Spring 5-31-2005

Rebuild performance enhancement using onboard caching and delayed vacation termination in clustered raid 5

Akheel Ahmed
New Jersey Institute of Technology

ABSTRACT

REBUILD PERFORMANCE ENHANCEMENT USING ONBOARD CACHING AND DELAYED VACATION TERMINATION IN CLUSTERED RAID 5

by
Akheel Ahmed

The Clustered Raid 5 (CRAID5) architecture with a parity group size(G) smaller than the number of disks(N) increases the load by the declustering ratio denoted by $\alpha = (G - 1)/(N - 1)$, which can be lesser than that in Raid 5 while switching to, and subsequently operating in rebuild mode. The Nearly Random Permutation (NRP) layout provides the flexibility to vary the declustering ratio (α) for a given N, and the Vacationing Server Model (VSM) of processing the rebuild requests provides acceptable rebuild and user response times.

The rebuild performance and the user response time can be improved by introducing an onboard buffer in the disks, which caches a single track upon arrival of a rebuild request while in rebuild mode. Such an enhancement is proposed, and the architecture is described along with an analysis using the DASim simulation toolkit developed at NJIT.

Also proposed is the delayed termination of vacations with two user requests as this improves the rebuild performance with a negligible negative impact on user response time. Finally, the effect of limiting the rebuild buffer on the rebuild performance is presented in the context of three different disk utilizations and declustering ratios.

REBUILD PERFORMANCE ENHANCEMENT USING ONBOARD CACHING AND DELAYED VACATION TERMINATION IN CLUSTERED RAID 5

by
Akheel Ahmed

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

Department of Computer Science

May 2005

APPROVAL PAGE

REBUILD PERFORMANCE ENHANCEMENT USING ONBOARD CACHING AND DELAYED VACATION TERMINATION IN CLUSTERED RAID 5

Akheel Ahmed

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

Dr. Teunis J. Ott, Committee Member — Date
Professor of Computer Science, NJIT

Dr. James M. Calvin, Committee Member — Date
Associate Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Akheel Ahmed

Degree: Master of Science

Date: May 2005

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2005

- Bachelor of Engineering in Computer Science,
  Visveswaraiah Technological University, India, 2002

ACKNOWLEDGMENT

I would like to express my deepest gratitude to Dr. Alexander Thomasian, who is an excellent research supervisor and a wonderful mentor who has provided me constant support, encouragement and countless valuable resources. Special thanks are given to Dr. Teunis J. Ott and Dr. James M. Calvin for participating in my committee.

Many of my fellow graduate students in the Integrated Systems Research Laboratory are deserving of recognition for their support. I especially thank Gang Fu for the help he has provided with DASim, an excellent disk simulation toolkit that he developed at NJIT.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Objective

Clustered Raid 5(CRAID5) has been the model of choice in database and *OLTP* (Online Transaction Processing) applications that have a high access rate to relatively small amounts of data. Although the performance of existing CRAID5 systems is acceptable, integrating an onboard-cache on the constituent hard disks improves both rebuild time as well as response time of the system. One of the objectives of this thesis is to propose this enhancement in processing rebuild requests. This is supported by performance analysis of the CRAID5 architecture with and without onboard caching.

When a quicker rebuild is critical, usually a tradeoff has to be made with the user response time. The document proposes an efficient way in which two user requests can be used to terminate the vacation of the system, resulting in a significant improvement in rebuild performance with a negligible impact on user response times.

Finally, the impact of limiting the rebuild buffer on rebuild performance is analyzed, and the effect of declustering on rebuild performance is presented. All performance analysis is done using the DASim[18] Disk simulator developed at NJIT.

1.2 Background Information

This section of the document presents information that serves as the foundation for subsequent chapters. An overview of hard disk drive technology along with a description of the various levels of redundant disk Arrays is presented in the following subsections.

1.2.1 Hard Disk Technology

Figure 1.1 depicts a disk drive as an assembly of a stack of platters that are mounted on a spindle.[1] The platters are double-sided, meaning they have magnetic coating on both surfaces. All platters rotate at a constant velocity which is measured in revolutions per minute (RPM). Each of the platter surfaces has a read-write head, which is in close proximity to the surface. All disk arms are mounted on an actuator, which moves the heads in unison. Only one of the heads is active at any given time since it is impossible to position all heads at the same time on corresponding tracks due to thermal variations in the platters and disk arms.[1]

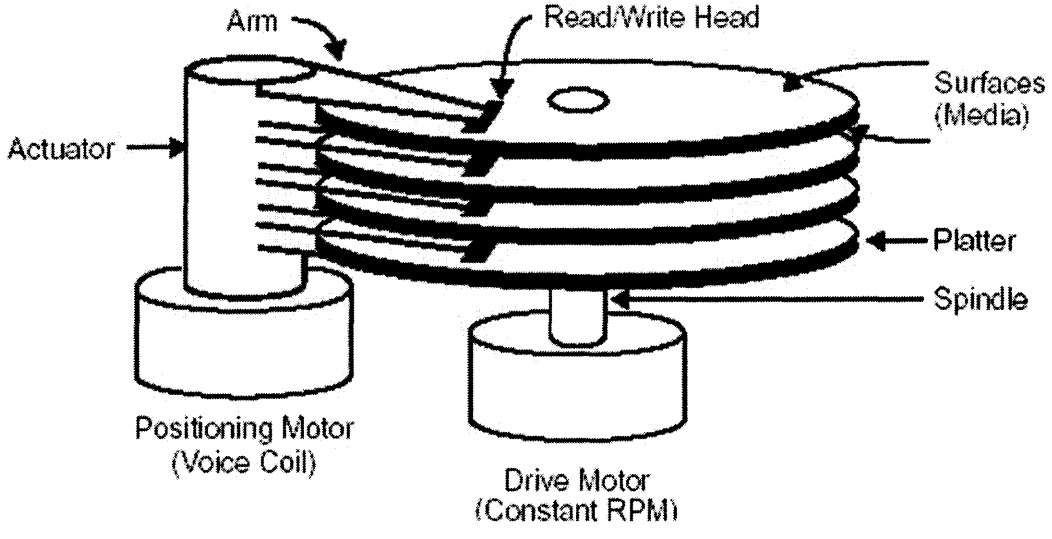

Figure 1.1 Physical structure of a hard disk.

Data storage in a hard disk is organized into cylinders, tracks and sectors. A sector is a block of sequential data, which is almost always 512 KB. It is the smallest unit of data that can be read from or written onto the disk. A sector is preceded by sector header that contains sector identification and clock synchronization information, and followed by a trailer that contains an error correcting code which is computed over the header and sector. All sectors on a platter surface that are equidistant from the spindle form a track. A set of tracks that are equidistant from the spindle form a cylinder. The sectors are numbered sequentially and form a linear address space.[1] The numbering begins with 0. This is illustrated in Figure 1.2.[1]

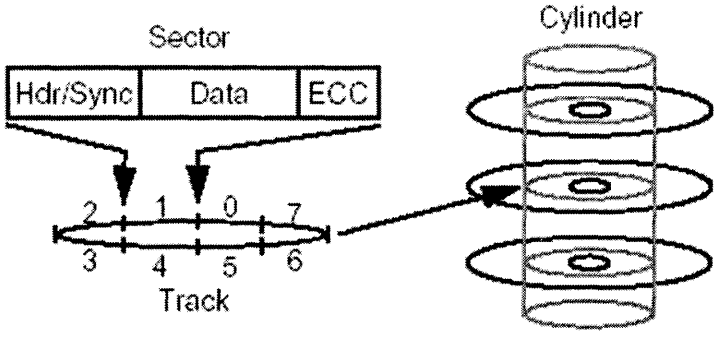

Figure 1.2 Grouping data into sectors, tracks and cylinders.

When data is being accessed, the actuator is moved to place the heads on the correct cylinder. After the sought after sectors come under the head, the data is read or written. The time it takes to move the actuator is called the *seek time*, which varies from 1 to 15 ms, depending on the seek distance. A read or write is performed by first seeking to the desired cylinder and then by *rotating* the disk to bring the starting sector under the head. Seeking and rotating constitute the *positioning* operation. When the request is to

access one full track, the read/write operation can be done without having to wait for the first sector to come under the head. The read/write can start as soon as the head is positioned over the right track. The sectors are accessed in the order they appear under the head. This is called a zero-latency operation, since there is no rotational latency. This operation is also employed when only a part of the track is being accessed.

Since the circumference of tracks farther from the spindle is more than those closer to it, more sectors are accommodated per track in the outer tracks than in the inner ones. This technique is called *Zone Bit Recording (ZBR)*. Here, 50 to 200 adjacent cylinders are grouped into *zones*, where each zone has fixed number of sectors per track, with this number higher in the outer zone than in the inner ones.

Figure 1.3 elucidates the arrangement of sectors, tracks and cylinders and the assignment of data. [1]

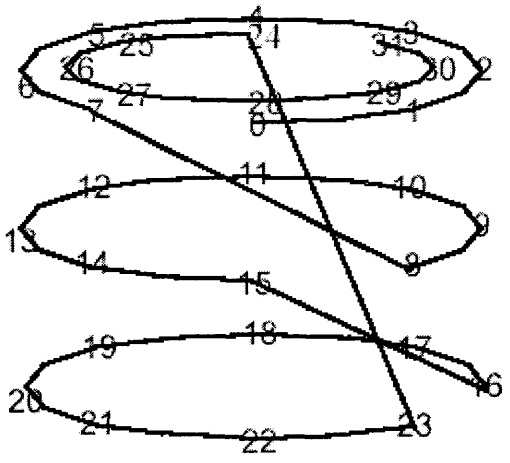

Figure 1.3 Sequential layout of sectors.

Amdahl's law illustrates that the performance of computer systems is limited by the performance of the I/O subsystems.[2] This is evident from the rate at which CPU performance has been increasing compared to storage performance. To utilize the improved CPU performance, there is a need to use parallelism in the I/O subsystem.

As shown in Figure 1.4, the current RAID systems are composed of disks that are connected via inexpensive low-bandwidth links to the array controller, which is connected to the host computer system using a high-bandwidth link.[3]

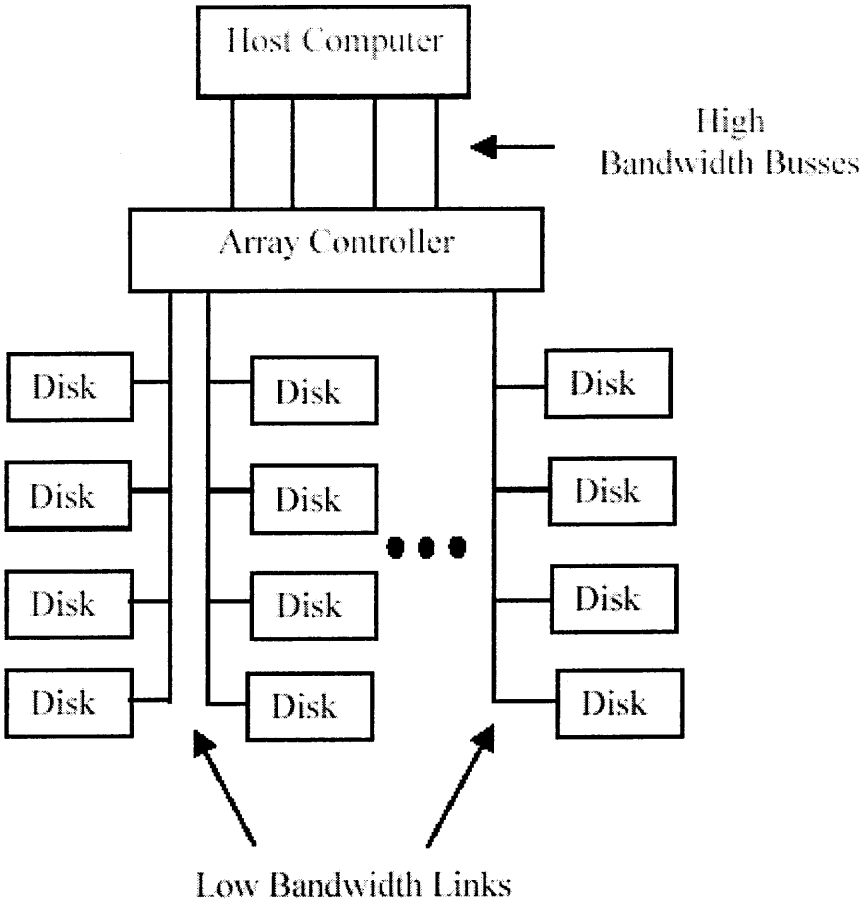

Figure 1.4 Disk-array architecture.

The array controller performs system-related operations such as controlling individual disks, maintaining address mapping and redundant information, and recovering from disk failures. The controller presents a linear address space to the host computer and maps this space to the individual disks. This mapping is known as the *data layout* of the array.

The array controller *stripes* (breaks up) contiguous data into units of a constant size called the *stripe units.*[4, 5, 6] These stripe units are assigned to consecutive disks. This provides load balancing in case of concurrent workloads and increased bandwidth for sequential transfer of large amounts of data by a single process. Disk arrays are normally classified into RAID 1 through 5, depending on the organization of redundant information and the layout of data.[4]

There is a RAID level 0, also called a non-redundant disk array. In this organization, there is only striping of data, but no duplication. This results in improved throughput. This layout is depicted in Figure 1.5.

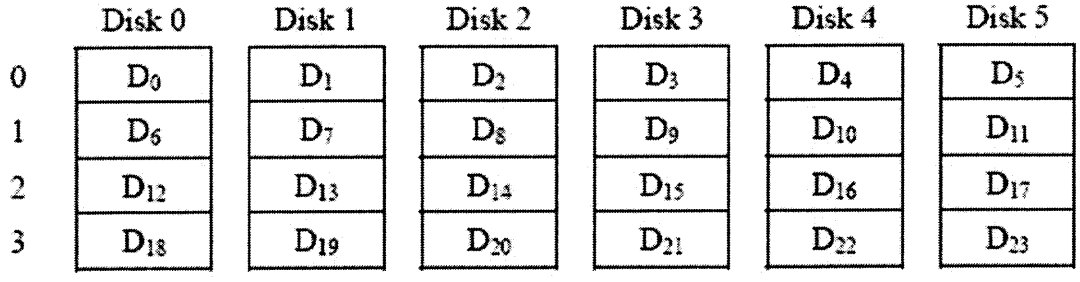

Figure 1.5 RAID level 0 (Non-redundant).

In RAID level 1, the disks are organized into pairs where each pair has identical data units. The data is striped across the *mirror* pairs. Figure 1.6 elucidates the organization of data in this RAID level. The highlighted stripes are for redundancy.

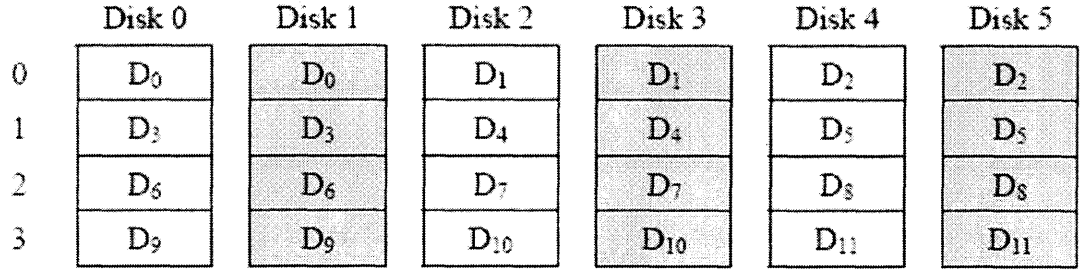

Figure 1.6 Data organization in RAID level 1 (Mirroring).

RAID 2 consists of disks that serve as data disks and check disks. The data disks use bit or byte striping, and the check disks use the Hamming code for error correction[8] as shown in Figure 1.7. The highlighted blocks/stripes are for parity or error correction.

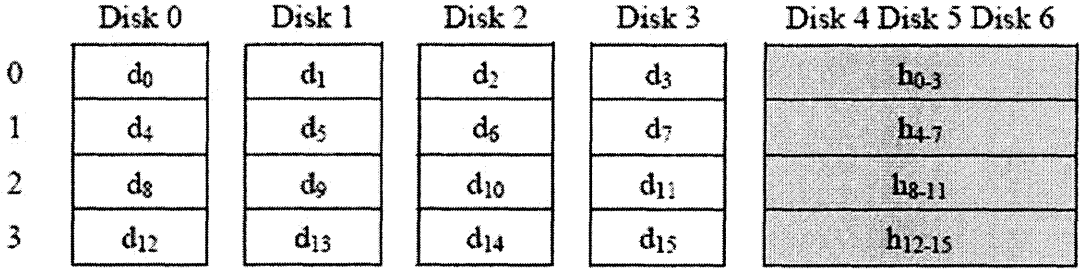

Figure 1.7 RAID 2 (Hamming Error Correction code).

In RAID level 3 a bit or byte interleaved parity is used. The data is striped across the data disks and a single parity disk stores the exclusive-or over the corresponding bits of the data disks. RAID level 4 is similar to RAID 3, the only difference being the use of

block-interleaved parity instead of one at the bit or byte level. These two RAID levels are illustrated in Figures 1.8 and 1.9.

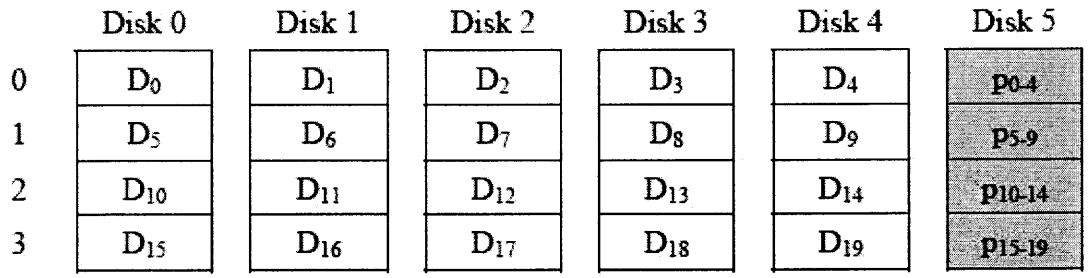

Figure 1.8 RAID level 3 using a byte-interleaved parity.

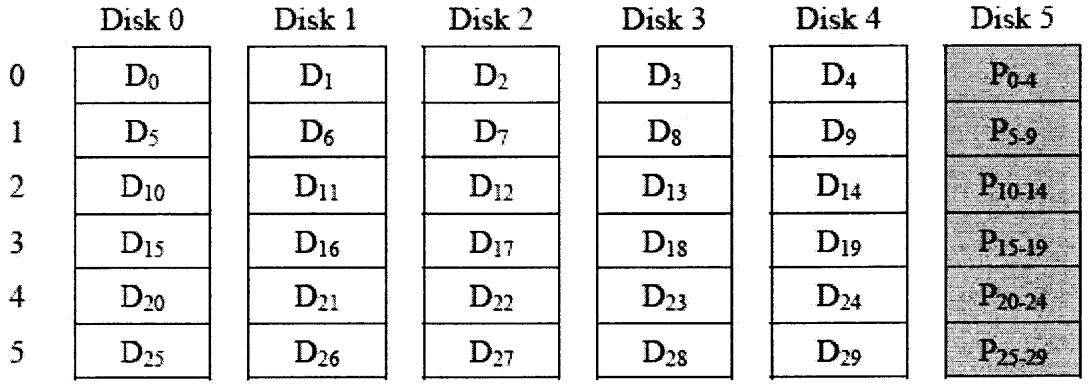

Figure 1.9 RAID level 4 using block-interleaved parity.

RAID 5 uses rotated block-interleaved parity, and the parity blocks are distributed in all disks. The RAID 5 considered here has a *left symmetric organization.*[9] In a left symmetric organization, the parity blocks are placed along the diagonal and the consecutive data stripe units are placed on the consecutive disks at the lowest available offsets. Parity is computed over a group of disks, called the *parity group.* In all the

above RAID levels, there is only one parity group. This is also true for RAID 5 as illustrated in Figure 1.10.

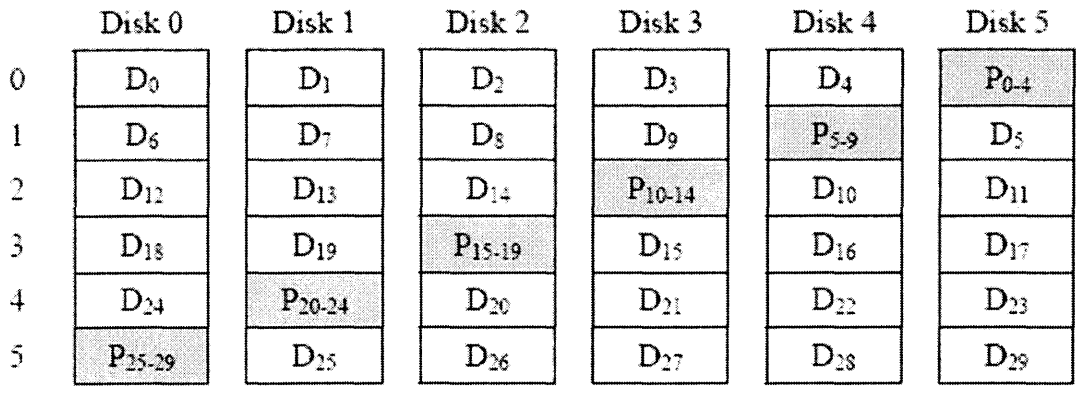

Figure 1.10 RAID level 5 (Rotated Block Parity).

The reliablility of a single-disk failure tolerant array can be measured as the *mean time to data loss* (MTTDL). It is given by the following expression.[4]

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

Here N is the total number of disks in the array, G is the number of disks in the parity group, $\mathbf{MTTF_{disk}}$ **is** the mean time to failure of a component disk, which is typically one million hours, and $\mathbf{MTTR_{disk}}$ is the mean time to repair a disk, which is typically a few hours.

It is possible to have more than one parity groups in a RAID level. If a RAID level has more than one parity group, it is called *Clustered RAID*, and the technique is called *Declustering*. Clustered RAID 5 is described in the following section.

1.2.2 Overview of Clustered RAID 5 Architecture

The *Stripe Units* (SUs) in RAID5 balance the workload on disks in case of a single disk failure. They are allocated in a round-robin manner across all the N disks. Each parity block is the Exclusive-OR (XOR) of the corresponding N-1 blocks. So the parity information occupies storage worth one full disk.

These parity blocks are kept up-to date, and this incurs a *small write penalty* when small randomly placed data blocks are updated. A non-volatile RAM can be used to cache modified blocks and then these data and parity blocks can be written at a lower priority than read requests.

During a disk failure, the requested data block on a failed disk is reconstructed by XORing the corresponding N-1 blocks on the surviving disks by an N-1 fork-join operation. The increase in load is at worst 100% when all requests are read. If the utilization is already high in normal mode, disk saturation will occur in degraded mode. This is possible in case of an infinite source model where the arrival rate of requests is not reduced due to disk failure and increased response time.

Clustered RAID[7] trades hard disk capacity against increase in load. The number of stripe units over which parity is computed, also called the parity group size denoted by G, is the same as the number of disks N in RAID5. In clustered RAID, G can be less than N.

Hence, to reconstruct a block in clustered RAID, only a fraction of the disks need to be accessed instead of all the N disks as in RAID5. This fraction, also called the *declustering ratio* (α) is given by the following relation: $\alpha = (G - 1)/ (N - 1)$. It is obvious that in RAID5 $\alpha = 1$.

There are two approaches to balancing the disk load during degraded mode. These are the *Balanced Incomplete Block Design (BIBD)* [10, 11] and the *Nearly Random Permutation (NRP)* [12] layouts. The NRP layout provides more flexibility than the BIBD layout. Hence it is described here and considered in the rest of the document.

The NRP layout[13] is described as follows: The disk array is considered as a two-dimensional array with N columns, equaling the number of disks, and M rows, each of which has N stripe units. There are NM stripe units numbered 0 to NM – 1. The parity group size is G, which is smaller than N. Parity group *i* occupies stripe units iG to iG + G – 1. This is called the initial logical organization. Figure 1.11[13] shows the layout for N = 10 and G = 4.

Disk numbers	0	1	2	3	4	5	6	7	8	9
Parity groups (initial allocation)	D0	D1	D2	P0-2	D3	D4	D5	P3-6	D6	D7
	D8	P6-8	D9	D10	D11	P9-11	D12	D13	D14	P12-14
	D15	D16	D17	P15-17	D18	D19	D20	P18-20	D21	D22
	D23	P21-23	D24	D25	D26	P24-26	D27	D28	D29	P27-29

Figure 1.11 NRP initial logical allocation for N = 10 and G = 4.

For each row *I*, a random permutation of {0, 1, 2,....., N-1} is generated, which is $\{P_0, P_1, P_2, \ldots\ldots P_{N-1}\}$. Here *I* is used to seed the random number generator. When *mod* (N, G) = 0, the above permutation operation should be repeated M times, otherwise it should be repeated K times where K = *LCM* (N,G) / N. LCM(N,G) is the least common multiple of N and G.

For example, if P_1 = {0, 9, 7, 6, 2, 1, 5, 3, 4, 8} then there will be the following data allocation[13] for the first two rows since K = 2. The data allocation is shown in Figure 1.12. There will be approximately the same number of parity blocks per row.

Disk numbers	0	1	2	3	4	5	6	7	8	9
Final allocation	D0	D4	D3	P3-6	D6	D5	P0-2	D2	D7	D1
	D8	P0-11	D11	D13	D14	D12	D10	D9	P12-14	P6-8

Figure 1.12 NRP data layout after permutation.

The next chapter presents the various ways in which the rebuild performance and user response time can be affected in CRAID5.

CHAPTER 2

REBUILD PROCESSING

2.1 Introduction to Rebuilding

Rebuild processing is the systematic reconstruction of data on the failed disk, on a hot spare. It is begun as soon as the hot spare is available. The disk then changes its mode of operation from degraded mode to rebuild mode. The smallest unit of data that is rebuilt as a whole is called the *Rebuild Unit(RU)* which is usually a stripe unit or a fraction of it.

The time taken to rebuild the failed disk $T_{Rebuild}(\rho)$, and the response time for user requests $T_{Response}(\rho)$ where ρ is the disk utilization in normal mode are the two main yardsticks to measure the performance of the system. After a disk failure, the RAID5 array operates at a higher disk utilization $\rho' = \beta\rho$ with $\beta = \alpha + 1$ where $\alpha = (G - 1)/(N - 1)$ when all requests are reads.

There are two kinds of rebuild[10] – *Stripe oriented* and *Disk oriented.* In a stripe oriented rebuild, each RU is rebuilt by a dedicated process that reads RUs from the surviving disks, XORs them and writes the resulting RU on the spare disk. Disk oriented rebuild dedicates a process to each disk that reads RUs from the surviving disks in an asynchronous way. This requires a larger buffer compared to stripe oriented rebuild when the number of processes are small.

Disk oriented rebuild has been known to outperform stripe oriented rebuild[10]. For this reason, the rest of the document restricts itself to disk oriented rebuild. There are two types of buffer, temporary buffers dedicated to reading from each disk and buffers dedicated to writing on the spare disk.[13]

The unit of buffering is a rebuild unit (RU), and the buffers are interchangeable. As soon as an RU is read into the temporary buffer, it is XORed with the corresponding RUs and the result is stored in the spare disk buffer. The XOR operation is very fast and the buffers are never exhausted. For this reason, the reading of RUs from disks during a disk-oriented rebuild is not stopped due to buffer limitation. The buffer for the spare disk, however, has an effect on the rebuild performance, and it is thoroughly investigated in this study. All future references to buffer imply the buffer for the spare disk.

Rebuild requests can be processed in two ways – at the same priority as user requests and at a lower priority than user requests. If they are processed at the same priority as the user requests, the processing model is called a *Permanent Customer Model* (PCM). Here only one request is processed at a time - a new rebuild request is inserted at the end of the queue as soon as the previous one is served.

Rebuild can also be performed in the following way. When there is no user request pending (i.e. as soon as the disk becomes idle) rebuild is started and is stopped when a user request arrives. This mode of rebuilding is called the *Vacationing Server Model* (VSM). Performance comparisons of VSM and PCM have shown VSM to give shorter rebuild and response times.[13] For this reason, only VSM is considered in this document.

Rebuild operations in RAID5 can further be classified into the following types.[7] In a *Baseline rebuild*, the materialized blocks of the spare are updated but are not used to satisfy read requests. If *Read Redirection* is used, the materialized blocks of the spare disk are not only updated on the disk but are also used to satisfy read requests resulting in

shorter response times for reads accessing the failed disk, lower utilization of existing disks and shorter rebuild time.

If *Piggybacking* is used at the block level, the reconstructed data are written to the spare disk when a read is targeted to the failed disk. This improves the performance only when used at the track level.[14] There are a few policies that improve user response times by pre-empting rebuild requests[15]. *Split seek* calls for pre-empting the read operation after the seek is completed if a user request is pending. In the absence of a user request, however, consecutive tracks are read until a user request arrives. *Split latency/transfer* proposes pre-emption even after seek and transfer have started. In this study, none of these options are considered since the improvement in response time is at the expense of rebuild time.

2.2 Performance Evaluation of Clustered RAID5

In the past, analytical modeling has been widely used to evaluate storage systems. The M/G/1 queuing model provides very accurate estimates of mean response times, even with *zoned disks* when rebuilding using the VSM policy.[13]

Analytical modeling has several shortcomings, the most important of them being the inability to incorporate passive resources such as buffers. The rest of this document is based on simulation of the Clustered RAID5 system using DASim.

This simulator uses a detailed single disk simulator which can simulate different disk drives by reading their characteristics from a specs file.[16] All further investigation pertains to the IBM 18ES disk drive which has a capacity of 9.17 GB and spins at 7200 RPM, resulting in a rotational time of 8.33 ms. It is comprised of 11 zones and the

number of sectors per track varies from 247 to 390. The average seek time is 7.16 ms and the average access time is over 11 ms. For reasons of efficiency, an OLTP workload is used[17] as 96% of he requests will be for 4 KB blocks and the remaining 4% will be for 24 KB blocks. This model is simplified and it is assumed that all disk requests are for 4 KB blocks. This simplification is possible because the positioning time dominates the service time. Also, the arrival process is assumed to be Poisson as it allows for varying the arrival rate of requests and obtaining the mean response time characteristic of the system in both normal and degraded modes. The rebuild is assumed to start immediately in degraded mode (i.e. as soon as a disk fails). The effect of VSM versus PCM for a declustering ratio $\alpha = 0.75$ is shown in Figure 2.1.[13]

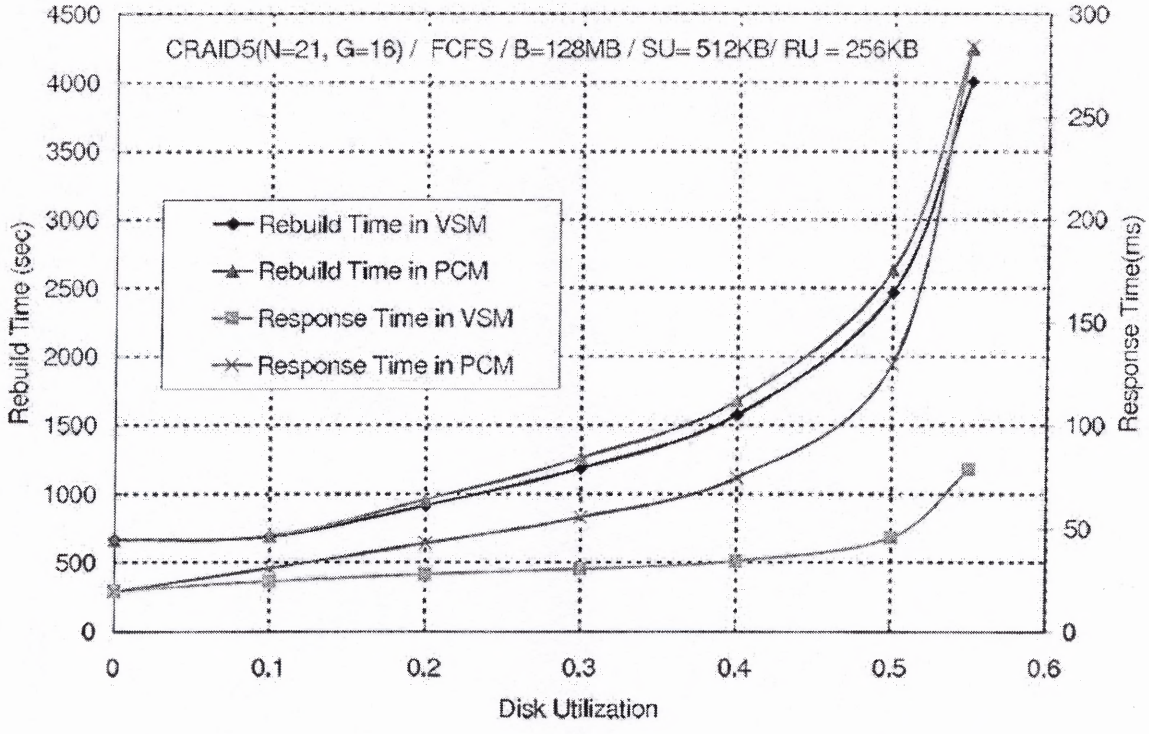

Figure 2.1 Mean user response time and rebuild time in VSM and PCM for $\alpha = 0.75$.

Neither VSM not PCM allow pre-emption of rebuild requests. So when a user request arrives while rebuild is in progress, it has to wait until the rebuild completes. As can be seen in Figure 2.1, the user response time and the rebuild time in VSM are always shorter than in PCM. For this reason, only VSM is considered hereafter. Figure 2.1 is obtained by simulating the Clustered RAID5 system with 21 IBM 18ES disks (N). The parity group size (G) is 16, scheduling policy is *FCFS* (First Come First Serve), Buffer size is 128 MB, Stripe unit(SU) size is 512 KB and Rebuild unit size(RU) is 256 KB.

The effect of read redirection (RR) is shown in Figure 2.2.[13]

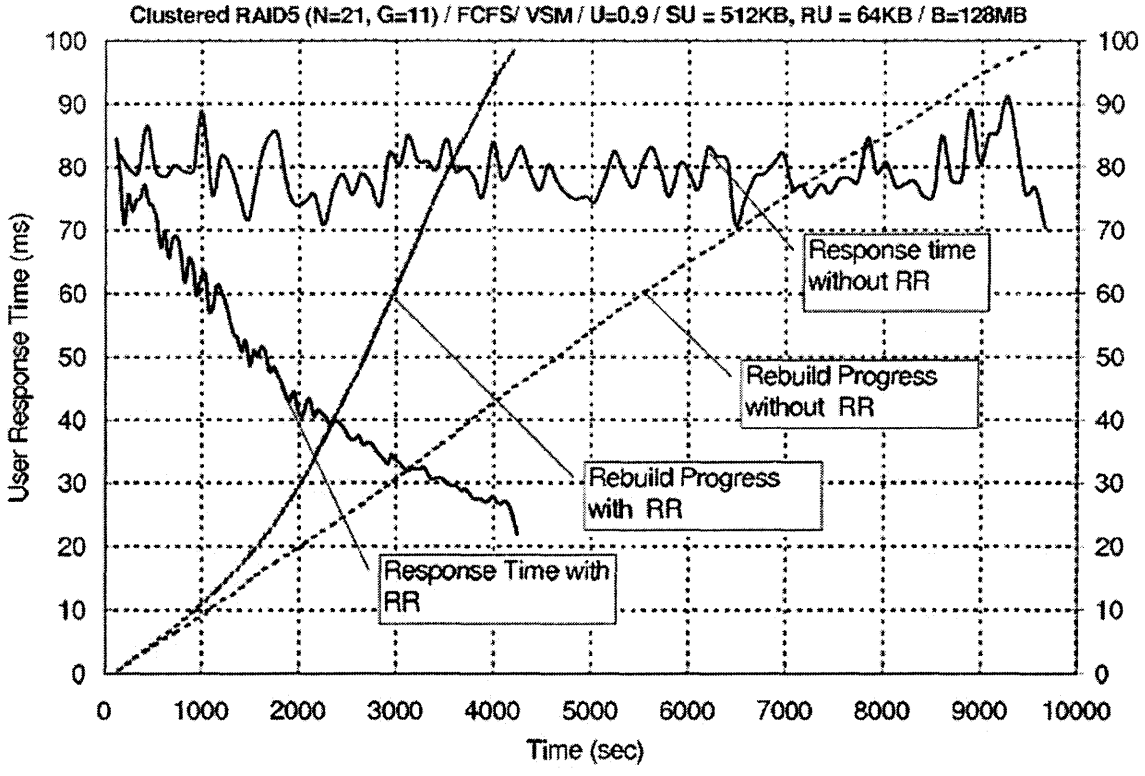

Figure 2.2 Effect of Read redirection on response time and Rebuild progress.

As can be seen in the above figure, read redirection shortens rebuild and response times significantly. It can, however, have a negative impact on the rebuild time when the

declustering ratio is very small as the surviving disks feed data to the spare disk at a rate higher than it can consume. This can be alleviated by dynamically controlling the fraction of read requests that are redirected to the spare disk. Figures[13] 2.3, 2.4, 2.5 and 2.6 illustrate the effect of using dynamic control of read redirection for declustering ratio α = 1, 0.75, 0.5 and 0.25,respectively.

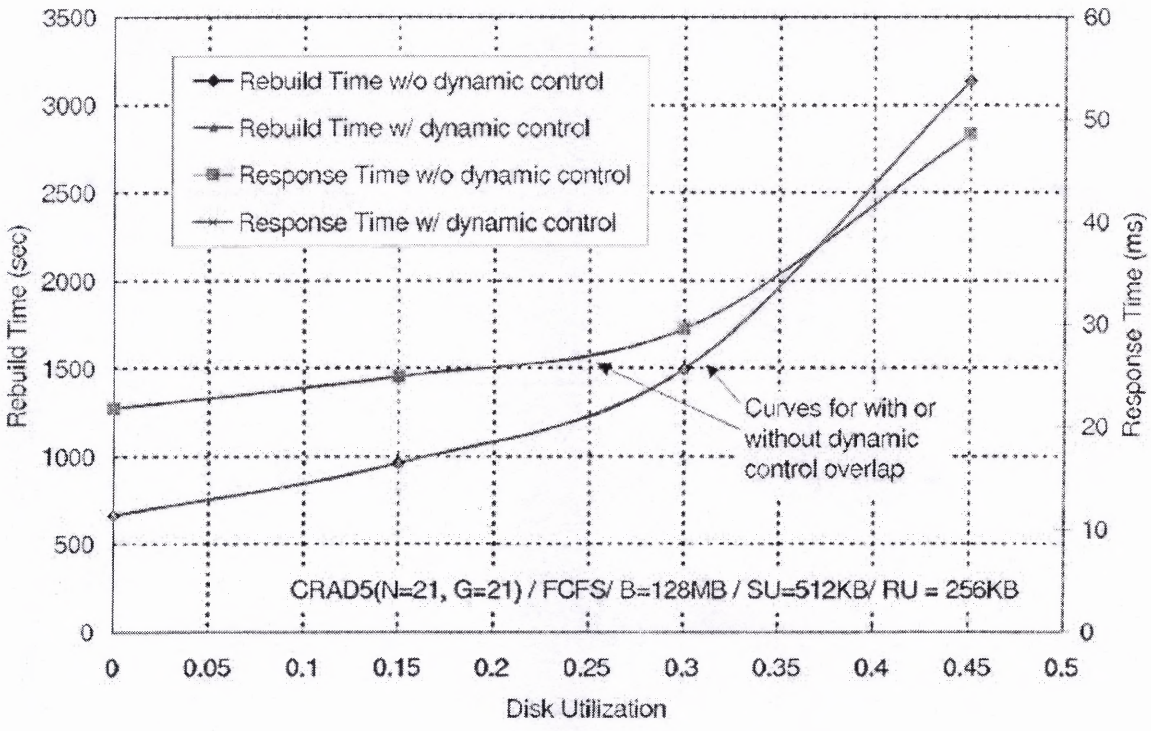

Figure 2.3 Effect of dynamic control of read redirection for α = 1.

As seen in Figure 2.3, dynamic control of read redirection has no effect on the rebuild or response times. The following figures show the effect when α is lowered.

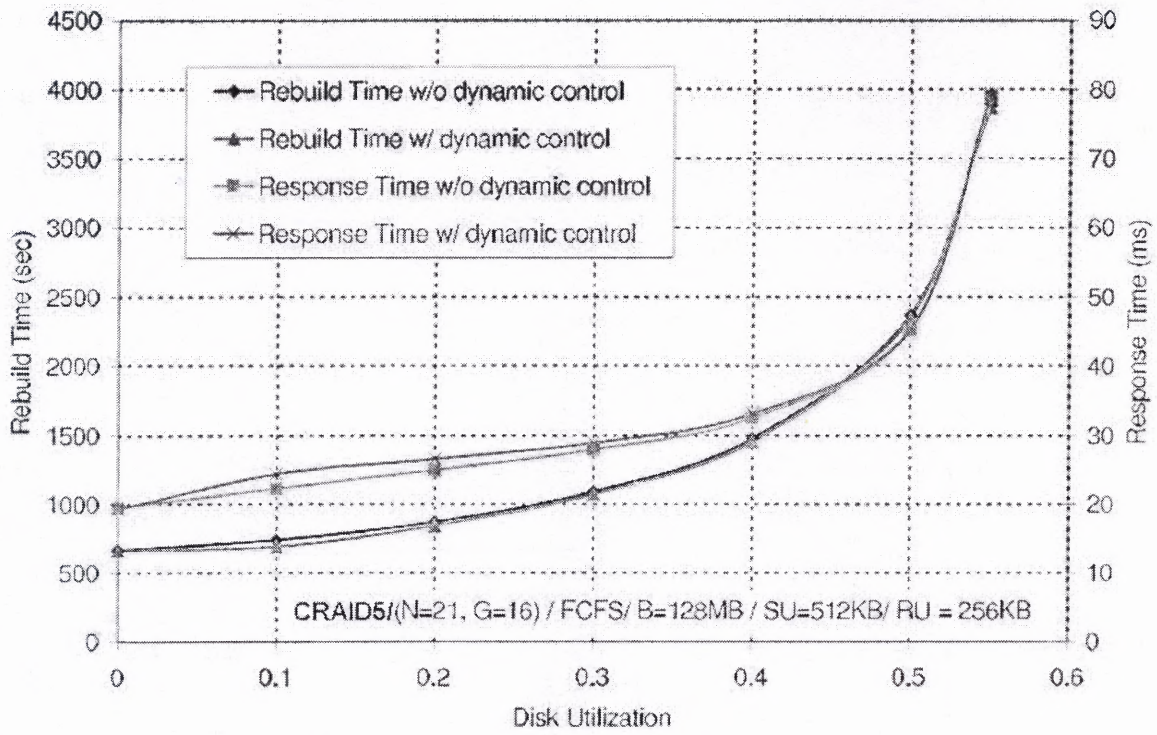

Figure 2.4 Effect of dynamic control of read redirection for $\alpha = 0.75$.

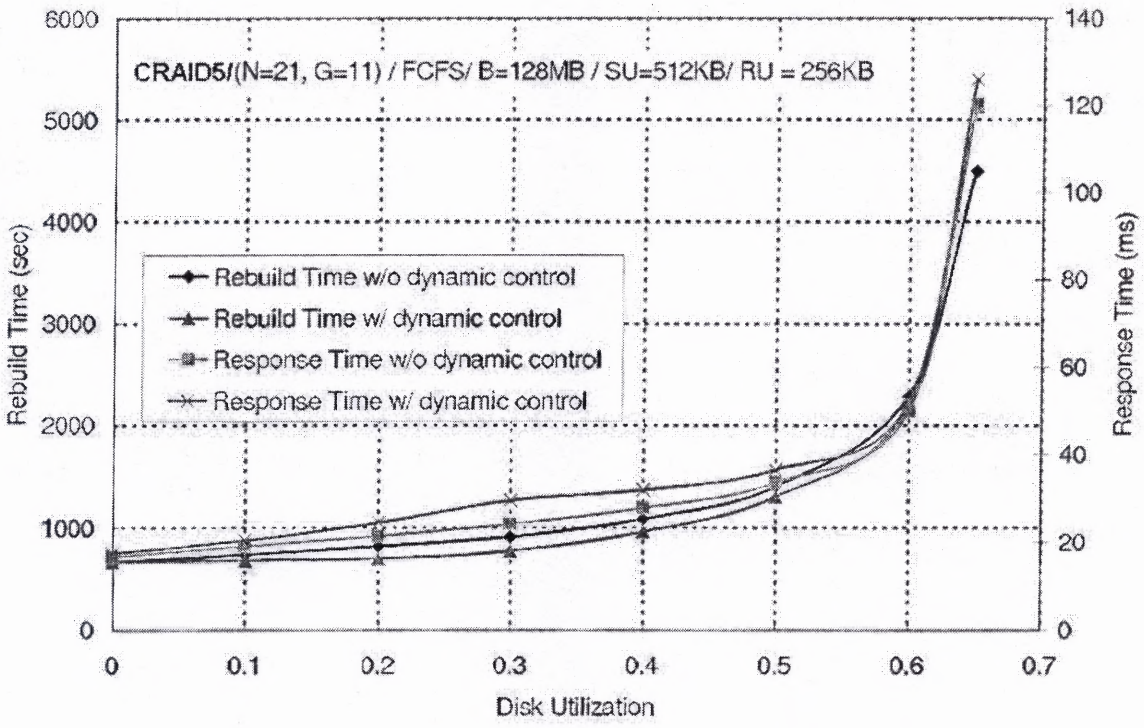

Figure 2.5 Effect of dynamic control of read redirection for $\alpha = 0.5$.

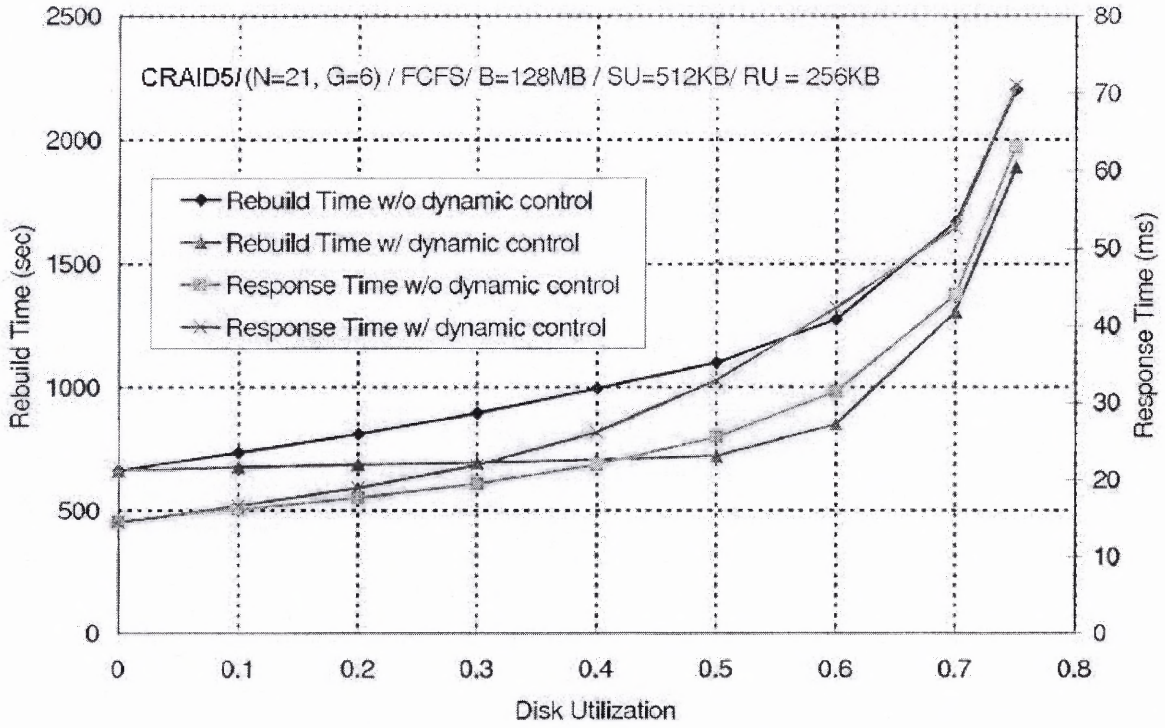

Figure 2.6 Effect of dynamic control of read redirection for $\alpha = 0.25$.

As seen in Figures 2.4, 2.5 and 2.6, dynamic control of read redirection has effect only for smaller values of α and at higher utilizations. The size of the rebuild unit has a significant impact on the rebuild and response times. A larger rebuild unit size leads to shorter rebuild time but a higher response time. This is due to the fact that a large rebuild unit takes longer to service during rebuild operation and the user requests have to wait longer before they can be serviced as there is no pre-emption of rebuild. This is illustrated in Figure 2.7[13] in the following page.

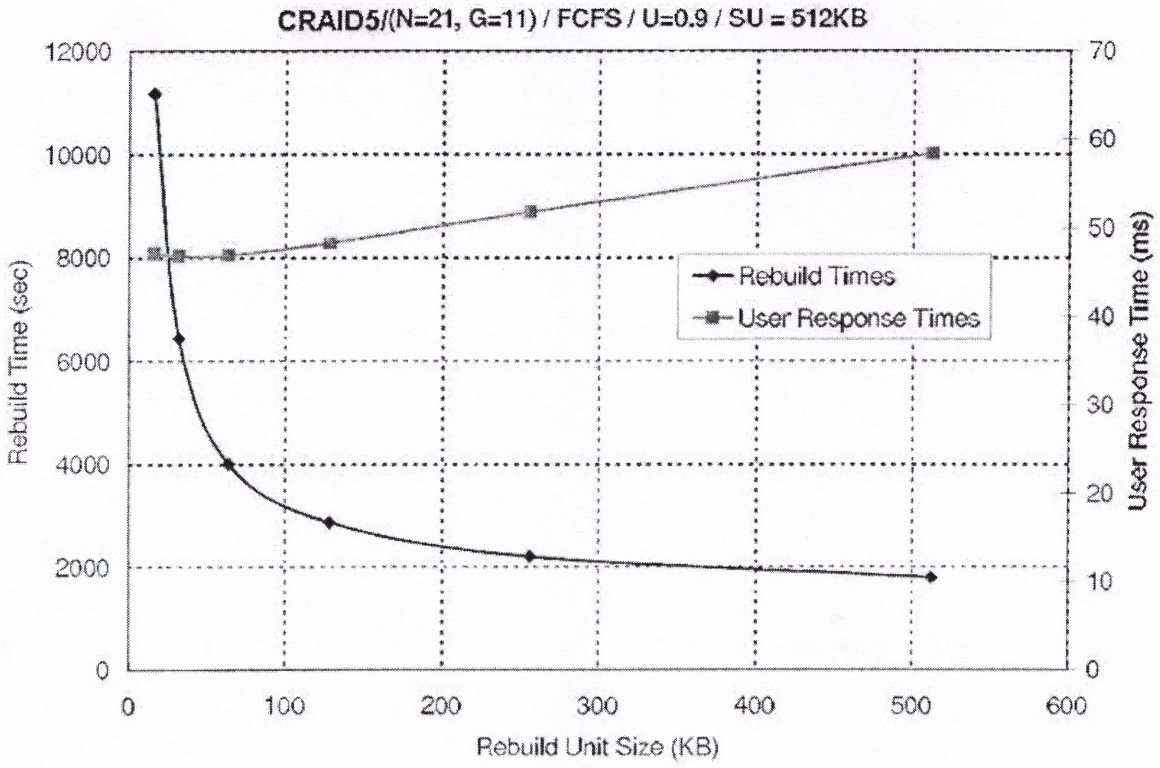

Figure 2.7 Impact of rebuild unit size on rebuild and user response times.

While the performance of Clustered RAID5 systems can be fine-tuned by varying the parameters discussed in this section, a significant improvement in rebuild and response times can be achieved by incorporating an on-board cache on the individual hard disks that constitute the disk array. This is the topic of discussion in the next chapter.

CHAPTER 3

ONBOARD CACHING IN CLUSTERED RAID5 DISKS

3.1 Advantages of Onboard Caching

A *Cache*, by definition is a memory that acts as a bridge between a small fast memory and a larger slower memory. It stores the data most frequently accessed by the larger memory from the smaller one. The cache is usually smaller in size than the smaller memory, but is much faster. This way, the frequently used data can be accessed much faster.

The onboard cache discussed here, as the name says, is one that is integrated into the hard disk. This cache is used to buffer a track during the rebuild operation. It can be enabled or disabled by the array controller at the user's discretion. This cache, as the later sections reveal, improves the rebuild performance and user response times significantly at an additional cost which is a small fraction of that of the entire hard-disk. It would be transparent to the array controller. Therefore, no major changes in the controller hardware are necessary to use the cache. The controller, however, should be capable of enabling or disabling the cache.

The next section discusses the features and design considerations of the onboard cache in detail. Also discussed in detail is the operation of the cache in rebuild mode.

3.2 Cache Architecture

The hard disk drive being considered is the IBM 18 ES that has a capacity of 9.17 GB with 11 zones and 247 – 390 sectors per track. It has a rotational speed of 7200 RPM. The major requirement for the cache is that it should be able to buffer the largest track (i.e. 390 sectors). Each sector is considered to be 512 Bytes long. So the size of the cache would be 512 x 390 Bytes. The smallest unit of data transferred into or out of the cache is a sector. The access times are neglected as they are very small compared to the disk access times. The data transfer times from the cache are also neglected as they are of the order of nanoseconds and those of the disk are in milliseconds.

The cache is used only in rebuild mode. When a user request arrives while the rebuild operation is going on, the track being read is cached. After the current rebuild unit is read, the user request is served by transferring the sectors directly from the cache, if there is a hit to the cache. Otherwise, a normal read operation is performed, without involving the cache.

When the user request arrives while the rebuild operation is reading a track, the unread part of the track is cached first. Then another read is issued to read the remaining sectors of the track. Following this, the rest of the rebuild unit is read. This fills the cache with one full track. Then after the rebuild unit is read, the user request is served directly from the cache if there is a hit. The cache is not used otherwise.

Figures 3.1 to 3.3 show the operation of the onboard cache when the user request arrives while rebuild is underway. Here S_0, S_1...S_n represent sectors 0 through n in the track being considered.

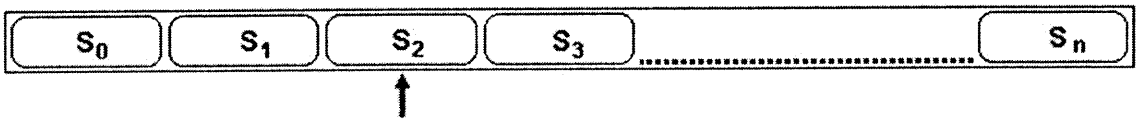

Figure 3.1 Arrival of user request while sector S_2 is being rebuilt in the above track.

At this point of time the buffering of the track starts. Since it is a zero-latency read operation, the entire track, starting from S_3 to S_n is read into the cache while the rebuild operation is underway. S_2 cannot be buffered since it would already have been in the midst of being read when the user request arrived. Buffering starts from the next sector in the track. This buffering is done simultaneously with the rebuild read operation. Figure 3.2 shows the contents of the onboard cache after these sectors have been read.

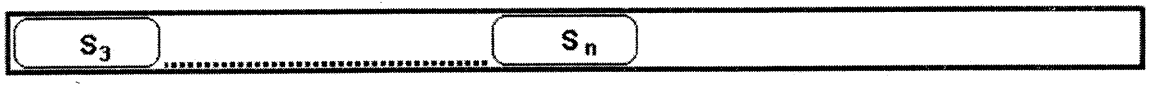

Figure 3.2 Cache contents after the track has been read once.

If the user request is for the entire track, sectors S_0, S_1 and S_2 are read now. This is shown in Figure 3.3.

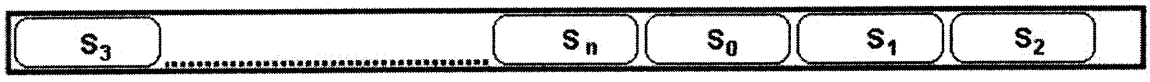

Figure 3.3 Cache contents after the track has been accessed twice.

The improvement in rebuild and response times depends heavily on the percentage of hits to the cache, which itself is dependent on the probability of having a user request for sectors in the current track being rebuilt. This probability in turn depends on the size of the rebuild unit being used, as can be seen in the simulation results in later

sections. In the discussion that follows, the cache is assumed to have zero latency for reads and writes. This is acceptable since cache delays are negligible when compared to disk access times. The following section discusses the data structures and algorithms supporting the operation of this onboard-cache.

3.3 Data Structures for CRAID5 Performance Analysis

The Clustered RAID5 (CRAID5) architecture involves the use of many data structures. Some of them are built upon the data structures pertaining to single disk operation. The following characterize the single disk.

- **Single_disk_info:** This structure, as the name says, has the information about the disk: The total number of blocks, block size, total number of tracks, the type (physical disk or virtual disk) and the queuing policy used. It is used by the single disk simulator in DASim, which in turn is used by the CRAID5 simulator as described later.
- **SDSim_framework:** Here, the functionality of the single disk controller is defined. It handles requests for read and write. It also stores disk state and performance monitoring information. It serves requests from a queue of disk access requests. There is a single queue for each disk. Also in our discussion, the queue is *FCFS*.

Following are the three most important data structures relevant to the operation of CRAID5.

- **CRAID5_info:** This characterizes the disk array with the number of disks, stripe unit size, group size and mode which is discussed later.

- **CRAID5_layoutmanager:** As the name implies, this structure manages the layout of the array. It is responsible for mapping of logical disks to physical disks (and also logical block address to a block address in a specific disk).
- **Rebuild_manager:** All issuing of rebuild requests is done here. Also, the rebuild read and write operations are handled here. In the following section, the performance of the CRAID5 architecture is analyzed with and without the cache, using various configurations that are commonly used.

3.4 CRAID5 Configurations and Simulation

The following configuration has been used in all the simulations. Two identical sets of simulations are performed, one with the cache and the other without it. In all cases the following parameters are constant.

Buffer Size = 128 MB

Read Redirection = Enabled

Model = VSM

Dynamic Control = Disabled

Disk Used = IBM 18ES (9.17 GB, 7200 RPM, 11 zones, 247-390 sectors per track)

Scheduling Policy = FCFS

Number of Disks(N) = 21

Parity Group Size (G) = 16, giving a declustering ratio $\alpha = (G - 1) / (N - 1) = 0.75$.

Stripe Unit Size = 128 Blocks = 512 KB, since each block is 4 KB

All disk accesses will be reads. The number of user requests to be processed before and after the rebuild are 10000. The system is simulated for post-failure disk utilizations from 0 to 96.25% (i.e. 0 to 55% before failure) with and without the onboard cache. Post-failure disk utilization (ρ') is calculated using expression 3.1.

$$\rho' = \rho\ (1 + \alpha) \tag{3.1}$$

Where ρ is the pre-failure disk utilization. The entire procedure is performed for rebuild unit (RU) size of 64, 128 and 256 KB. The graph in Figure 3.4 shows the difference in rebuild performance seen while using the cache. The rebuild unit size here is 16 blocks (i.e. 64 KB).

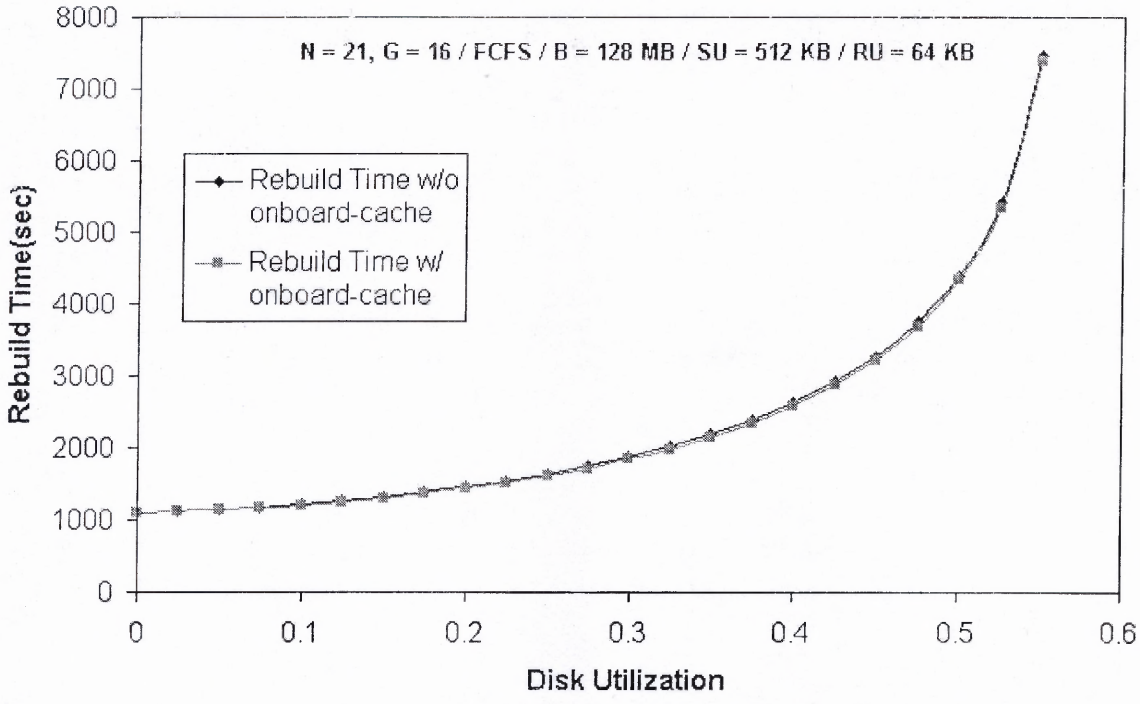

Figure 3.4 Effect of onboard-cache on rebuild time for rebuild unit size of 64 KB.

As seen in the previous figure, there is not much difference in the rebuild times with and without the cache. Figure 3.5 shows the effect on user response times.

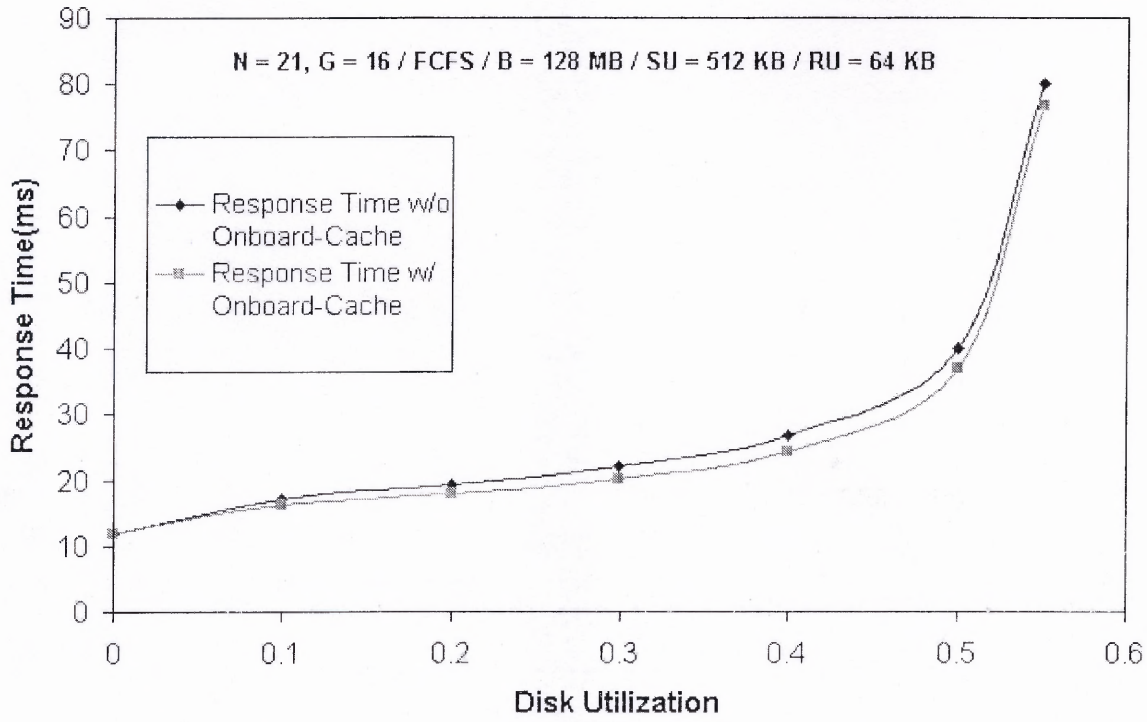

Figure 3.5 Effect of onboard cache on user response time for rebuild unit size of 64 KB.

The average number of times a track and sector is accessed with and without the cache justifies the difference in the rebuild and response times. If the cache is enabled, a track and sector will be accessed fewer times. Also, when the disk utilization is zero, the arrival rate of user requests is not absolutely zero. Also, a stripe oriented rebuild is used in this and the rest of the simulations since this utilizes the rebuild buffer in a more efficient way. Disk oriented rebuild does not fully utilize the rebuild buffer. The reason for a very small difference in the rebuild times becomes clear after analyzing the following graphs. Table 3.1 shows the differences in the rebuild and response times, average number of times a track and sector are accessed with and without the cache.

Table 3.1 Simulation Results with Rebuild Unit Size of 64 KB

U	t_{rb}	n_t	n_s	t_{rs}	T_{rb}	N_t	N_s	T_{rs}
0.000	1097	1.30	1.15	11.89	1096	1.30	1.15	11.86
0.025	1123	57.29	56.44	15.56	1119	56.24	56.13	15.37
0.050	1150	58.83	57.28	16.06	1142	57.74	56.75	15.72
0.075	1181	60.19	57.94	16.58	1170	59.06	57.19	16.09
0.100	1217	61.42	58.47	17.07	1203	60.25	57.5	16.42
0.125	1264	63.33	59.68	17.61	1246	62.12	58.49	16.81
0.150	1318	65.19	60.84	18.14	1297	63.94	59.43	17.19
0.175	1384	66.28	61.23	18.67	1360	65.00	59.60	17.56
0.200	1462	69.20	63.45	19.22	1434	67.88	61.61	17.96
0.225	1542	71.47	65.02	19.87	1511	70.11	62.96	18.46
0.250	1640	73.94	66.79	20.52	1606	72.54	64.51	18.95
0.275	1756	77.15	69.3	21.23	1718	75.71	66.80	19.51
0.300	1886	80.18	71.63	22.01	1845	78.70	68.91	20.14
0.325	2032	85.16	75.91	22.94	1987	83.64	72.97	20.91
0.350	2197	89.23	79.28	24.00	2149	87.67	76.12	21.82
0.375	2396	95.30	84.65	25.28	2345	93.70	81.27	22.95
0.400	2636	102.11	90.76	26.77	2581	100.47	87.17	24.29
0.425	2925	111.16	99.11	28.68	2867	109.48	95.30	26.04
0.450	3274	121.18	108.43	31.21	3213	119.46	104.4	28.42
0.475	3746	135.43	121.98	34.72	3681	133.67	117.73	31.78
0.500	4398	154.87	140.72	40.10	4330	153.07	136.25	37.00
0.525	5416	186.47	171.62	50.18	5344	184.63	166.93	46.93
0.550	7471	248.26	232.71	80.11	7396	246.38	227.80	76.71

Here, **U** is the disk utilization before failure occurs, $\mathbf{T_{rb}}$ and $\mathbf{t_{rb}}$ are the rebuild times in seconds, $\mathbf{T_{rs}}$ and $\mathbf{t_{rs}}$ are the response times in milliseconds, $\mathbf{N_t}$ and $\mathbf{n_t}$ are the average number of times a track is accessed, and $\mathbf{N_s}$ and $\mathbf{n_s}$ are the average number of times a sector is accessed with and without the onboard cache respectively. Figures 3.6

and 3.7 show the effect on the rebuild and response times when the rebuild unit size is 128 KB. Table 3.2 gives the simulation results for this rebuild unit size.

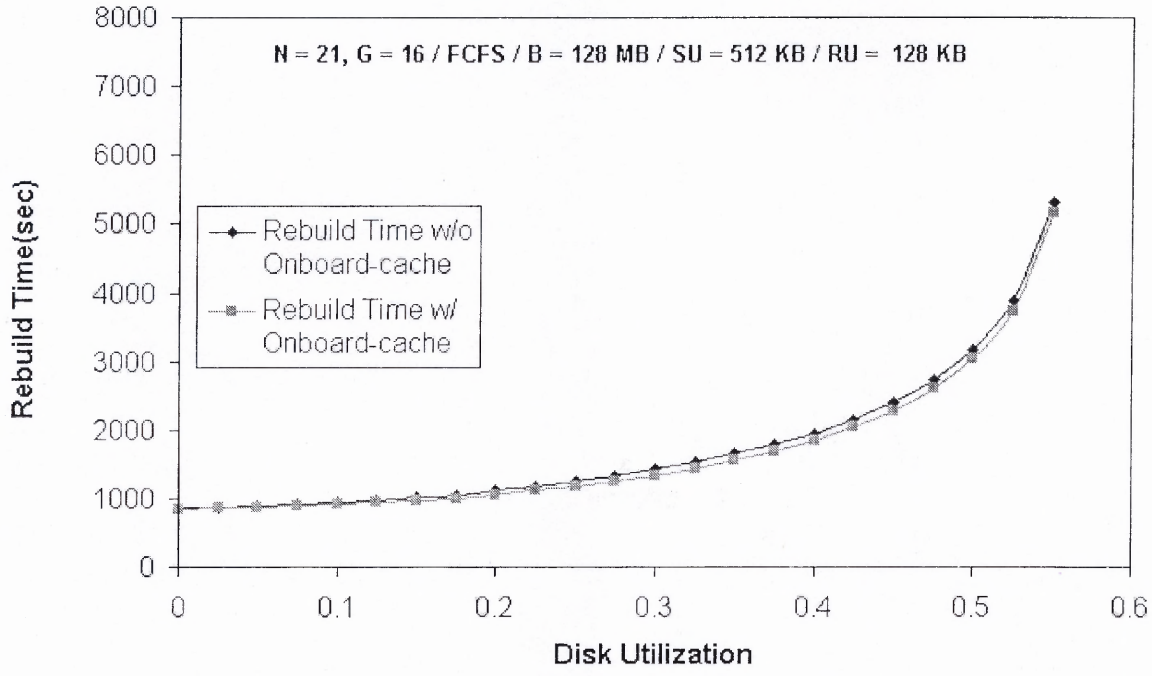

Figure 3.6 Effect of cache on rebuild time for rebuild unit size of 128 KB.

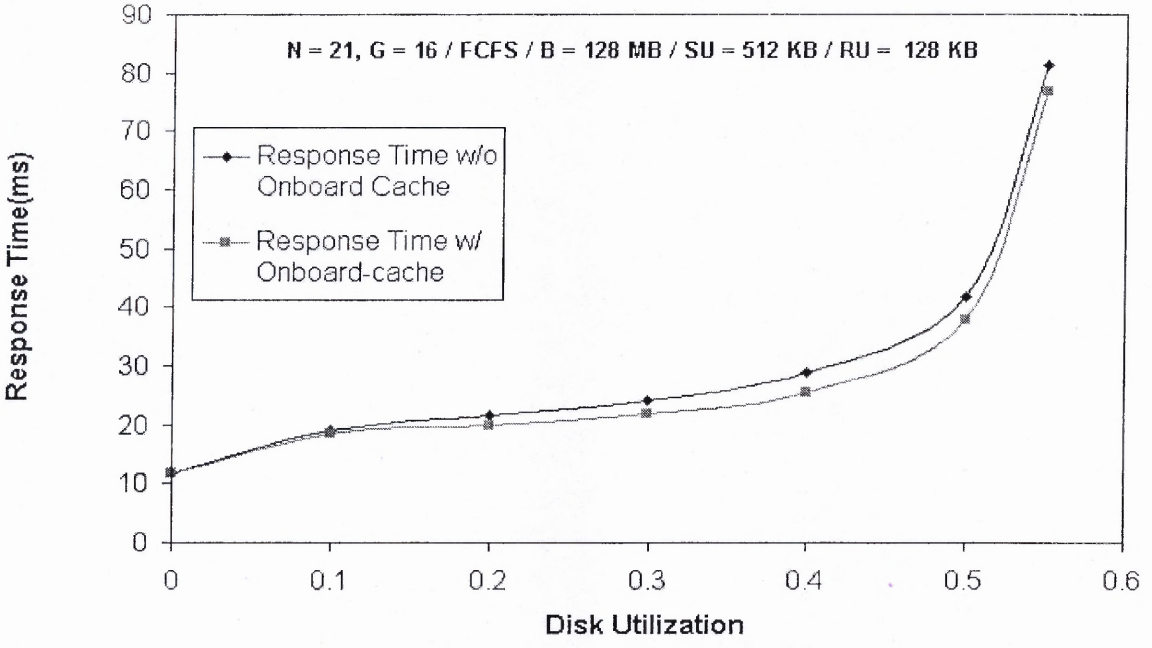

Figure 3.7 Effect of cache on user response time for rebuild unit size of 128 KB.

Table 3.2 Simulation Results with Rebuild Unit Size of 128 KB

U	t_{rb}	n_t	n_s	t_{rs}	T_{rb}	N_t	N_s	T_{rs}
0.000	857	1.28	1.11	11.89	855	1.28	1.10	11.85
0.025	877	40.48	39.48	17.34	869	38.93	38.10	17.10
0.050	892	41.36	39.52	18.01	877	39.77	38.38	17.57
0.075	913	42.28	39.61	18.58	891	40.65	38.41	17.94
0.100	940	43.16	39.66	19.20	911	41.49	38.50	18.37
0.125	975	44.47	40.14	19.78	939	42.76	38.71	18.75
0.150	1012	45.15	40.19	20.34	969	43.40	38.90	19.11
0.175	1058	47.23	41.24	20.93	1009	45.45	39.29	19.51
0.200	1114	48.34	41.52	21.53	1058	46.52	39.31	19.91
0.225	1179	50.47	42.82	22.12	1116	48.61	40.34	20.30
0.250	1246	52.41	43.92	22.76	1176	50.51	41.18	20.75
0.275	1327	53.96	44.64	23.47	1250	50.02	41.64	21.26
0.300	1423	56.78	46.63	24.24	1340	54.80	43.37	21.83
0.325	1532	60.17	49.19	25.11	1442	58.15	45.66	22.51
0.350	1649	62.88	51.07	26.15	1552	60.82	47.28	23.35
0.375	1789	67.73	55.09	27.33	1685	65.63	51.04	24.33
0.400	1951	72.39	58.92	28.81	1840	70.25	54.61	25.62
0.425	2150	78.16	63.86	30.65	2032	75.98	59.28	27.26
0.450	2402	86.48	71.34	33.01	2278	84.26	66.50	29.42
0.475	2729	95.92	79.95	36.44	2598	93.66	74.85	32.65
0.500	3181	110.10	93.31	41.74	3043	107.81	87.95	37.76
0.525	3883	131.40	113.70	51.44	3738	129.02	108.10	47.26
0.550	5312	174.50	156.00	81.26	5160	172.07	150.10	76.88

Lastly, Figures 3.8 and 3.9 show the effect on the rebuild and response times for a rebuild unit size of 256 KB.

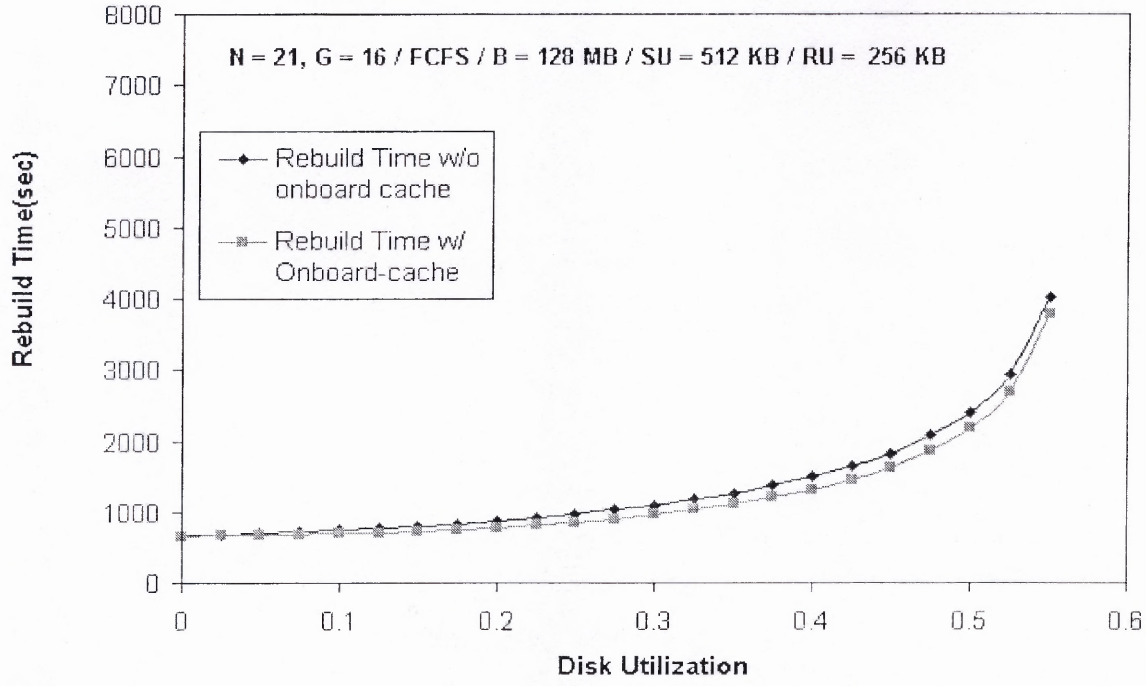

Figure 3.8 Effect of cache on rebuild time for rebuild unit size of 256 KB.

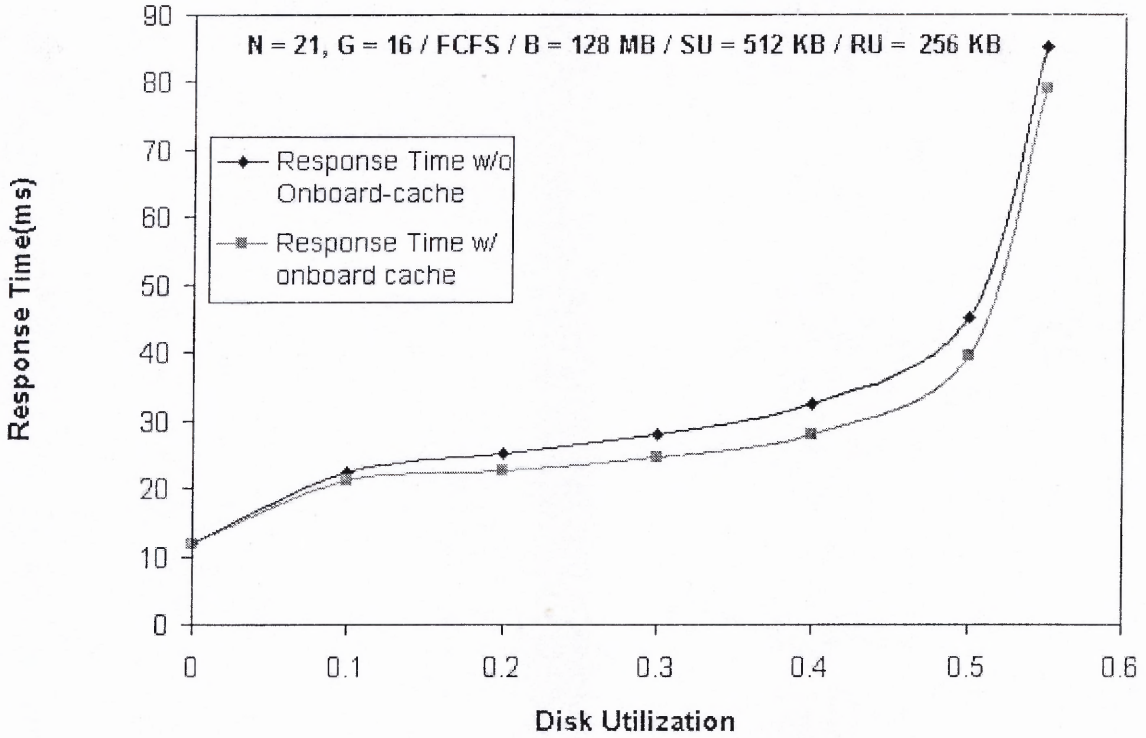

Figure 3.9 Effect of cache on user response time for rebuild unit size of 256 KB.

Table 3.3 Simulation Results with Rebuild Unit Size of 256 KB

U	t_{rb}	n_t	n_s	t_{rs}	T_{rb}	N_t	N_s	T_{rs}
0.00	667	1.19	1.07	11.89	664	1.19	1.07	11.83
0.025	687	29.96	28.8	20.14	673	27.91	26.16	19.79
0.050	703	31.15	29.02	20.86	678	29.06	26.22	20.23
0.075	722	32.27	29.18	21.62	686	30.14	26.43	20.70
0.100	743	33.39	29.34	22.37	695	31.22	26.64	21.17
0.125	767	33.49	29.48	23.02	708	31.28	26.75	21.53
0.150	800	34.52	29.71	23.66	730	32.27	26.81	21.89
0.175	832	36.18	29.83	24.37	751	33.90	26.87	22.31
0.200	875	37.37	29.87	25.01	782	35.05	26.94	22.67
0.225	921	38.25	29.93	25.67	817	35.89	26.99	23.05
0.250	974	40.18	30.35	26.41	859	37.78	27.03	23.50
0.275	1032	41.44	30.65	27.14	906	39.00	27.15	23.95
0.300	1103	42.78	31.03	27.90	966	40.30	27.22	24.42
0.325	1180	46.31	33.60	28.82	1031	43.79	29.49	25.06
0.350	1272	48.42	34.74	29.81	1112	45.86	30.32	25.76
0.375	1376	51.93	37.29	30.98	1205	49.33	32.56	26.65
0.400	1501	54.89	39.29	32.40	1391	52.25	34.26	27.79
0.425	1646	59.74	43.18	34.25	1452	57.06	37.84	29.35
0.450	1827	65.01	47.48	36.66	1622	62.29	41.83	31.48
0.475	2073	73.26	54.77	39.84	1857	70.50	48.82	34.37
0.500	2409	83.39	63.94	45.16	2182	80.59	57.68	39.41
0.525	2925	99.14	78.73	54.90	2686	96.30	72.17	48.86
0.550	4027	132.07	110.70	85.28	3777	129.19	103.82	78.96

In the following figures, the effect on rebuild performance and user response times is summarized. As can be seen from these figures, the effect of enabling the onboard-cache increases with the size of the rebuild unit. Figure 2.7 shows that as the

rebuild unit size increases, rebuild time improves at the expense of user response time. This is reflected in Figure 3.10. Also, the difference in rebuild performance increases with the rebuild unit (RU) size. The reason for this lies in probability theory. The probability of a user request finding the requested track in the cache is higher for larger rebuild unit size than for a smaller one. This is due to the fact that user requests are processed among rebuild requests and there is no preemption of either of them by the other. So the likelihood of a user request missing the requested track in the rebuild unit is high when the rebuild unit size is small. Therefore the percentage of hits to the cache will increase with the rebuild unit size, resulting in an increase in rebuild performance improvement.

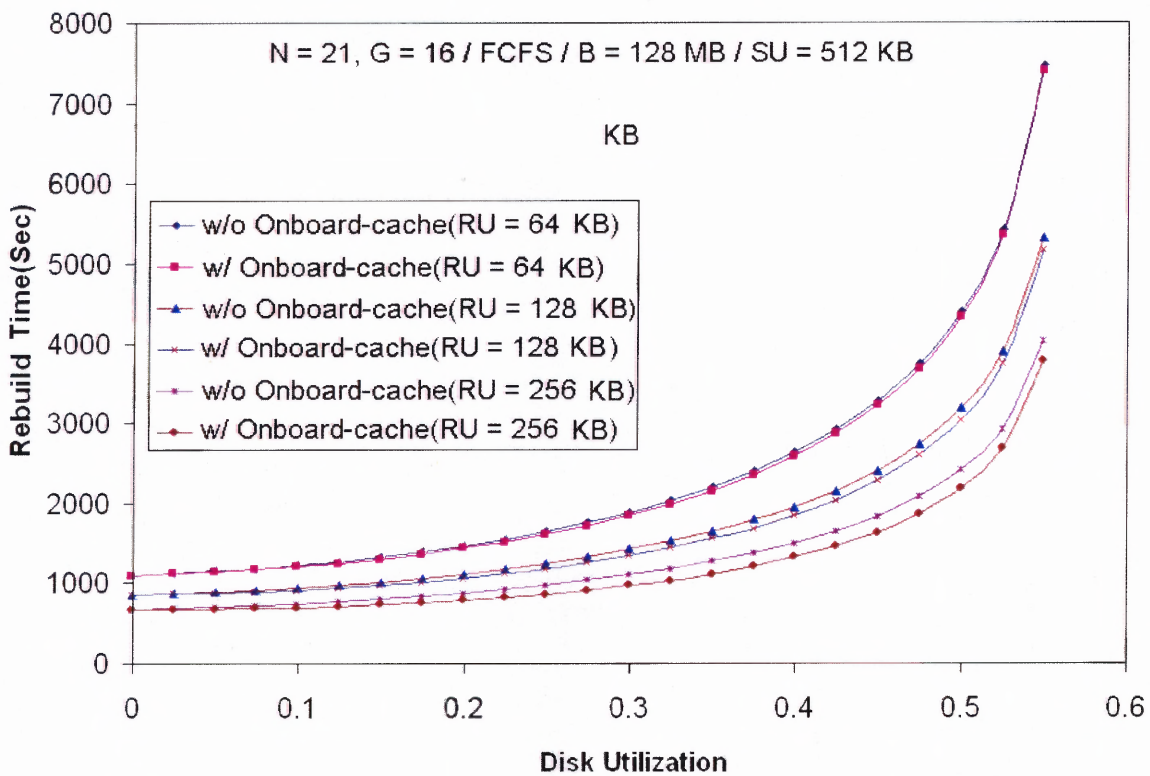

Figure 3.10 Effect of rebuild unit size on rebuild performance improvement using onboard cache.

A direct result of using onboard cache is an improvement in user response time as seen in the previous figures. Hence,the improvement in user response time also increases with the rebuild unit size as seen in Figure 3.11.

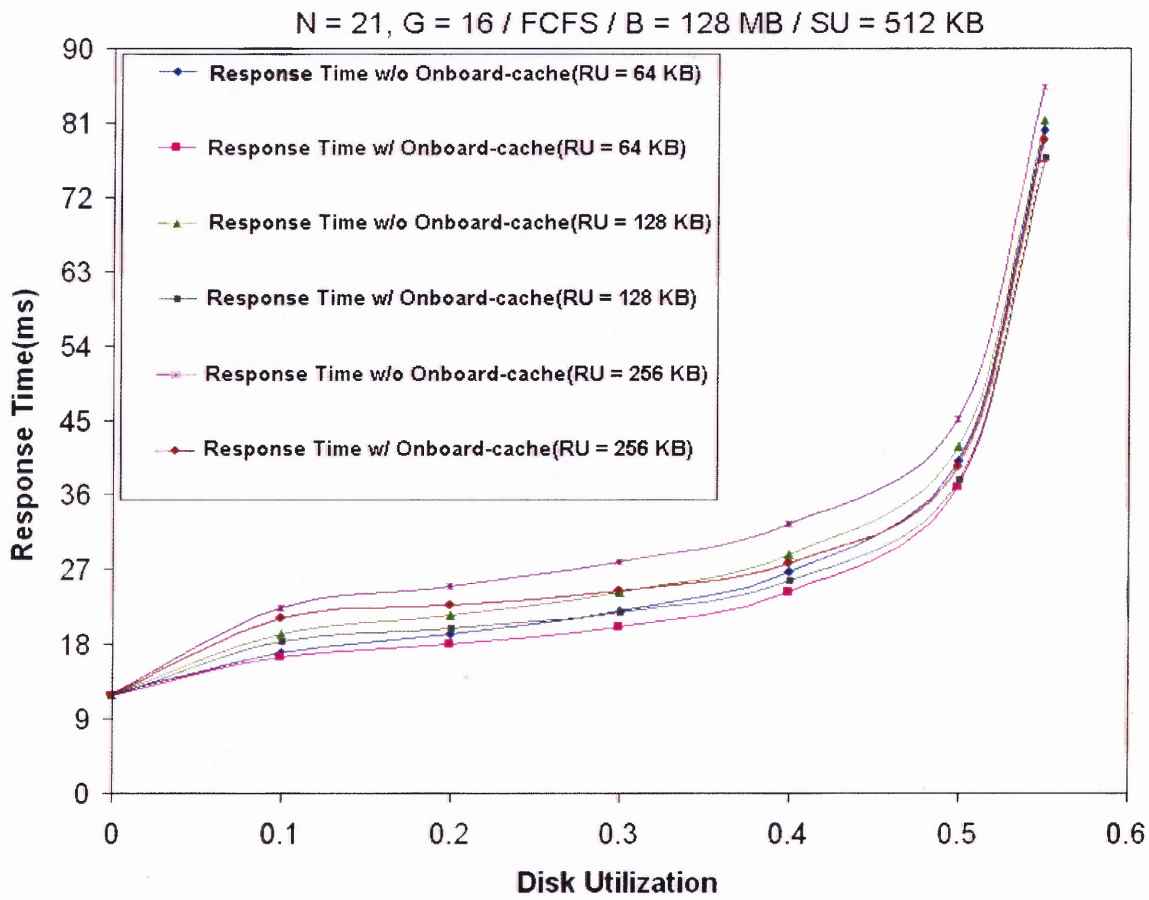

Figure 3.11 Effect of rebuild unit size on user response time improvement using the cache.

Improvements in rebuild and response times go hand in hand. An improvement in user response time results in an improvement in rebuild performance since the rebuild operation can be resumed quickly. The next chapter discusses another way of improving rebuild performance by delaying the termination of vacations.

CHAPTER 4

DELAYING VACATION TERMINATION

4.1 Enhancing VSM Processing

In the traditional VSM (Vacationing Server Model) processing, the rebuild requests are considered to be of lower priority than user requests. But there is no preemption of rebuild requests due to user requests. Rebuild requests are processed only when the disk is idle (i.e. in Vacation). When a user request arrives, it has to wait until the current rebuild unit is being rebuilt. This is depicted in Figure 4.1.

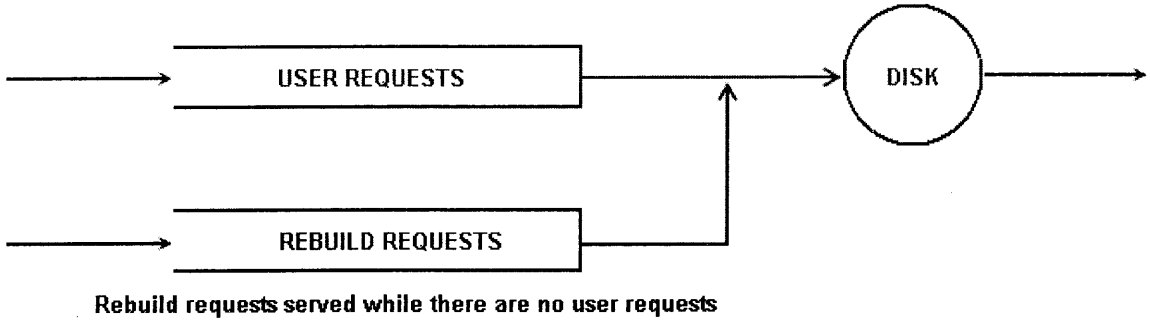

Figure 4.1 Rebuilding in VSM Mode.

The rebuild performance in VSM can be improved by waiting for example two user requests before serving them, thus delaying the termination of the disk vacations. This improvement is at the expense of the user response time. But as the simulation results in the next section show, the increase in user response time is negligible, and only occurs at lower utilizations. The configuration for the simulation is exactly the same as in section 3.4 except that the parameter for the VSM processing is set to two so that the disks are allowed to be idle until there are two user requests.

4.2 Effect on Rebuild Performance

The effect on rebuild performance is discussed for rebuild unit sizes of 64, 128 and 256 KB. As seen in Figure 4.2, the improvement in rebuild performance increases as the rebuild unit size is decreased. The reason for this lies in the number of rebuild units that can be processed before two user requests are processed. For the same user request arrival rate, more number of rebuild units (RUs) can be processed when the RU size is small than when it is large. In summary, the greater the inter-arrival time of user requests compared to rebuild unit processing time, the greater the rebuild performance improvement.

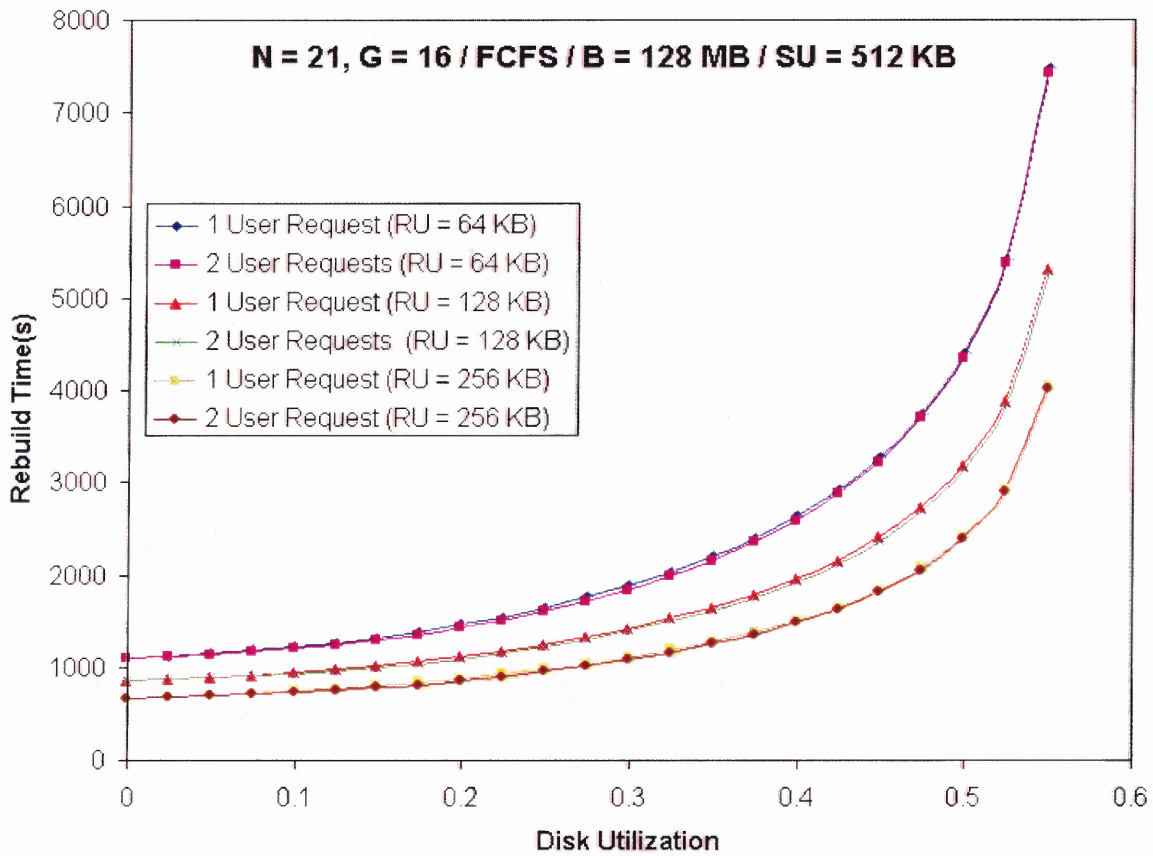

Figure 4.2 Effect of rebuild unit size on rebuild performance improvement

Figure 4.3 shows the effect of batch processing user requests when the rebuild unit size is 256 KB. The effect the similar for lower rebuild unit sizes as discussed later.

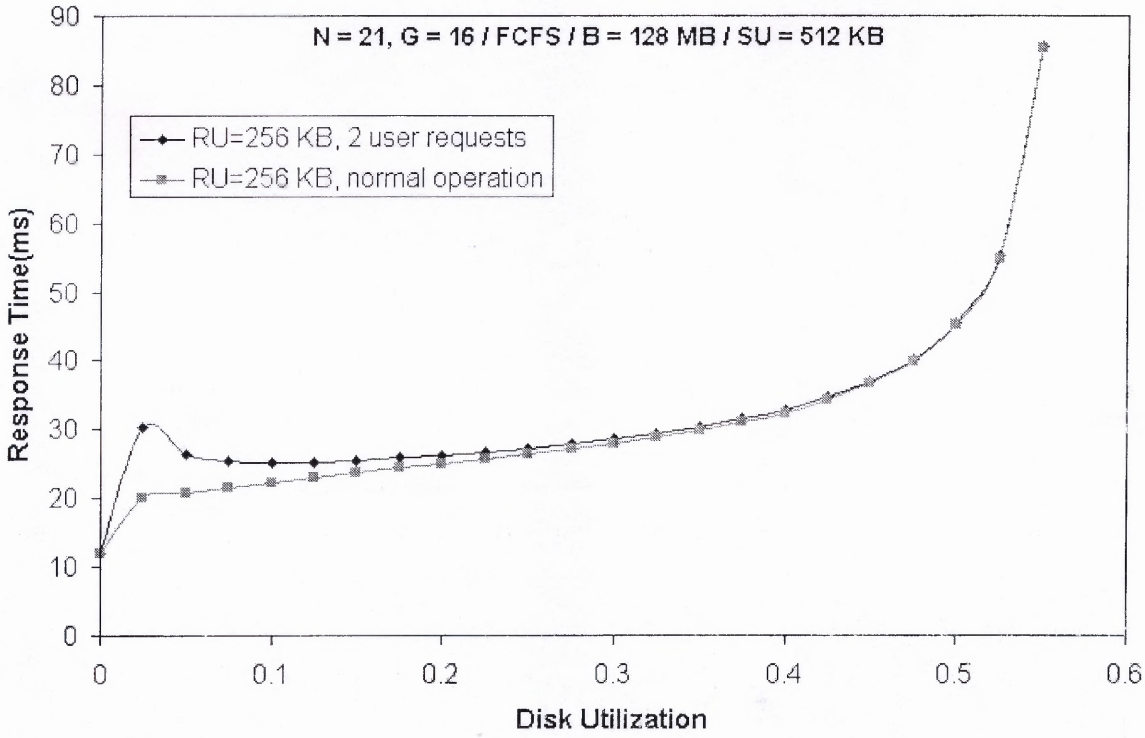

Figure 4.3 Effect of delayed vacation termination on the response time for rebuild unit size of 256 KB.

As seen in the Figure 4.3, there is an increase in the response time at lower utilizations since the arrival rate of user requests will be very small (i.e. the inter-arrival time of user requests will be high). So a user request will have to wait longer before it can be processed since there have to be two user requests before their processing can begin. As the utilization increases, the difference diminishes. Since the utilization will be usually high during rebuild, this negative impact on user response time is not perceptible. Figure 4.4 shows the relative impact on user response times for the three rebuild unit sizes.

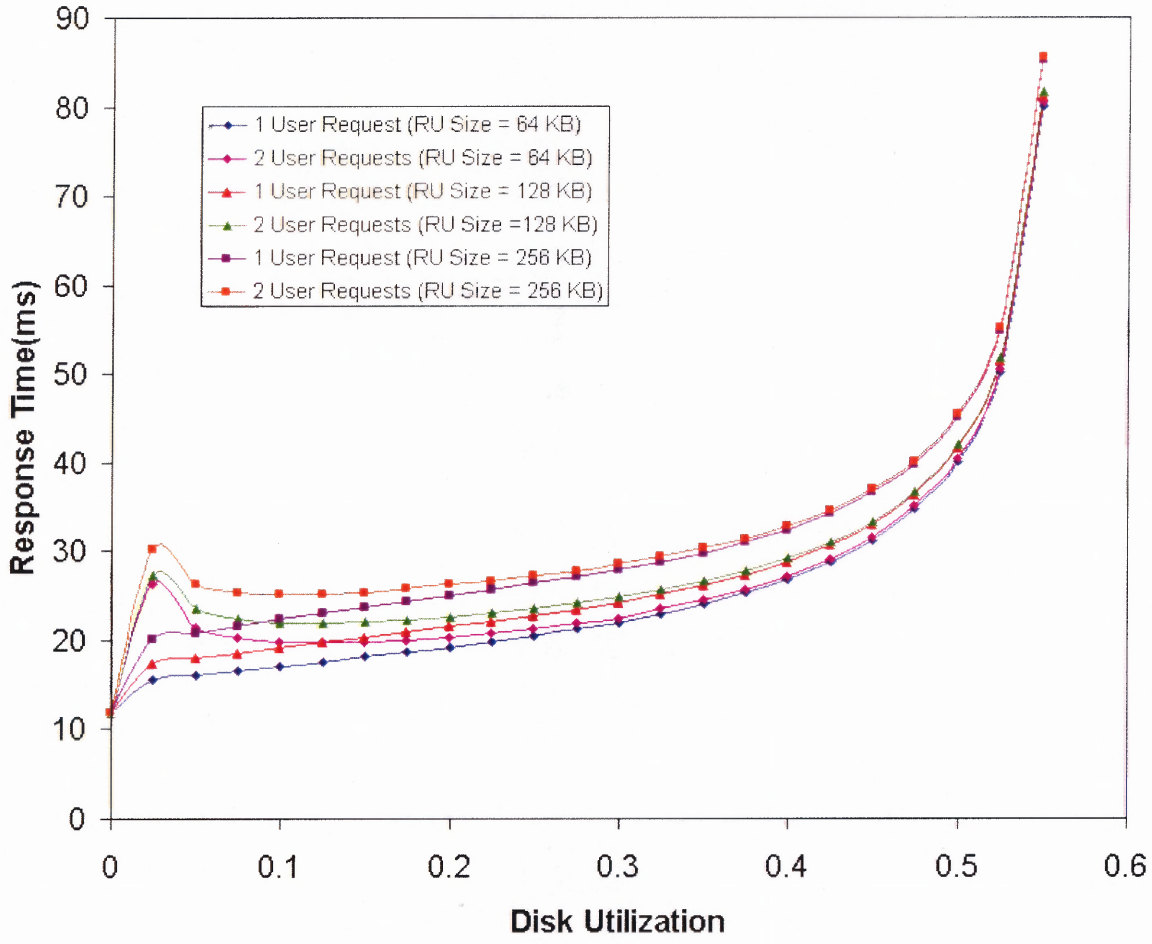

Figure 4.4 Relative effect of delayed vacation termination on user response time.

The conclusion that can be drawn from this study is that the rebuild unit size does not have a major impact on the effect the delaying of vacation termination has on response time, but the improvement in rebuild performance is higher for smaller rebuild unit sizes.

The next chapter discusses the effects a small rebuild buffer would have on the rebuild performance.

CHAPTER 5

REBUILDING WITH A LIMITED BUFFER

5.1 Effect of Rebuild Buffer on Rebuild and Response Time

The size of the rebuild buffer has an effect on rebuild performance and user response time as shown in Figure 5.1.[13] The parameters for the simulation are the same as those discussed in chapter 3. The rebuild unit size is 64 blocks (256 KB), and the disk utilization is 0.9. A shared buffer is used for the simulation. After a rebuild unit from the surviving disks is read, it is XORed onto the rebuild working buffer and will be written to the spare disk later. When all the surviving rebuild units from parity group are read, the buffer contains the reconstructed data which can be written to the spare disk.

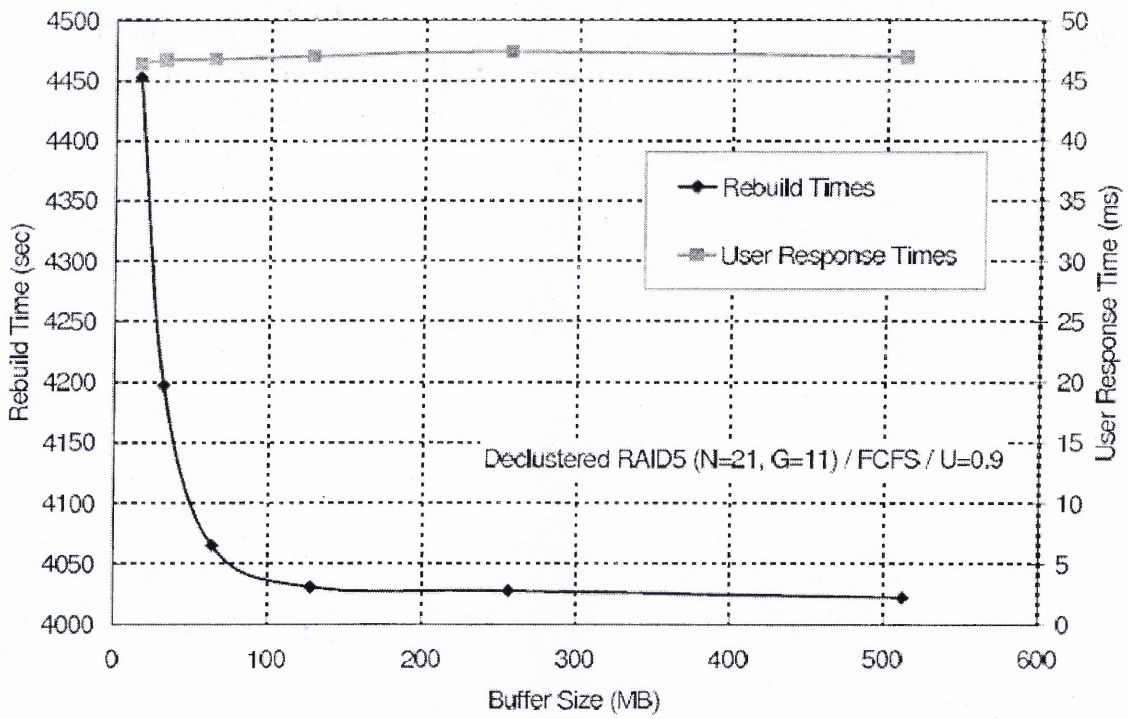

Figure 5.1 Impact of buffer size on rebuild and response times.

5.2 Rebuild Time Analysis with a Limited Buffer

Each disk has dedicated buffers that are used before each rebuild unit is XORed to the rebuild buffer. These buffers are very small and can be neglected. The user response time is not considered here since it is not affected significantly. There is a temporary load imbalance, as a result of which the rebuild process is suspended.[13] A larger buffer reduces the possibility of this suspension. The improvement in rebuild times is significant for smaller sizes of the buffer, and diminishes as the size gets larger. It can be easily seen that there is no improvement in rebuild time beyond a buffer size of 128 MB. Further, Figure 2.7 shows that the rebuild performance does not improve significantly after a rebuild unit size of 256 KB. Figure 5.2 shows the effect of buffer size on rebuild time for a utilization of 0.3. The declustering ratio is taken as one since this eliminates the delay due to the spare disk.

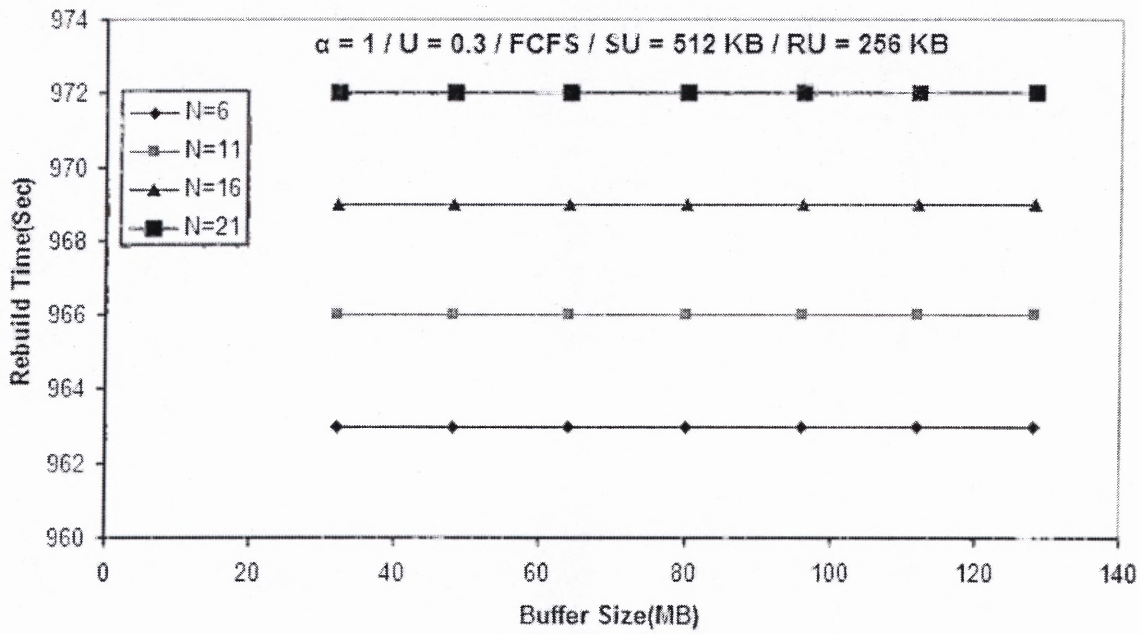

Figure 5.2 Effect of buffer size on rebuild time for disk utilization of 0.3.

Figures 5.3 and 5.4 show the effect on rebuild time for utilizations of 0.6 and 0.9.

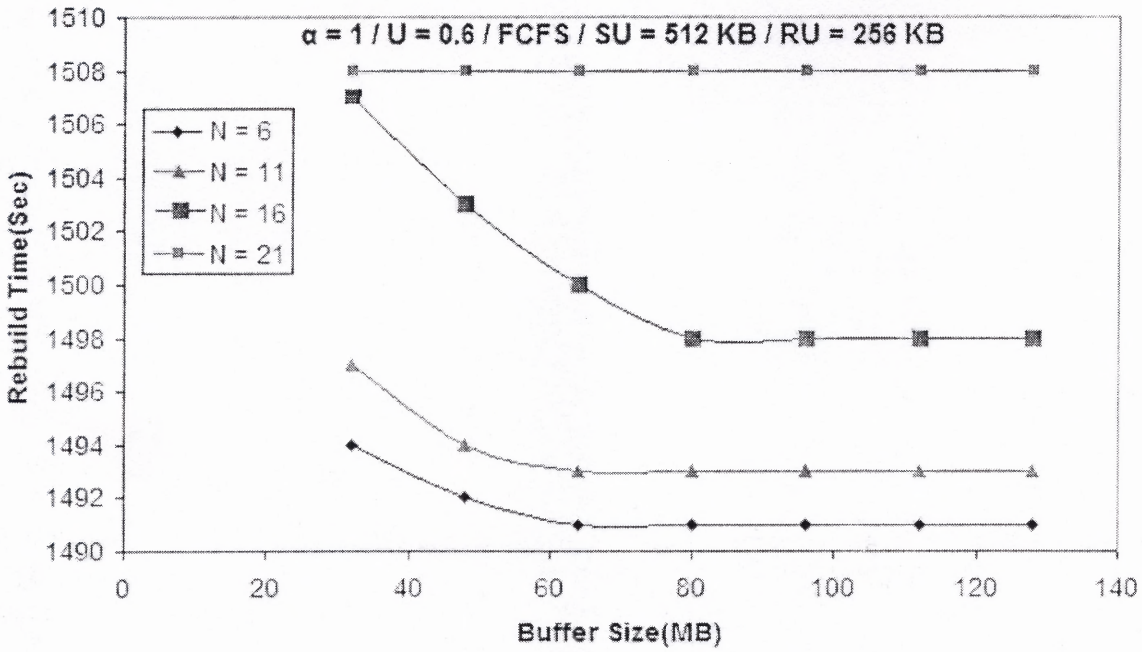

Figure 5.3 Effect of buffer size on rebuild time for disk utilization of 0.6.

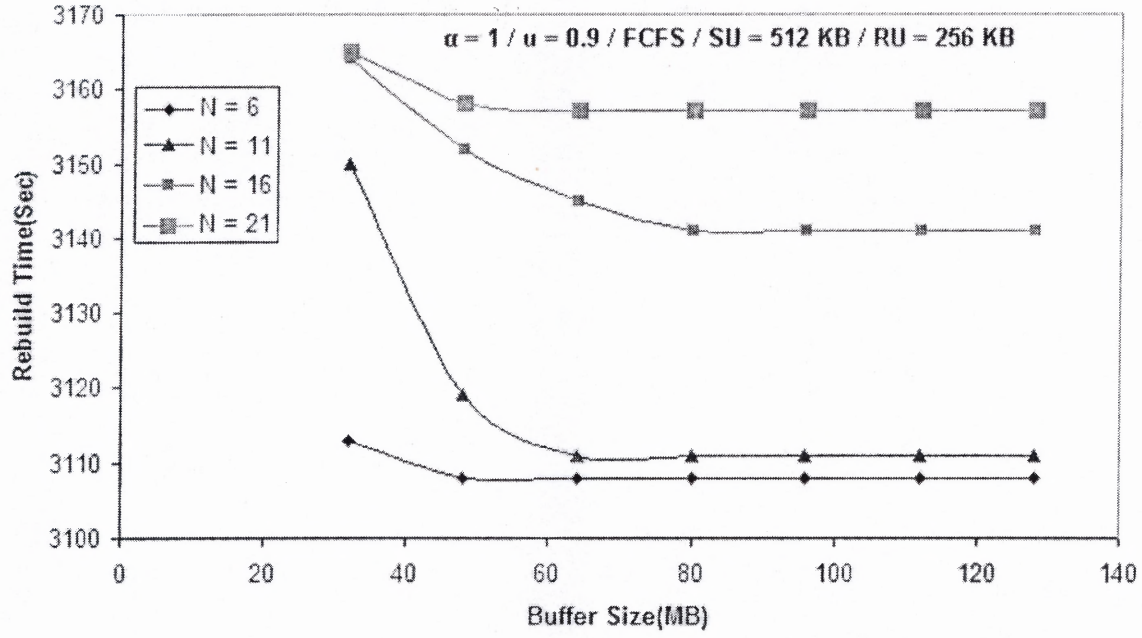

Figure 5.4 Effect of buffer size on rebuild time for disk utilization of 0.9.

It can be inferred from the previous three figures that the rebuild times are constant until the buffer size is 80 MB, and start increasing when it is reduced. The key transition points here are sizes 80, 64 and 48 MB where the rebuild time increases in most cases. Another key observation that can be made from these figures is that the effect of limiting the buffer increases with the disk utilization. When the utilization is low (e.g. 0.3), there is no effect of reducing the buffer until a size of 32 MB, but as Figure 5.5 reveals, the rebuild time increases when the buffer is reduced to 16 MB whereas in Figures 5.3 and 5.4, it increases when the buffer is reduced to 64 MB. This implies that buffer usage increases with disk utilization.

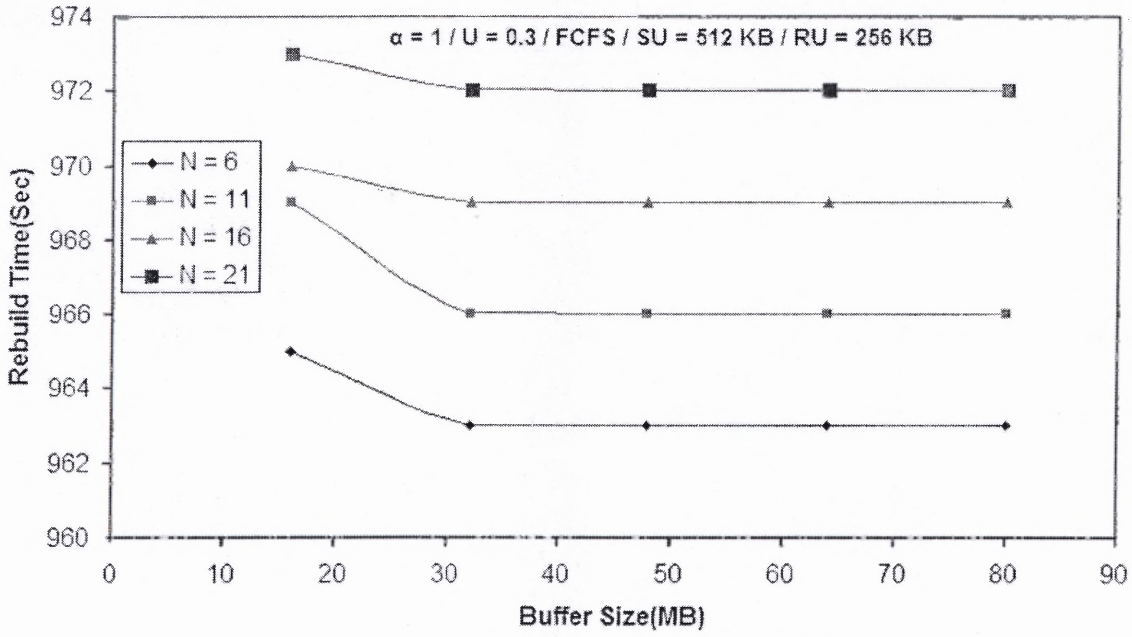

Figure 5.5 Effect of disk utilization on rebuild buffer usage.

Until now a declustering ratio (α) of one was used. Figure 5.6 shows the effect α has on the rebuild time for 21 disks. In this figure, G = 6, 11 and 21 imply declustering ratios of 0.25, 0.5 and 1.0,respectively.

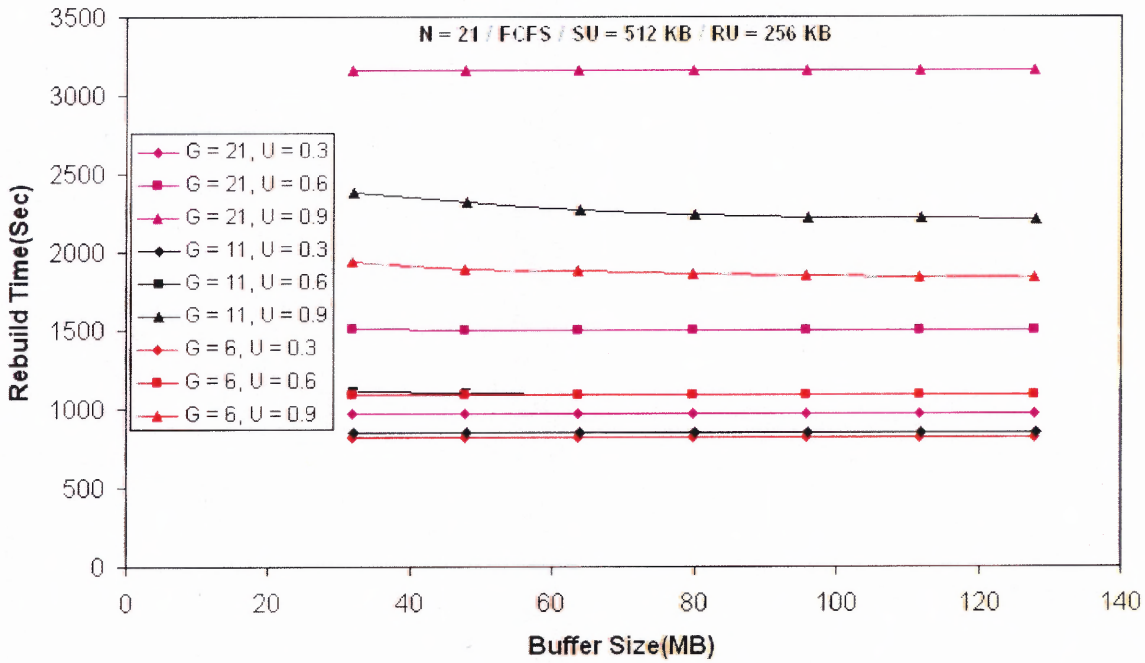

Figure 5.6 Effect of declustering ratio on rebuild time.

It is clearly evident from Figure 5.6 that for the same utilization, a lower declustering ratio gives a shorter rebuild time. Also, the difference in rebuild times is more visible at a higher utilization. The effect of disk utilization on the rebuild times is shown in Figure 5.7. The rebuild time increases with disk utilization at different rates for different declustering ratios. The rate increases with the declustering ratio. Here buffer sizes of 48, 64 and 80 MB are used for reasons stated earlier. Also, the number of disks used is 21 and three different declustering ratios are considered.

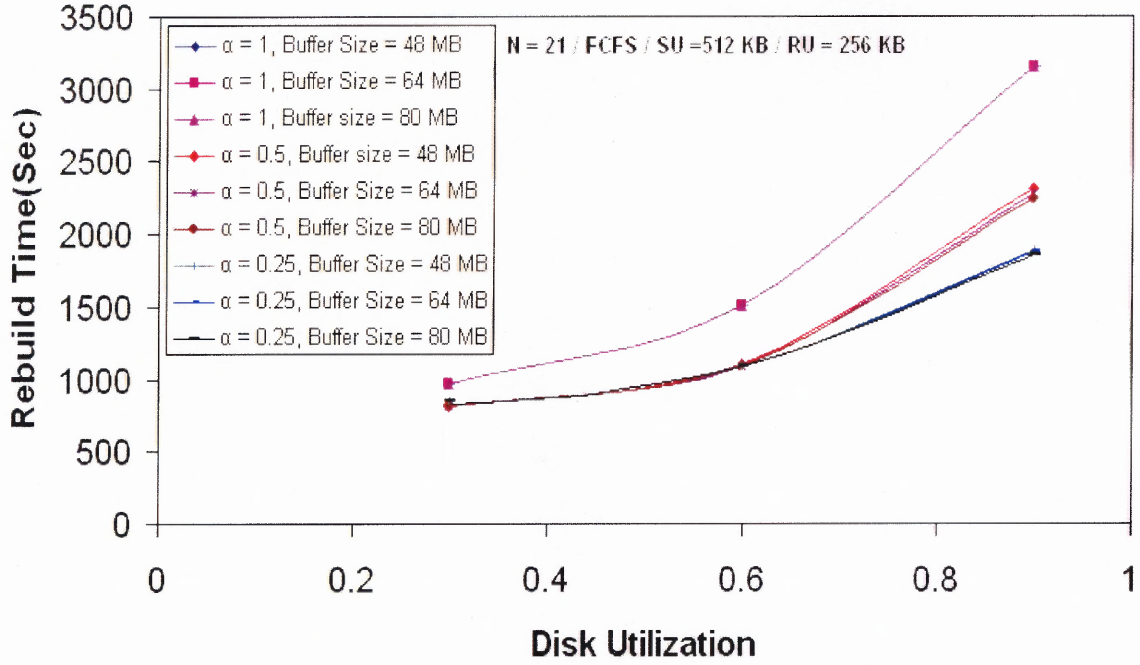

Figure 5.7 Effect of disk utilization on rebuild time.

Also of interest is the impact the number of disks (α = 1) has on the rebuild time of the system. Figure 5.8 shows this for utilization of 0.3.

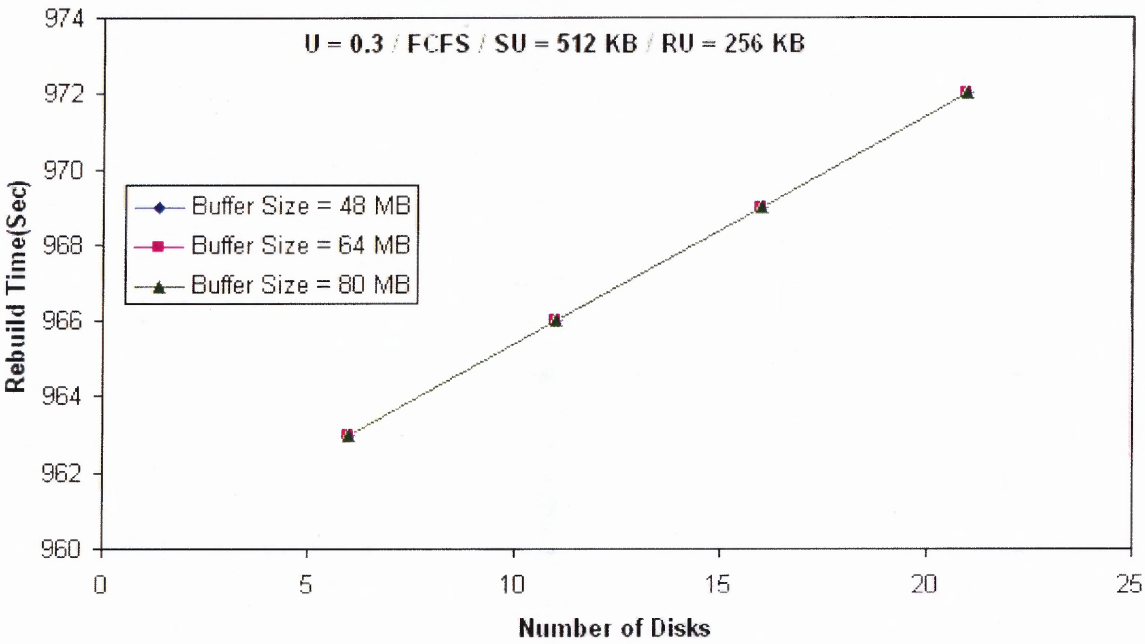

Figure 5.8 Impact of number of disks on the rebuild time, for utilization of 0.3.

Figures 5.9 and 5.10 show the same for utilizations of 0.6 and 0.9.

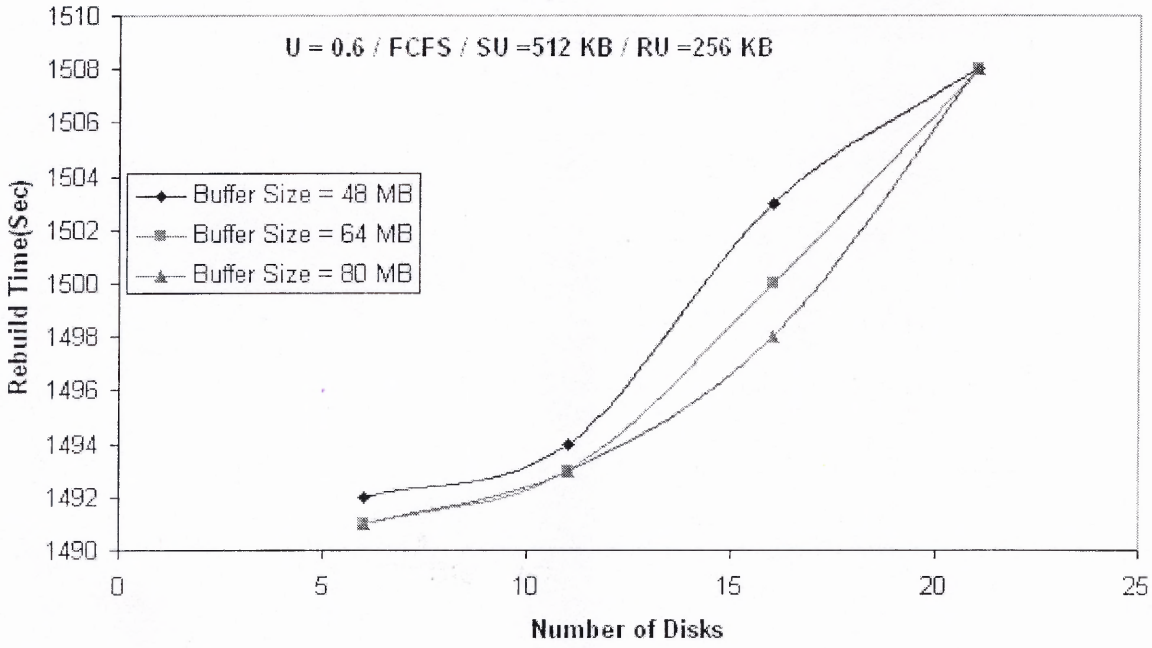

Figure 5.9 Impact of number of disks on rebuild time, for utilization of 0.6.

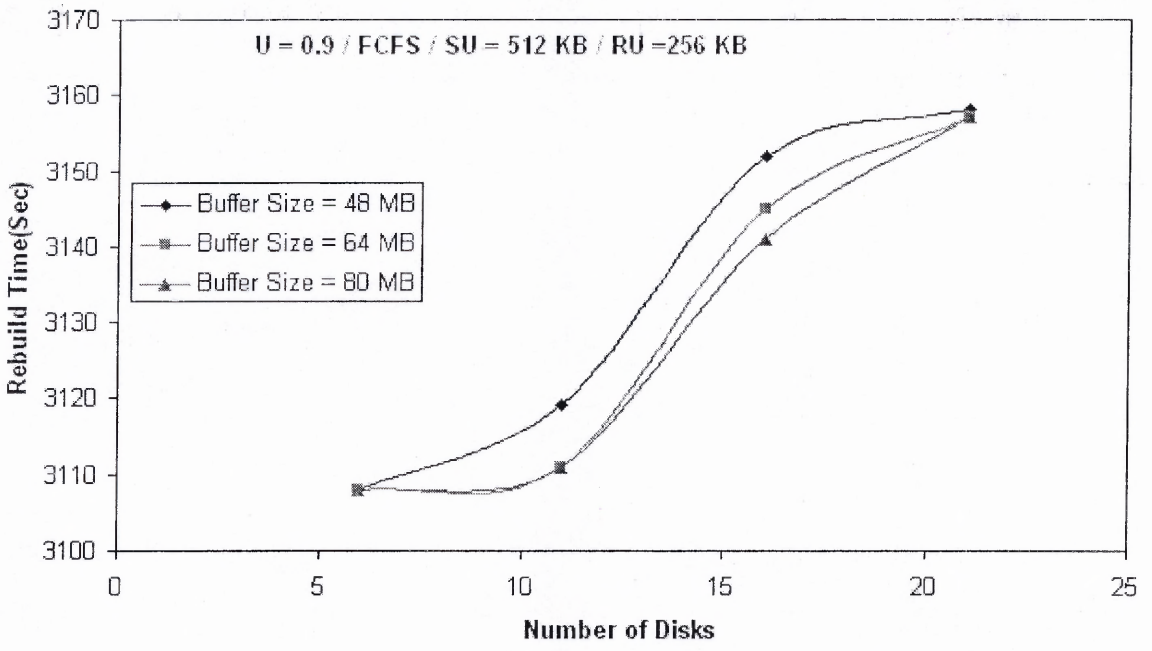

Figure 5.10 Impact of number of disks on rebuild time, for utilization of 0.9.

It can be observed that for a utilization of 0.3, the increase in the rebuild time is linear. Also, there is no difference in the rebuild time for the three different sizes, implying that the buffer is not utilized fully at this ulitization. In Figures 5.9 and 5.10, the increase in rebuild time takes a different pattern for different sizes of the buffer and between different numbers of disks, the reason being that the utilization of the rebuild buffer depends not only on its size but also on the number of disks being used.

The rebuild process is stalled when the buffer reaches 100% utilization and resumes when utilization goes down. An analysis of the number of rebuild stalls shown in Figure 5.11 justifies the increase in rebuild time when the buffer size is decreased.

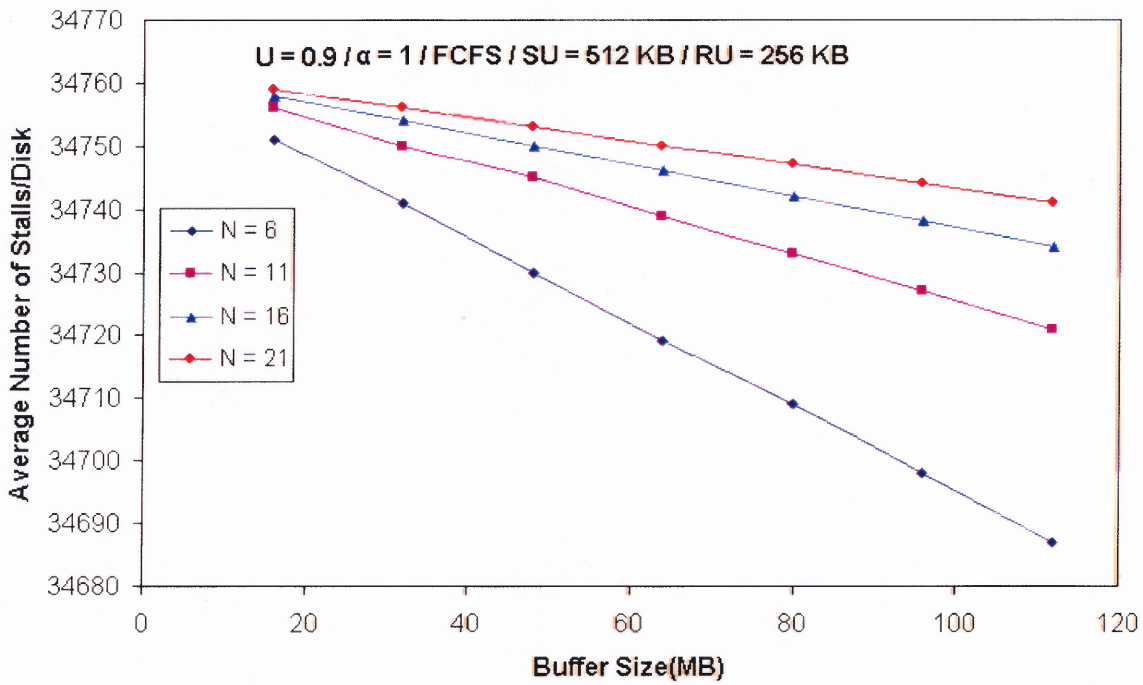

Figure 5.11 Effect of buffer size on disk rebuild stalls.

It is evident that with fewer disks, the number of rebuild stalls increases at a higher rate, and as Figures 5.12 and 5.13 show, the mean number of stalls per disk decrease as the number of disks increases and as the utilization decreases.

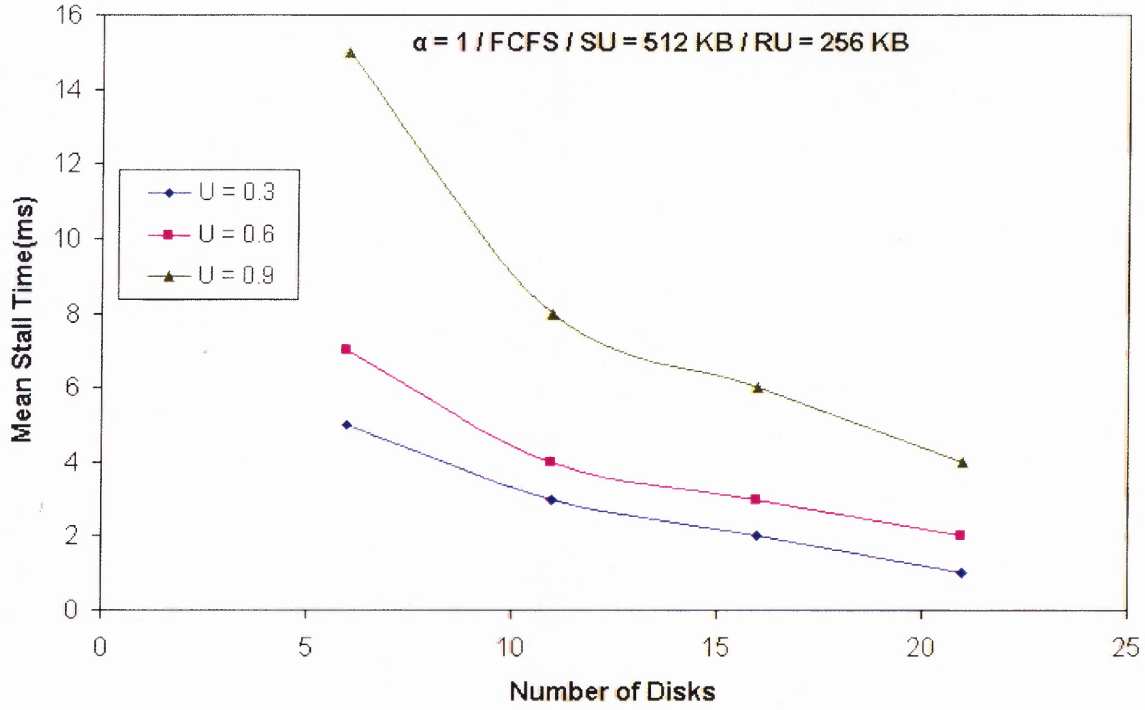

Figure 5.12 Variation of mean stall time with number of disks with a 48 MB buffer.

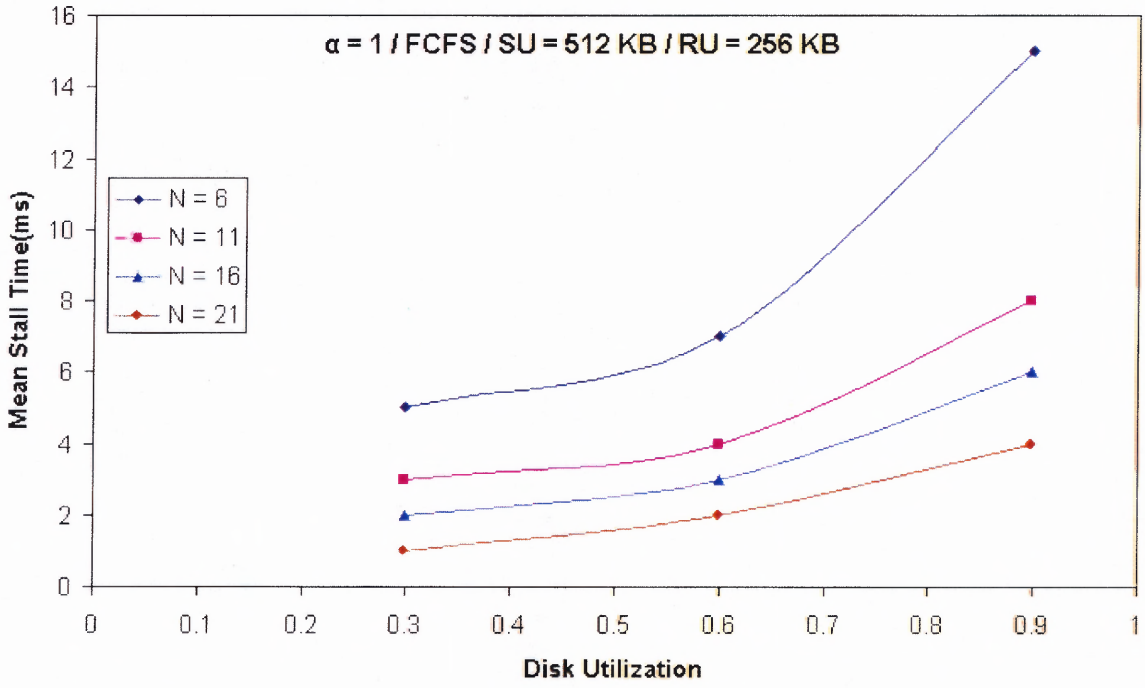

Figure 5.13 Variation of mean stall time with disk utilization with a 48 MB buffer.

Finally, Table 5.1 shows the relationship between the mean stall time, average number of stalls per disk and the rebuild time, for buffer sizes of 128, 80, 64 and 48 MB. A size of 128 MB is considered close to an infinite buffer. The mean stall time varies very insignificantly, as a result of which it appears constant in the table.

Table 5.1 Relationship Between Mean Stall Time, Average Number of Stalls and Rebuild Time

Buffer Size(MB)	Rebuild Time(ms)	Mean number of stalls per disk	Mean stall time(ms)
128	1843389	-	-
80	1862974	9917	9
64	1876994	9920	9
48	1893788	9923	9

In the above table, for any buffer size, the sum of the rebuild time with infinite buffer (i.e. 128 MB) and the product of the mean stall time and mean number of stalls per disk is close to the rebuild time. The difference is about 4%, and is attributed to the lack of synchronization of the disks.

CHAPTER 6

CONCLUSION

An improvement in rebuild performance and user response time resulting from the inclusion of an onboard cache was shown using the DASim toolkit. The architectural features of the cache were presented and its operation was described followed by a performance analysis done using the toolkit. The improvement in rebuild time increased with the rebuild unit size.

A novel technique to enhance the rebuild performance was proposed that delays the termination of the vacationing disks until there are two user requests waiting to be processed. This was shown to improve the rebuild times without significantly affecting the user response time. The rebuild time improved significantly when a smaller rebuild unit was used, and the improvement in user response time was unaffected by the rebuild unit size.

Finally, the consequences of having a limited rebuild buffer were presented. Most of the previous work in CRAID5 had been done assuming an infinite rebuild buffer (practically, a very large size), as a result of which its impact on rebuild performance was not known. Using a thorough analysis, it was proved that the rebuild performance was affected when the buffer size went below a cut-off value which itself depended on the disk utilization, the number of disks being used and the declustering ratio.

APPENDIX A

RELIABILITY MODELING

Given the Mean Time To Failure (MTTF) and a target Mean Time To Data Loss (MTTDL), The MTTDL of a disk array that can tolerate one failure is expressed as:

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

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).

APPENDIX B

SETTING UP DASIM FOR CRAID5 SIMULATION

The CRAID5 module in DASIM uses command line arguments to set up the simulation environment. The normal and degraded mode is simulated in the file CRAID5Sim.h, and the rebuild mode in CRAID5Sim2.h. The file CRAID5Frame.h contains definitions for the CRAID5 controller, which use definitions of the single disk controller, defined in SDFrame.h. Each simulation generates 4 files in a directory whose name should be specified at the command line. The most important file is Rebuild.txt, which gives the rebuild and mean response times. The following Parameters specified on the command line in the order: buffer size, rebuild unit size, read redirection, model, model parameters, rebuild type, dynamic control, request arrival rate, results directory, name of the disk, scheduling policy, number of priority classes, threshold, number of disks, group size, failed disk, stripe unit size, read/write ratio, number of requests processed before rebuild, number of requests processed after rebuild, and the cache. A batch file containing the required simulation command lines can be used. This eliminates the recompilation step that was required in the earlier versions of DASim.

REFERENCES

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

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

3. Holland, M., Zelenka J et al. (1996). RAIDframe: A Rapid Prototyping Tool for RAID Systems. Parallel Data Laboratory, Carnegie Mellon University.

4. 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.

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

6. 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.

7. 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.

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. 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.

11. Ng S. W. and Mattson R. L. (1994). *Uniform parity distribution in disk arrays with multiple failures.* IEEE Trans. Computers 43(4): 501-506.

12. Merchant, A. and Yu, P. S. (1996). *Analytic modeling of clustered RAID with mapping based on nearly random permutation.* IEEE Trans. Computers 45(3): 367-373.

13. Fu, G., Thomasian, A., Han, C., and Ng, S. W. (2004). Rebuild strategies for clustered redundant disk arrays. SPECTS04, 598-607.

14. Fu, G., Thomasian, A., Han, C. and Ng, S. W. (2004). *Rebuild strategies for redundant disk arrays*. Proceedings of the IEEE/NASA Conference on Mass Storage Systems and Technologies, 3 – 19.

15. Thomasian, A. (1995). Rebuild options in RAID5 disk arrays. Proceedings of the 7th IEEE Symposium on Parallel and Distributed Systems, 511-518.

16. http://www.pdl.cmu.edu/DiskSim/diskspecs.html. (Retrieved Nov. 2004).

17. Ramakrishnan, K. K., Biswas, P., and Karedla, R. (1992). Analysis of I/O traces in commercial computing environments. Proceedings of the Joint ACM SIGMETRICS/Performance '92 Conf, 78 - 90.

18. Thomasian, A., Han, C., Fu, G. and Liu, C. (2004). A performance tool for RAID disk arrays. Quantitative Evaluation of Systems (QEST), 6 – 7.