0.999961	0.982288	0.989938	0.982310	0.905335
Ux(1)	0.979776	0.962989	1.000000	0.991781	0.979263	0.907640
Ux(2)	0.974622	0.970862	0.730528	0.728313	0.739521	0.689079
Ux(3)	0.419901	0.413409	0.293921	0.318238	0.317340	0.332361
Ux(4)	0.397409	0.388828	0.306830	0.325501	0.307210	0.346503
Ux(5)	0.655125	0.748192	0.453887	0.466733	0.489540	0.491078
Ux(6)	0.998912	0.953782	0.999810	0.964082	1.000000	0.950157
Ux(7)	0.985658	0.979509	0.977041	0.994290	0.993320	0.990452
Ux(8)	0.969331	0.961500	0.724710	0.736524	0.743054	0.747033
Ux(9)	0.421264	0.424882	0.311146	0.292116	0.331036	0.342688
Ux(10)	0.358549	0.464085	0.317204	0.304111	0.322835	0.349855
Ux(11)	0.626537	0.716240	0.465612	0.465040	0.467171	0.462554
σ_x	0.277345	0.257235	0.303347	0.299985	0.297054	0.266230

Ux(i) = Throughput Utilization for Disk i

Table 3.5 Simulation Results for an HDA System with 12 Disks (Capacity)

Placement Strategy	Best Fit with objective F1	Best Fit with objective F2	Round-Robin	Random	Proportional to Max xput	Proportional to Capacity
TotalReq	40640200	41940600	31152400	30985300	31775700	31720300
Uc(0)	0.724460	0.738087	0.635890	0.633619	0.640432	0.645927
Uc(1)	0.724460	0.735816	0.633619	0.633619	0.640432	0.648198
Uc(2)	0.725985	0.737400	0.634666	0.632383	0.639232	0.636724
Uc(3)	0.725066	0.744146	0.639202	0.639202	0.648743	0.641501
Uc(4)	0.725066	0.744146	0.639202	0.639202	0.639202	0.631501
Uc(5)	0.727912	0.737068	0.636351	0.631773	0.640929	0.645992
Uc(6)	0.724460	0.735816	0.635890	0.631348	0.640432	0.648198
Uc(7)	0.724460	0.735816	0.635890	0.631348	0.640432	0.648198
Uc(8)	0.725985	0.737400	0.634666	0.632383	0.639232	0.636724
Uc(9)	0.725066	0.744146	0.639202	0.639202	0.648743	0.641501
Uc(10)	0.725066	0.744146	0.639202	0.639202	0.648743	0.641501
Uc(11)	0.727912	0.737068	0.636351	0.631773	0.640929	0.645992
σ_c	0.001250	0.003680	0.002023	0.003485	0.003939	0.005426

Uc(i) = Capacity Utilization for Disk i

To closely observe the behavior of the *best-fit with objective* F_2 allocation algorithm, the utilizations of both bandwidth and capacity for all the disks are plotted over the entire simulation time frame in Figure 3.5 and Figure 3.6.

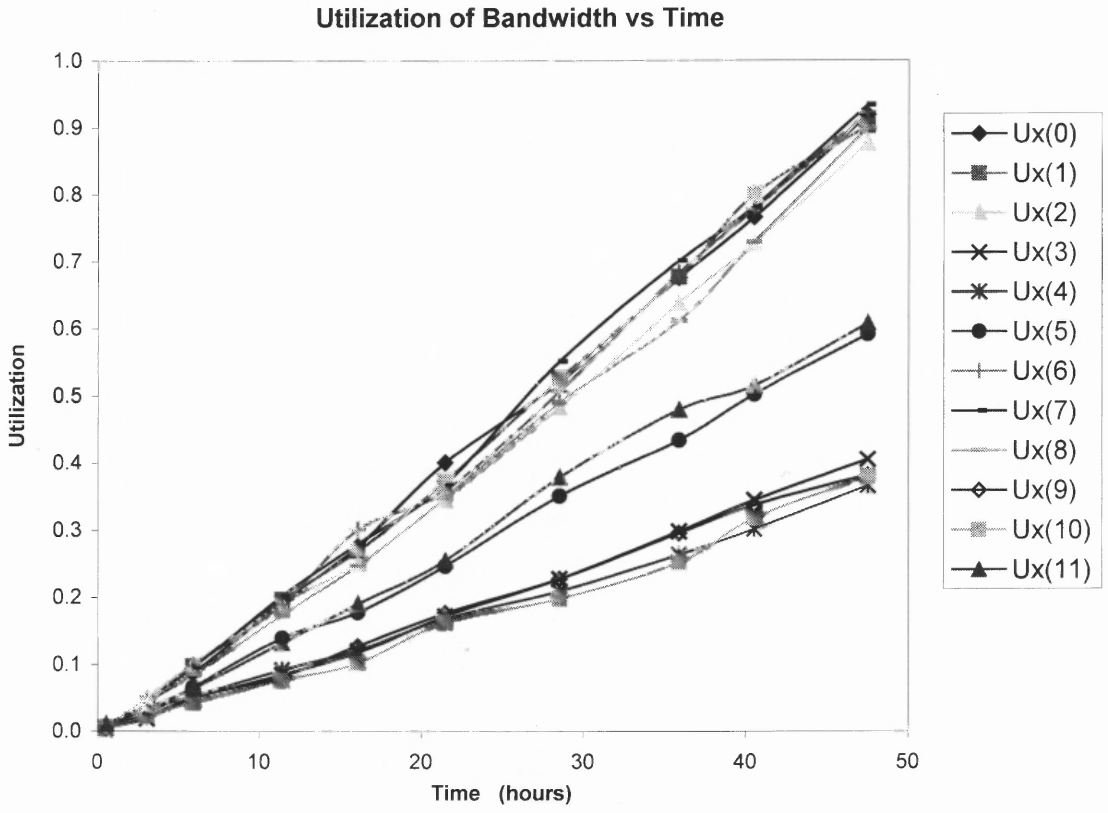

Figure 3.5 Utilization of bandwidth over time.

It is apparent from Figure 3.5 that the bandwidth utilizations of all the disks are relatively close to each other, with some exceptions, more prevalent over time. However, there are no fluctuations or exceptions in Figure 3.6, when the capacity utilizations are graphed over the simulation's time. The values are always very close and indicate a great

workload-balancing act. The system's resources seem to be fully exploited and both capacity and bandwidth utilized in a balanced way.

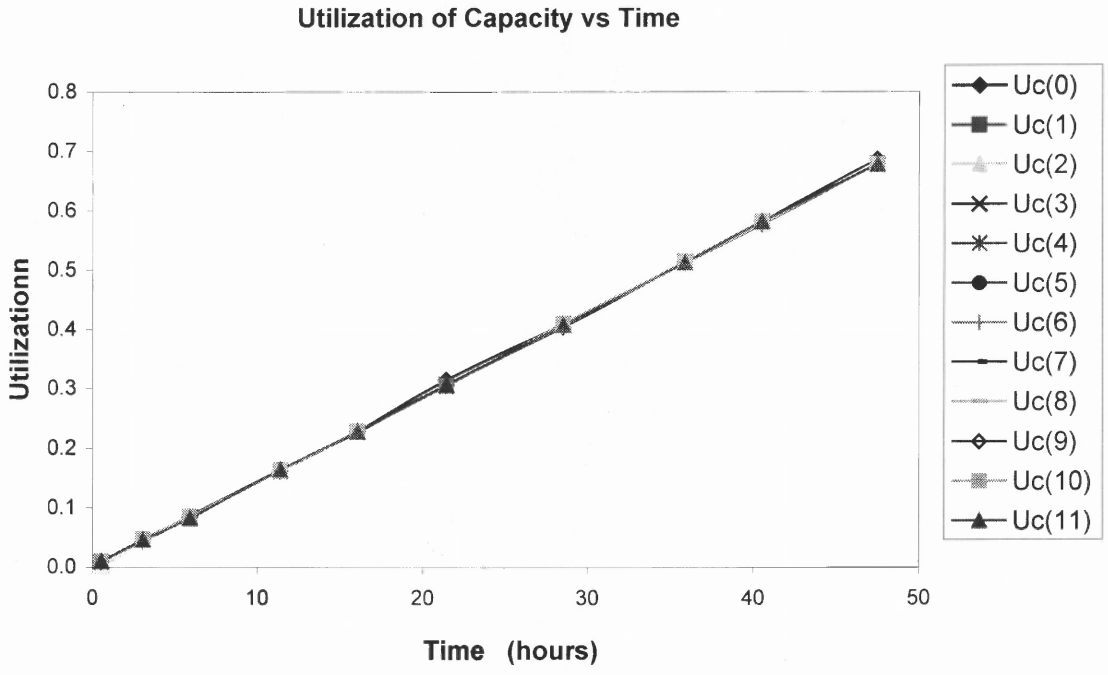

Figure 3.6 Utilization of capacity over time.

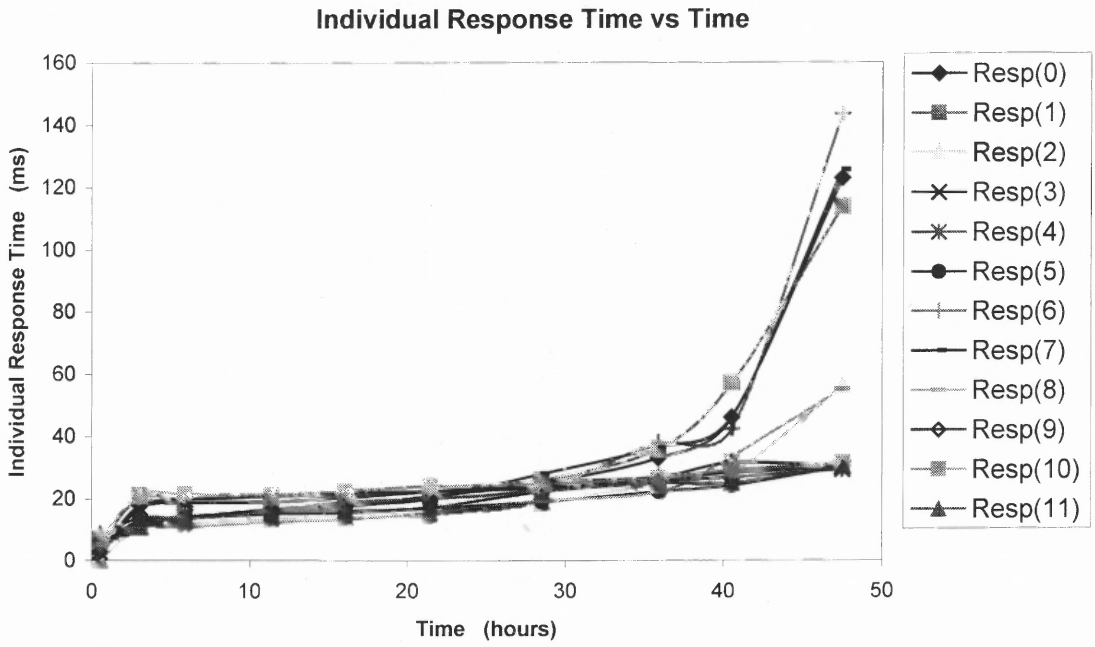

Figure 3.7 Individual response time over time.

Figure 3.7 depicts the behavior of the read response time for the individual disks that make up the storage system. The curves follow the same trend and response time remains steady in a wide range of time. This implies the system works in a stable manner.

CHAPTER 4

CONCLUSION

Performance studies of various data allocation methods were conducted on a Heterogeneous Disk Array (HDA) system. An improved best-fit allocation algorithm was proposed and tested against the data allocation methods.

The trends, motivations and previous work were reviewed before describing the HDA architecture. Features of the new architecture include: (i) allowing multiple disks of various make and characteristics to coexist, (ii) using capacity and bandwidth to the maximum extent possible, (iii) allowing multiple RAID schemes to coexist on the same device, and (iv) balancing the disk loads in terms of both bandwidth and capacity utilization.

The data structures of the HDA architecture were described and a problem that deals with balancing the utilization in terms of both bandwidth and capacity was formulized. A possible solution to this problem was described in the form of an improved best-fit scheduling algorithm, and a data allocation method was modeled based on this solution. An implementation of the HDA architecture was used to investigate its performance under various data allocation methods. Simulation results showed that it is possible to balance the utilization of bandwidth and capacity at the same time, and therefore provides efficient usage of the available resources in a heterogeneous disk environment. The proposed best-fit data allocation algorithm outperformed the others and is more cost-efficient.

APPENDIX

RELIABILITY MODELING

Given a set of n disk drives with their Mean Time To Failure (MTTF): $\{m_1, m_2, \ldots, m_n\}$, and a target Mean Time To Data Loss (MTTDL), determine the level x, where $x \in \{0, 1, 5, 6\}$, for RAID, and determine the number of disks $1 \le g \le n$ such that a RAIDx array consisting of any subset of g disks has MTTDL $\ge$ target MTTDL.

The MTTDL of a disk array that can tolerate one failure is expressed as[1]

$$MTTDL = MTTF_{RAID5} = \frac{(MTTF_{disk})^2}{N(G-1)MTTR_{disk}} \qquad (A.1)$$

where N is the total number of disks in the array, G is the number of disks in a RAID group (i.e. a set of disks over which a parity is computed), MTTFdisk is the mean time to failure of a component disk, MTTRdisk is the mean time to repair of a component disk, which is in fact the rebuild time of a disk array. This model assumes that disk failure rates are identical, independent, and exponentially distributed random variables. In arrays that maintain one or more on-line spare disks, the repair time can be very short, usually less than an hour, and so that the MTTDL can be very long and exceeds the normal projected disk deployment intervals (five years).

REFERENCES

1. Patterson, D., Gibson, G., & Katz, R.A. (1988). *A case for redundant arrays of inexpensive disks (RAID)*, Chicago: Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD), 109-116.

2. Holland, M. (1994). On-line Data Reconstruction in Redundant Disk Arrays. Carnegie Mellon University: Technical report cmu-cs-94-164.

3. Hennessy, J.L., Patterson, D.A., & Goldberg, D. (2003). Computer Architecture: A Quantitative Approach (3rd ed.). Morgan-Kauffman Publishers.

4. American National Standard for Information Systems—Small Computer System Interface (SCSI), (1986). ANSI X3.132-1986, New York.

5. American National Standard for Information Systems—High Performance Parallel Interface—Mechanical, Electrical, and Signalling Protocol Specification. (1991). ANSI X3.1983-1991, New York.

6. Gibson, G.A. (1992). Redundant Disk Arrays: Reliable, Parallel Secondary Storage. MIT Press. Boston.

7. Merchant, A. Yu, P. (1993) Performance Analysis of a Dual Striping Strategy for Replicated Disk Arrays. San Diego: Proceedings of the Second International Conference on Parallel and Distributed Information Systems, 148-157.

8. Peterson, W. Weldon, E. (1972) Error-Correcting Codes. MIT Press. Boston.

9. Lee, E. Katz, R. (1991). *Performance consequences of parity placement in disk array.* Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, 190-199.

10. Muntz, R.R. Lui, J.C.S. (1990). *Performance analysis of disk arrays under failure.* Brisbane, Australia: Proceedings of the 16th VLDB Conference, 162-173.

11. Holland, M. Gibson, G.A. & Siewiorek, D.P. (1994). Architectures and algorithms for on-line failure recovery in redundant disk arrays. Journal of Distributed and Parallel Databases, 2(3), 295-335.

12. Allen, N. (2001). Don't waste your storage dollars: What you need to know. Research note COM-13-1217, Gartner Group.

13. Paquet, R. Nicolett, M. (2001). The cost of storage management: A sanity check. Research note DF-14-6838, Gartner Group.

14. Han, C. (2004). Studies of Disk Arrays Tolerating Two Disk Failures and a Proposal for a Heterogeneous Disk Array. New Jersey Institute of Technology Ph.D. Dissertation.

15. Wolf, J. (1989). *The placement optimization program: a practical solution to the disk file assignment problem.* Berekely, CA: *Proceedings of the 1989 ACM SIGMETRICS International Conference*, 1-10.

16. Scheuermann, P. Weikum, G. & Zabback, P. (1994). Disk cooling in parallel disk systems. IEEE Data Engineering Bulletin, 17(3), 29-40.

17. Scheuermann, P. Weikum, G. & Zabback, P. (1998). Data partitioning and load balancing in parallel disk systems. VLDB Journal, 7(1), 48-66.

18. Lavenberg, S.S. (editor). (1983). Computer Performance Modeling Handbook. Academic Press.