Scalability of RAID Systems

Yan Li

Doctor of Philosophy

Institute of Computing Systems Architecture

School of Informatics

University of Edinburgh

2010

Abstract

RAID systems (Redundant Arrays of Inexpensive Disks) have dominated back-end storage systems for more than two decades and have grown continuously in size and complexity. Currently they face unprecedented challenges from data intensive applications such as image processing, transaction processing and data warehousing. As the size of RAID systems increases, designers are faced with both performance and reliability challenges. These challenges include limited back-end network bandwidth, physical interconnect failures, correlated disk failures and long disk reconstruction time.

This thesis studies the scalability of RAID systems in terms of both performance and reliability through simulation, using a discrete event driven simulator for RAID systems (SIMRAID) developed as part of this project. SIMRAID incorporates two benchmark workload generators, based on the SPC-1 and Iometer benchmark specifications. Each component of SIMRAID is highly parameterised, enabling it to explore a large design space. To improve the simulation speed, SIMRAID develops a set of abstraction techniques to extract the behaviour of the interconnection protocol without losing accuracy. Finally, to meet the technology trend toward heterogeneous storage architectures, SIMRAID develops a framework that allows easy modelling of different types of device and interconnection technique.

Simulation experiments were first carried out on performance aspects of scalability. They were designed to answer two questions: (1) given a number of disks, which factors affect back-end network bandwidth requirements; (2) given an interconnection network, how many disks can be connected to the system. The results show that the bandwidth requirement per disk is primarily determined by workload features and stripe unit size (a smaller stripe unit size has better scalability than a larger one), with cache size and RAID algorithm having very little effect on this value. The maximum number of disks is limited, as would be expected, by the back-end network bandwidth.

Studies of reliability have led to three proposals to improve the reliability and scalability of RAID systems. Firstly, a novel data layout called PCDSDF is proposed. PCDSDF combines the advantages of orthogonal data layouts and parity declustering data layouts, so that it can not only survive multiple disk failures caused by physical interconnect failures or correlated disk failures, but also has a good degraded and rebuild performance. The generating process of PCDSDF is deterministic and time-efficient. The number of stripes per rotation (namely the number of stripes to achieve rebuild

workload balance) is small. Analysis shows that the PCDSDF data layout can significantly improve the system reliability. Simulations performed on SIMRAID confirm the good performance of PCDSDF, which is comparable to other parity declustering data layouts, such as RELPR.

Secondly, a system architecture and rebuilding mechanism have been designed, aimed at fast disk reconstruction. This architecture is based on parity declustering data layouts and a disk-oriented reconstruction algorithm. It uses stripe groups instead of stripes as the basic distribution unit so that it can make use of the sequential nature of the rebuilding workload. The design space of system factors such as parity declustering ratio, chunk size, private buffer size of surviving disks and free buffer size are explored to provide guidelines for storage system design.

Thirdly, an efficient distributed hot spare allocation and assignment algorithm for general parity declustering data layouts has been developed. This algorithm avoids conflict problems in the process of assigning distributed spare space for the units on the failed disk. Simulation results show that it effectively solves the write bottleneck problem and, at the same time, there is only a small increase in the average response time to user requests.

Publication List:

- Y. Li, T. Courtney, R. N. Ibbett and N.Topham, "On the Scalability of the Back-end Network of Storage Sub-Systems", in *SPECTS 2008*, pp 464-471, Edinburgh UK.

- Y.Li and R.N. Ibbett, "SimRAID-An Efficent Performance Evaluation Tool for Modern RAID Systems", in *Poster Competition of University of Edinbrugh Informatics Jamboree*, Edinburgh, 2007. the First prize

- Y. Li, T. Courtney, R. N. Ibbett and N.Topham, "Work in Progress: On The Scalability of Storage Sub-System Back-end Network", in *WiP of FAST*, San Jose, USA, 2007.

- Y. Li, T. Courtney, R. N. Ibbett and N. Topham, "Work in Progress: Performance Evaluation of RAID6 Systems", in *WiP of FAST*, San Jose, USA, 2007.

- Y. Li, T. Courtney, F. Chevalier and R. N. Ibbett,"SimRAID: An Efficient Performance Evaluation Tool for RAID Systems", in *Proc. SCSC*, pp 431-438, Calgary, Canada, 2006.

- T. Courtney, F.Chevalier and Y. Li, “Novel technique for accelerated simulation of storage systems”, in *IASTED PDCN*, pp 266 - 272, Innsbruck, Austria, 2006.
- Y. Li and A. Goel. ”An Efficient Distributed Hot Sparing Scheme In A Parity Declustered RAID Organization”, under US patent application. (application number 12247877).

Acknowledgements

Many thanks to my supervisor, Prof. Roland Ibbett, for his invaluable supervision, warm-hearted encouragement, patience to correct my non-native English and understanding throughout my PhD. Without him, this Ph.D. and thesis would never have happened. Thanks must also go to Prof. Nigel Topham, my current supervisor, for his supportive supervision and discussion and for his trust and support. Without his support and trust, I wouldn't have been able to finish this thesis.

In addition, many thanks to my colleagues Tim Courtney and Franck Chevalier for their help and discussions. Their help has been invaluable for me, especially at the beginning of this PhD. My sincere thanks must go to David Dolman for solving HASE problems and maintaining HASE. Without his help and quick response, I wouldn't be able to finish my experiments. Also thank to my other colleagues in the HASE group, Juan Carballo and Worawan Marurngsith for the useful discussions and help over the years.

Many thanks to Atul Goel and Tom Theaker from NetApp for their help and support during my internship in NetApp. I learned a lot from Atul on designing useful and practical systems. Chapter 6 of this thesis is based on my intern project. Tom Theaker has been a very supportive boss and I really appreciate the internship and job opportunities he gave me.

Many thanks to my friends Lu, Yuanyuan, Lin and Guo for their friendship. Talking with them has always been my pleasure. Finally, my sincere thanks to my beloved parents and my dear husband Zhiheng for their unwavering support through all these years.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Yan Li*)

To Zhiheng and Grace.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Redundant Arrays of Inexpensive Disks (RAID) were introduced to meet the performance gap between main memory and hard disks in the late 1980's [77]. Due to their high performance and reliability compared with Single Large Expensive Disks (SLEDs), RAID systems have been the dominant products for back-end storage systems for more than two decades, including both primary storage systems and near-line systems (backup and archival storage systems). Currently they face unprecedented capacity challenges from data intensive applications such as image processing, transaction processing and data warehousing, so the scale and complexity of RAID systems are growing at an unprecedented rate. For example, the EMC^{TM} Symmetrix DMX-4 can be configured with up to 2400 disks [29], and the NetApp^{TM} FAS6000 series can support more than 1000 disks per node, with up to 24 nodes in a system [74]. Such large RAID systems usually consist of one or two specialised RAID controllers (software or hardware), a number of hard disks and the back-end network (backplane). The controllers transparently partition I/O requests over multiple disks to make them appear like a single large logical disk to users. They also provide resiliency mechanisms to tolerate failures in the storage subsystems. The RAID controllers are connected to the disks through a Host Bus Adapter (HBA) and the back-end network. It becomes increasingly challenging to build high performance and reliable RAID systems as the scale of systems keeps increasing. For instance, as more disks are added into a system, higher contention for the back-end network can limit system performance. Using more HBAs means adding more expense to the system design. Moreover, a system needs more interconnection components such as shelf enclosures and network connections to accommodate such a large number of disks, leading to reduced system reliability caused by component failures. Furthermore, due to rapid increases in disk capacity

and the slow improvement in disk access speed, RAID system are facing much longer disk reconstruction times than before, which can also reduce system reliability. All of these factors will eventually limit system scalability. Traditional RAID systems use scale up architecture, which is one big system. Next generation of RAID systems uses scale out architecture [72], which consists of multiple small storage system and these small storage system are connected together to provide unified and protected storage space. The scale of these small systems are still limited by aforementioned factors.

This dissertation studies both performance and reliability aspects of the scalability of RAID systems through simulation. A parameterised simulation model of RAID systems called SIMRAID has been created using HASE, a Hierarchical computer Architecture design and Simulation Environment developed at the University of Edinburgh, and appropriate simulation experiments have been carried out. In terms of performance, the dissertation studies the scalability of back-end networks. In terms of reliability, it proposes three approaches to improve system reliability, including a novel parity declustering data layout that can survive physical interconnect failures or correlated disk failures, a system architecture and rebuilding mechanism for fast disk reconstruction, and an efficient distributed hot spare allocation algorithm for general parity declustering data layouts.

The remainder of this chapter is organised as follows: Section 1.1 briefly introduces the architecture and basic concepts of RAID systems; Section 1.2 describes the technology trends in RAID system architecture design; Section 1.3 discusses the challenges that modern RAID systems are facing; Section 1.4 proposes the PhD project objectives; Section 1.5 highlights the dissertation contributions; and Section 1.6 provides an overview of this dissertation.

1.1 RAID Systems

A RAID system is a storage system that combines a number of hard disks together to achieve higher performance, reliability and data volume. Figure 1.1 shows a basic architecture of a mid-sized RAID system. A RAID system consists of one or two specialised RAID controllers (software or hardware), a number of hard disks and the back-end network (backplane). The controllers transparently partition I/O requests over multiple disks to make them appear like a single large logical disk to users. They also provide resiliency mechanisms to tolerate failures in the storage subsystems. The RAID controllers are connected to the disks through a Host Bus Adapter (HBA) and

the back-end network. In mid-sized or large-sized RAID systems, disks are mounted in shelf enclosures that provide power supplies, cooling services and pre-wired backplanes for the disks. One or more shelf enclosures can be connected via cables to the HBA. In high-end RAID systems, there are usually redundant back-end networks connected from the controllers to the disks to provide higher reliability, as shown by the dashed lines in Figure 1.1.

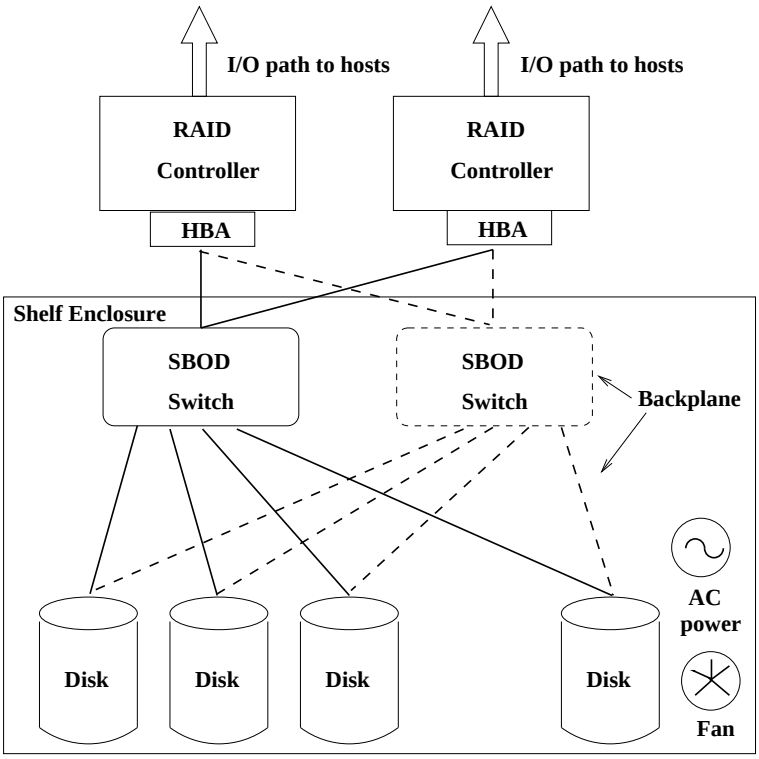


Figure 1.1: **A basic architecture of a mid-sized Fibre Channel (FC) connected RAID system.** SBOD refers to Switched Bunch of Disk.

Two basic concepts used in RAID systems are *data striping* to improve performance and *redundancy* to improve reliability.

1.1.1 Data Striping

RAID systems improve system performance by serving multiple disk commands in parallel. This process is called data striping. There are two aspects to this parallelism. Firstly, individual large requests can be served by separate disks acting in coordination, which increases the effective transfer rate for a single request, as shown in Figure 1.2 (a). Secondly, multiple, independent small requests can be served in parallel by multiple separate disks, which decreases the queueing time for each I/O request and

increases the I/O rate, as shown in Figure 1.2 (b). Potentially, the more disks there are in a system, the higher the performance [61, 44].

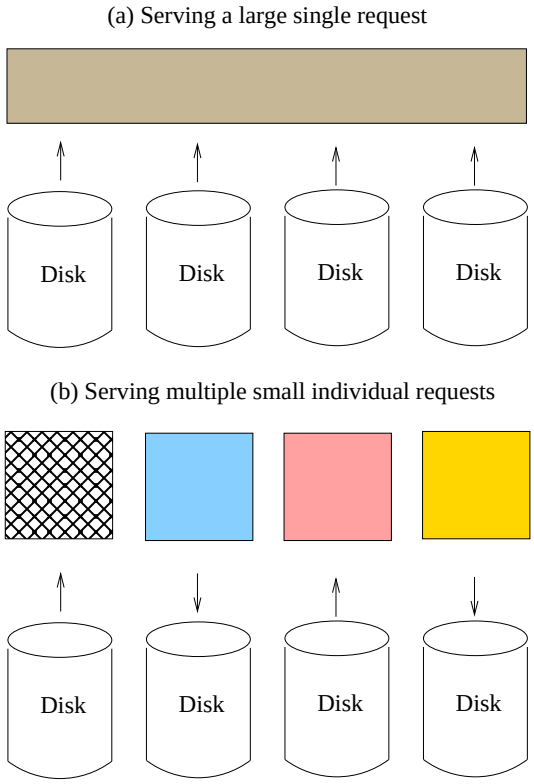


Figure 1.2: **Data Striping.** (a) illustrates the disk array serving a large single request; and (b) illustrates the disk array serving multiple small requests in parallel. (figure from [77])

1.1.2 Redundancy

System reliability is measured by the *mean time to fail (MTTF)* of the system. Using a large number of disks decreases the overall reliability of the disk array. Assuming the failures of each disk are independent, 100 disks collectively have only $1/100$th reliability of a single disk. To overcome this reliability challenge, redundancy is deployed to tolerate disk failures and to avoid data loss. In particular, the disk arrays are divided into *protection groups*, with each group having one or more extra *redundancy disks* containing the redundant information, as shown in Figure 1.3 (a). When a disk fails, it will be replaced by a new one and the information on the failed disk will be reconstructed using the redundant information (see Figure 1.3 (b)). The time to reconstruct a disk is called the *reconstruction time.* The expected reconstruction time

is called *Mean Time To Repair (MTTR)*. With one redundancy disk in each protection group, the MTTF of a RAID system is the mean time to two disks failing at the same time in one protection group. Assuming there are N disks in a RAID system and each protection group contains G disks, the MTTF of such a RAID system is calculated in [77] as:

$$\mathbf{MTTF_{system}} = \frac{MTTF_{disk}{}^2}{N * (G-1) * MTTR_{disk}} \tag{1.1}$$

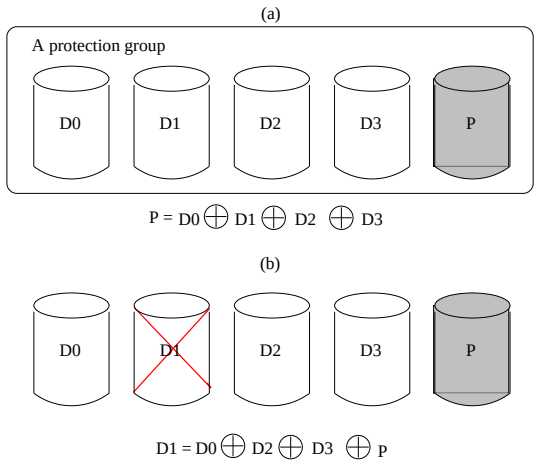


Figure 1.3: **Illustration of redundancy.** 'D0', 'D1', 'D2' and 'D3' represent data, and 'P' represents redundancy information.

1.1.3 Orthogonal RAID

The MTTF described above only considers disk failures. However, in real systems the failure of other components can also lead to data service loss. Schulze *et al.* [87] showed that the failure of the HBA, the power supplies and fans cannot be ignored. In order to survive these component failures, they proposed a data layout called *Orthogonal Data Layout* to improve system reliability, as shown in Figure 1.4. In this orthogonal data layout, protection groups are mapped onto the disk array that is orthogonal to the interconnection cable, the AC power, and the cooling groups, so that no single hardware failure will cause data loss. Because loss of any hardware affects only one hard disk in each protection group, each of these is recoverable. Moreover, this orthogonal data layout has the benefit of minimizing network conflicts when multiple disks from a protection group transfer data simultaneously.

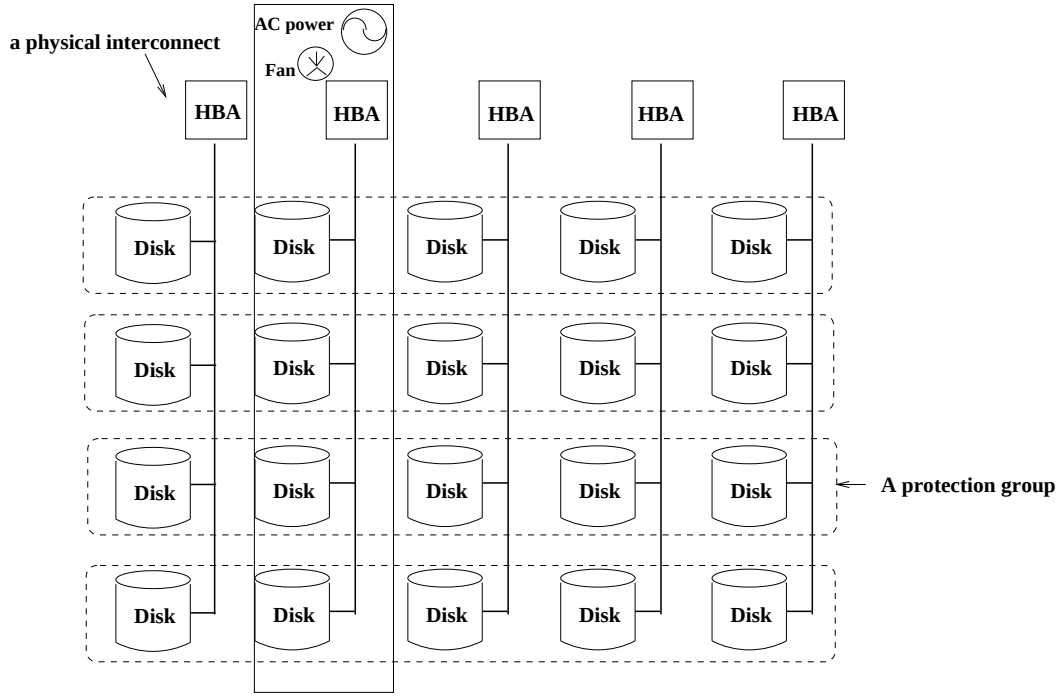


Figure 1.4: **Orthogonal data Layout.**

1.1.4 Degraded/Rebuilding Performance and Parity Declustering Data Layouts

Many applications require continuous service from storage, which means that the RAID system must be able to serve user requests even in the presence of disk failures and during the process of rebuilding the failed disks. In the organisation described in Section 1.1.2, when a disk fails, the system needs to read from all the other data disks and the redundancy disk to serve requests to the failed disk, which increases the workload to the surviving disks by 100%. During the process of disk reconstruction, on the one hand the system needs to read from all the rest of the protection group to rebuild the content on the failed disk. On the other hand, the system needs to serve requests to the failed disk. Both of these operations significantly affect system performance. To improve the performance of the system whilst it is degraded/rebuilding and to shorten the disk reconstruction time, Muntz and Lui [73] proposed *parity declustering data layouts*[1]. Parity declustering data layouts separate the issue of protection group size (namely the number of disks in a protection group) and the disk array over which the groups are distributed. In particular, they set the number of disks in a disk array to

[1]They initially called it *clustered data layouts*. Holland and Gibson later renamed it *parity declustering* [41], which is more popularly accepted.

be larger than the number of disks in a protection group and distribute the protection groups over all disks. Figure 1.5 compares a non-parity declustering data layout with a parity declustering data layout. In the non-parity declustering data layout, the size of the protection group and the number of disks in the array are both equal to five. When a disk failure occurs, the extra workload for the degraded read and for the rebuilding is spread over the four surviving disks. However, in the parity declustering data layout, the protection group size is equal to four and they are distributed over five disks. Thus, there are $\binom{5}{4}$ choices for placing the protection groups. When a disk failure occurs, the extra load needs to read the three surviving disks of that group and they are evenly distributed over four surviving disks. In this way, each surviving disk is able to serve more requests. At the same time, the system can reconstruct $\frac{5}{4}$ groups at the same time, so that the disk reconstruction time is reduced.

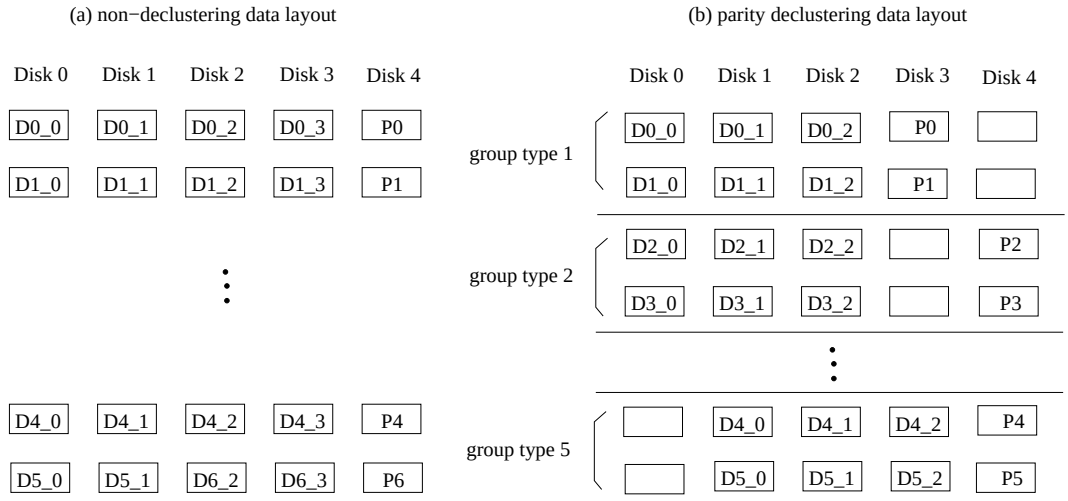


Figure 1.5: **Non-declustering data layout** *vs.* **parity declustering data Layout.** In (b), each group type corresponds to one way to select four disks out of the five. The blank block in (b) means that type of group does not reside on that disk. (figure from [73])

1.2 Technology Trends in RAID System Architecture

Although the basic concepts behind RAID systems have remained unchanged for many years, in order to achieve higher performance and efficiency, their architectures have evolved. In particular, current RAID systems show the following technology trends:

1. **Disk capacity is increasing dramatically:** As the areal density of hard disk

platters keeps increasing at an amazing rate, disk capacity is increasing at an accelerating rate. According to Shugart's law [46], disk capacity is increasing at 50% each year. The average disk capacity of a 3.5" disk was 0.15 TB in 2000, and increased to 1 TB by 2006. The average disk capacity of a single disk is expected to increase to 2 TB in 2010.

2. **The number of disks is increasing:** Although disk capacity has increased dramatically, the volume of data is increasing at an even higher rate. Therefore, the number of disks in storage systems has to be increased to deal with the increasing data volume. Increasing the number of disks not only increases the storage capacity but also improves system performance [61]. Therefore, the trend is to use large numbers of disks in a single system. For example, the number of disks in a 3U[2] shelf-enclosure in 2003 was 14, whereas this is expected to increase to 84 in a 4U shelf-enclosure in 2009.

3. **Less impressive growth in back-end network bandwidth:** Although back-end network bandwidth is also increasing, compared with the dramatic increases in disk capacity, the volume of data and storage system size, the growth in back-end network bandwidth is less impressive. The average bandwidth of a Fibre Channel (FC) cable in 2002 was 2 Gpbs. This increased to 3Gpbs in 2007, and is expected to reach 6 Gpbs in 2009. On average, back-end network bandwidth is increasing by 30% every year.

4. **RAID controllers are becoming increasingly sophisticated:** As communications networks become faster, it is necessary for storage controllers to perform their tasks faster to service the incoming traffic stream. Since Gilder's law states that communication bandwidth and usage doubles in 12 months while Moore's law states that silicon capabilities double in 18 months [39], increases in the rate at which storage controllers must deal with requests are outstripping increases in their processing power. So storage controllers are increasing in complexity to overcome this discrepancy. Moreover, due to the increasing demands on data service performance and reliability, the complexity and functionality of RAID controllers keep increasing. These new complexities are highlighting new bottlenecks in storage controllers that require further investigation.

5. **Storage systems are becoming heterogeneous:** As new high speed serial in-

[2] 'U' is the unit for height of shelf-enclosures. 1 U = 1.75 inch.

terconnection technologies are introduced, such as Serial Attached SCSI (SAS) [94], full systems no longer have the luxury of being based on a single standard with the associated advantages that this brings. This can lead to a capability mismatch between the communication networks on either side of the storage controller. There are also new storage technologies being developed to interoperate seamlessly with disks or to replace them, such as Solid State Disks (SSD) [2]. There is increasing interest of shipping hybrid systems containing both hard disks and SSDs. Simulated performance analysis of these hybrid systems will enable manufacturers to optimise cost/performance of their systems.

1.3 RAID Systems Challenges

Because increasing the number of disks not only increases the storage capacity but also improves system performance [61, 44], the scale of RAID systems in disks keeps increasing. However, as the scale increases, RAID systems are facing both performance and reliability challenges.

1.3.1 Performance Challenge - Limited Back-end Network Bandwidth

As described in Section 1.2, the technology trend is to use a large number of small disks rather than a few big disks, since increasing the number of disks not only increases the storage capacity but also improves system performance. However, there are system bottlenecks that limit the number of disks that can be added to a system. The back-end network is one such bottleneck. Because the back-end interconnection networks are shared by all the disks and the RAID controllers, the more disks that are added to the system, the higher the contention for the shared media. Beyond a certain threshold, adding more disks will give no further gain in performance, due to saturation of the back-end network. As described in Section 1.2, although network bandwidth has been increasing rapidly, the growth rate of network bandwidth is less than the scale of storage systems. In addition, the design of backplanes at and beyond 6-7 Gbps is very complex and expensive due to interference between links [43], increasing the cost penalty for either increasing link speed or increasing the number of pathways. A Qlogic 2Gb Dual Fibre Channel HBA is asked for more than 1300 dollars [83] and a 8G Fibre Channel HBA is asked for nearly 3000 dollars [84]. Adding more

HBA to the systems means adding more expense to the systems. It is thus essential to study the scalability of the back-end network to help design a cost-effective scalable storage subsystem. As technology moves towards a service oriented model, it is also necessary to be able to understand the service levels achievable by a given device. In the case of storage, this is, or can be in some scenarios, governed by the back-end network.

1.3.2 Reliability Challenges

It has been increasingly challenging to build reliable RAID systems as the scale and complexity of RAID systems grows. In particular, designers face the following challenges:

- **Physical Interconnect Failures** A comprehensive study of RAID system failures conducted by Jiang *et al.* shows that disk failures are not the only factors that contribute to RAID system component failures - physical interconnect failures (including shelf enclosures) also account for significant percentages of failures [49]. In near-line systems, physical interconnect failures make up 27% of system failures, whereas that number is 68% and 43% for low-end and high-end primary storage systems respectively. These interconnect failures are usually caused by HBA failures, broken cables, shelf enclosure power outage and shelf backplane errors.

  As a cable or a shelf enclosure connects to multiple disks, their failures affect all the disks connected to (or mounted in) them. When a physical interconnect failure occurs, the affected disks appear to be missing from the system. From the controller's point of view, multiple disks from the same loop or shelf are unavailable. The widely used RAID protection algorithm RAID 5 can only protect at most one disk failure from one protection group. Even RAID 6 [79], which has been extensively researched recently, protects against at most two disk failures. As physical interconnect failures cause multiple disks from the same protection group to be unavailable, these failures result in loss of data service during the period of the failures. The orthogonal RAID described in Section 1.1.3, which selects at most one disk (or two for RAID 6) from each loop to form a protection group, could survive such loop failures. However, as described in Section 1.1.4, such data layouts suffer from poor degraded performance and rebuilding performance. Although parity declustering data layouts [41] have excellent degraded

and rebuilding performance, most existing parity declustering data layouts do not consider the situation where multiple disks are unavailable. Thus, under physical interconnect failures, the parity declustering data layouts will lose data availability. It is obvious that as the scale of systems increases, the possibility of having physical interconnects failures also increases. Physical interconnect failures will eventually limit system scale. Therefore, it is important to combine the advantages of orthogonal data layouts and parity declustering data layouts to design a new data layout that can not only survive multiple disk failures resulting from loop or shelf failures, but also has good degraded performance and rebuilding performance.

- **Correlated Disk Failures.** To simplify analysis, it is traditionally assumed that disk failures follow an exponential distribution and that they are independent of each other. However, more and more research indicates that there is a high correlation between disk failures. Schroeder *et al.* [86] studied the disk replacement log collected from a large number of storage systems. They found that the time between disk failures in the same machine room exhibited significant levels of correlation. Jiang *et al.* [49] also found that disks failures have high temporal locality and disks from the same shelf enclosure show a bursty failure pattern. There are several explanations for this high correlation: (1) disks from the same machine room/shelf-enclosure are usually the same age and the same model - [86] shows that disk replacement rates grew constantly with age; (2) disks from the same machine room/shelf-enclosure operate in the same environment, such as the same temperature. Pinherio *et al.* [78] show that at high temperatures ($> 45^o$) the disk failure rate grows as the temperature grows.

  Like physical interconnect failures, correlated disk failures can cause multiple disks from the same protection group to become unavailable. The solution to this problem is to design a new data layout that combines the advantages of orthogonal data layouts and parity declustering data layouts.

- **Long Disk Reconstruction Time.** As shown in Equation 1.1, system reliability is inversely proportional to the disk reconstruction time. The longer the disk reconstruction time, the longer the system vulnerability window, which means a higher possibility of data loss. Hence it is important to reduce the reconstruction time. In addition, this time has become more important with higher demands on scalability. Increasing the capacity of RAID systems increases the probability

of disk failures, resulting in more frequent disk reconstructions. The disk reconstruction time is determined by the time to read data from the surviving disks and the time to write the reconstructed data to the replacement disks. However, two factors limit the data read and write speed: the slow disk access speed and limited back-end network bandwidth.

Compared with the dramatic increases in disk capacity, improvements in disk access speed have been much smaller - while disk capacity is more than 1000 times greater than two decades ago, disk access times have just halved. Therefore, much longer time is needed to reconstruct a disk than before. For instance, assuming that all the disk bandwidth is used for disk reconstruction and the rebuilding speed stays at 40MB/sec, it takes at least 10 hours to build a 1 TB disk. Although a variety disk reconstruction algorithms have been investigated, there has been no research on accelerating the disk reconstruction speed from the aspect of system architecture design. In addition, most research assumes that a dedicated replacement disk is used during reconstruction, which means that the overall reconstruction speed will be restricted by the speed of a single disk. Distributed hot sparing [68] allows reconstruction of several disks at the same time so that the rebuilding speed is not restricted by the disk write bandwidth. However, there is no efficient algorithm for assigning distributed hot spare space for a parity declustering data layout. Thus, it is important to review the disk reconstruction problem and to design and evaluate a system architecture that can provide a fast disk reconstruction speed.

Moreover, in an Online Transaction Trocessing (OLTP) environment, most of the requests to disk are random accesses in normal operation mode. However, disk reconstruction involves sequential reads and writes, which means that a larger data rate is transmitted over the back-end network during the disk reconstruction process and more network bandwidth is required. A back-end network which is adequate in normal operation mode might become a bottleneck during the disk reconstruction process. How to reduce the effect of this problem while designing a system architecture for fast disk reconstruction is also of interest.

1.4 Research Objectives

In order to address the aforementioned challenges and meet the scalability demands of RAID systems, this dissertation studies both performance and reliability aspects of scalability of RAID systems through simulation. In particular, it pursues the following objectives:

- From the aspect of performance, it investigates the scalability of the back-end network, namely the capacity of the back-end network in terms of the number of disks that can be attached to one link.
- From the aspect of reliability, it pursues the following sub-tasks:
  - the design and evaluation of a parity declustering data layout that can not only survive multiple disk failures caused by physical interconnect failures and correlated disk failures, but also has good degraded and rebuilding performance
  - the design and evaluation of a system architecture for fast disk reconstruction
  - the design of an efficient distributed hot spare assignment algorithm for parity declustering data layouts.

1.5 Thesis Contributions

The major contributions of the work described in this dissertation are as follows:

- Co-development of a discrete-even driven storage simulator SIMRAID. I am the main contributor to the SIMRAID simulator software. My work has included verifying the network abstraction technique and designing and developing the SIMRAID software.
- Studies of the scalability of the back-end networks of RAID systems. In particular, two problems have been solved: (1) determining the factors that affect the back-end network bandwidth requirement of the storage system; (2) identifying the network scalability and saturation points for a number of conventional system configurations. The major results achieved for this part include:

  - the bandwidth requirement per disk is primarily determined by workload features and stripe unit size. Cache size and RAID algorithms have very little effect on this value.
  - the number of disks that a system can scale to is limited by the back-end network bandwidth. A smaller stripe unit size has better scalability than a larger one.

- The development and evaluation of a novel parity declustering data layout which not only survives multiple disk failures caused by physical interconnect failures or correlated disk failures, but also has a good degraded performance and rebuilding performance. Its generating process is simple, deterministic and efficient in terms of both storage and time. The size of a rotation is small. Deploying this data layout can significantly improve the system reliability compared with those only deploying RAID 5 systems. The performance of this data layout is comparable to other parity declustering data layouts.

- Design and evaluation of a system architecture and a rebuilding mechanism for fast disk reconstruction.

- A proposal for an efficient distributed hot spare allocation and assignment algorithm for general parity declustering data layouts. This algorithm avoids conflict problems in the process of assigning distributed spare space for the units on the failed disk. Simulation results show that it effectively solves the write bottleneck problem. At the same time, there is only a small increase in the average response time of user requests.

The publications that are related to this dissertation include:

- Y. Li, T. Courtney, R. N. Ibbett and N.Topham, “On the Scalability of the Back-end Network of Storage Sub-Systems”, in *SPECTS 2008*, pp 464-471, Edinburgh UK.

- Y.Li and R.N. Ibbett, “SimRAID-An Efficent Performance Evaluation Tool for Modern RAID Systems”, in *Poster Competition of University of Edinbrugh Informatics Jamboree*, Edinburgh, 2007. the First prize

- Y. Li, T. Courtney, R. N. Ibbett and N.Topham, “Work in Progress: On The Scalability of Storage Sub-System Back-end Network”, in *WiP of FAST*, San Jose, USA, 2007.

- Y. Li, T. Courtney, R. N. Ibbett and N. Topham, “Workin Progress: Performance Evaluation of RAID6 Systems”, in *WiP of FAST*, San Jose, USA, 2007.
- Y. Li, T. Courtney, F. Chevalier and R. N. Ibbett,“SimRAID: An Efficient Performance Evaluation Tool for RAID Systems”, in *Proc. SCSC*, pp 431-438, Calgary, Canada, 2006.
- T. Courtney, F.Chevalier and Y. Li, “Novel technique for accelerated simulation of storage systems”, in *IASTED PDCN*, pp 266 - 272, Innsbruck, Austria, 2006.
- Y. Li and A. Goel. ”An efficient distributed hot sparing scheme in a parity declustered RAID organization”, under US patent application. (application number 12247877).

1.6 Thesis Overview

This introductory chapter has briefly introduced the research presented in this dissertation. The remainder of the dissertation explains in detail the research background, the related work and the simulation model, SIMRAID. Following that, it presents research on the scalability of back-end networks of RAID systems and how to improve RAID system reliability.

The dissertation outline is as follows:

Chapter 2 provides background information about RAID systems. Firstly, RAID system implementation issues are discussed, including the definition of RAID levels and some terms, data layouts on disk arrays, the read and write operations, and disk reconstruction algorithms. Secondly, the physical components of RAID systems are introduced: hard disk drives, RAID controllers, back-end networks and shelf-enclosures.

Chapter 3 presents a survey of other research work that is related to this dissertation. This includes storage system performance modelling and evaluation, RAID system reliability studies, parity declustering data layout design, distributed hot sparing and simulation of storage systems.

Chapter 4 presents the design and implementation of the RAID system simulation model SIMRAID. Firstly, the simulation model development environment, HASE, is introduced, including its facilities and ability to model RAID systems. Secondly, the design and implementation of SIMRAID are presented. Thirdly, the validation of SIMRAID is described. Lastly, the model’s performance is discussed.

Chapter 5 studies the scalability of the back-end networks of storage sub-systems in terms of the number of disks that can be linked to the network. In particular, Fibre Channel (FC) Switched Bunch of Disks (SBOD) [30] has been chosen as the research subject, since it represents the current state of the art in scalable back-end storage sub-systems. This chapter aims to answer the following two questions: first, given a number of disks, which factors affect back-end network bandwidth requirements of disks; second, given an interconnection network, how many disks can be connected to the system.

Chapter 6 proposes several approaches to improve the system reliability and scalability. Firstly, it proposes a novel parity declustering data layout that can survive physical interconnect failures and correlated disk failures. Secondly, it presents the design and evaluation of a system architecture and a rebuilding mechanism for fast disk reconstruction. Lastly, it develops an efficient distributed hot spare allocation and assignment algorithm for general parity declustering data layouts.

Chapter 7 presents a summary of the findings and contributions of this dissertation to the field of storage system design. This chapter also outlines future prospects and directions of this research.

Chapter 2

RAID Systems

This chapter provides background information about RAID systems. Firstly, RAID system implementation issues are discussed, including the definition of RAID levels and some terms, data layouts on disk arrays, the read and write operations, and disk reconstruction algorithms. Secondly, the physical components of RAID systems are introduced: hard disk drives, RAID controllers, back-end networks and shelf-enclosures.

2.1 RAID System Implementation Issues

As introduced in Chapter 1, the two basic concepts of RAIDs are the use of data striping to improve the performance and redundancy to improve reliability. In practice, there are numerous ways of implementing a RAID system and the implementation details have a significant effect on system performance and reliability. This section discusses the issues involved. Firstly, the definitions of basic RAID levels are introduced and their performance and storage efficiency are compared. Secondly, data layouts on disk arrays are introduced. Thirdly, the read and write operations are described. Lastly, disk reconstruction algorithms are discussed.

2.1.1 Basic RAID Levels

Based on the granularity of data interleaving and the methods and patterns in which the redundancy data are computed and distributed across the disk array, RAID systems are classified into seven basic levels [21]. This section briefly describes their definitions and compares their performance and storage efficiency.

2.1.1.1 Definitions

Figure 2.1 illustrates the seven basic RAID levels schematically.

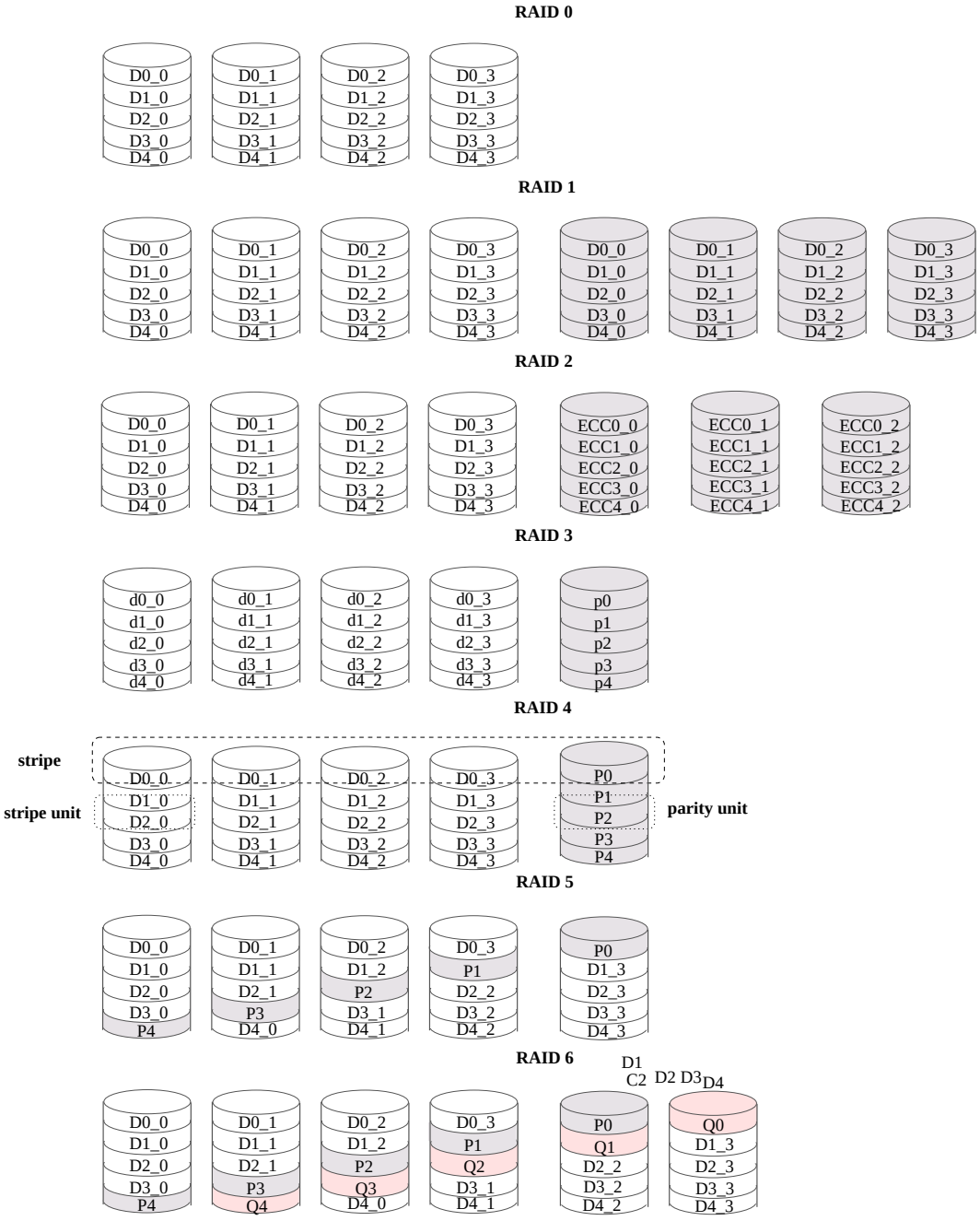


Figure 2.1: **Basic RAID levels.** White blocks refer to data and grey blocks refer to redundancy information. For RAID 6, pink blocks refer to the second independent redundancy information. Lower case letters refer to bits and upper case letters to blocks.(figure from [21])

Their definitions are as follows:

- **RAID 0 (Non-redundancy)** Data are interleaved across multiple disks, but no redundancy is provided.

- **RAID 1 (Mirrored)** There are two copies of all information. This scheme, also called **Data Mirroring**, uses twice as many disks as RAID 0.

- **RAID 2 (Memory-Style ECC)** RAID 2 uses Hamming code [62] as the redundancy information to protect data disks, which is similar to the protection scheme in semiconductor memory. RAID 2 provides recovery from data disk failures with less cost than RAID 1. For instance, four data disks require three redundancy disks. Upon a disk failure, among these three redundancy disks, only one is used to recover the failed disk with the other two being used to identify the failed disk. However, in the RAID environment, the RAID controller can easily identify which disk failed. Hence, the redundancy information for identifying the failed disk is just a waste. For this reason, RAID 2 is not a popular level.

- **RAID 3 (Bit-interleaved Parity)** Data are spread over disks by using bit-wise interleaving. Only one redundancy disk is used to tolerate any single disk failure. The redundancy disk is called a *parity disk* and stores the exclusive OR (XOR) of the data in the same position as the data on the data disks in the same protection group.

- **RAID 4 (Block-interleaved Parity)** Like RAID 3, RAID 4 also just uses one dedicated parity disk to protect data. The difference from RAID 3 is that it uses block-wise interleaving instead of bit-wise interleaving. The unit of data interleaving is called a *stripe unit*. The data stripe unit is called a *data unit* and the redundancy stripe unit is called a *parity unit.* The minimum collection of the stripe units (including both data units and parity units) over which the parity unit is computed is called a *stripe.* The number of disks in a stripe is called the *stripe width.*

- **RAID 5 (Blocked-Interleaved Distributed-Parity)** Like RAID 4, RAID 5 also adopts block-wise interleaving but instead of using one dedicated parity disk, RAID 5 distributes redundancy units over all of the disks.

- **RAID 6 (P+Q Redundancy)** RAID 6 is essentially an extension of RAID 5 which allows for additional fault tolerance by using a second independent parity unit. Data are striped over disks in RAID 5 style. Two independent parity units

are provided to tolerate a minimum of two disk failures. There are a number of encoding methods for RAID 6, such as EVENODD [15] and RDP [26]. Normally, the first parity, also called *P parity*, is the same as RAID 5 parity. The second parity, also called *Q parity*, is independent of the first.

These basic RAID levels can be combined together to form more complex RAID levels to provide higher performance and reliability, for example RAID 0+1 (mirrored stripes)and RAID 1+0 (striped mirrors). RAID 0+1 creates two RAID 0 stripes, each of them mirroring the other. RAID 1+0 creates RAID 0 stripes over two sets of mirrored disks.

2.1.1.2 Comparisons

Table 2.1[21] tabulates the throughput and storage efficiency of these basic RAID levels[1] relative to RAID 0. To simplify the comparison, it compares systems with equivalent *file capacity*, which is the amount of information that the file system can store on the device and excludes the storage used for redundancy.

	Small Read	**Small Write**	**Large Read**	**Large Write**	**Storage Efficiency**
RAID 0	1	1	1	1	1
RAID 1	2	1	2	1	50%
RAID 3	$1/D$	$1/D$	1	1	$D/(D+1)$
RAID 4	1	$D/2(D+1)$	1	1	$D/(D+1)$
RAID 5	$(D+1)/D$	$max(1/D, 1/4)$	1	1	$D/(D+1)$
RAID 6	$(D+2)/D$	$max(1/D, 1/6)$	1	1	$D/(D+2)$

This table compares the throughput of various RAID levels for four types of I/O request. *Small* refers to I/O requests of one stripe unit. *Large* refers to I/O requests of one full stripe. D is the number of data disks in each protection group. The entries in the table only account for major performance effects, not some second-order effects.

Table 2.1: Throughput and Storage Efficiency Relative to RAID 0

As shown in Table 2.1, RAID 0 offers the best write performance since there is no need to update any redundancy data. However, as there is no redundancy, disk failures will lead to data loss.

[1] As RAID 2 is seldom used in practice, it is not listed in this table

RAID 1 has the best read performance. Because there are two copies, when data is read, the disk with the shorter queue will respond [20]. However, compared with other RAID levels, RAID 1 has the highest redundancy overhead. RAID 1 is widely used in environments in which the transaction rate is critical but the storage efficiency is less important. In addition, RAID 1 is also used quite a lot in very small RAID systems where there are just two disks since it is the only RAID scheme one can be implemented by adding one disk to a single-disk system.

RAID 3 has high read and write performance comparable with RAID 1 but it has the worst small read performance. Because even small requests in RAID 3 need to access all data disks (including the parity disk in case of write requests), only one request can be served at a time. Nevertheless, RAID 3 is easy to implement, therefore, RAID 3 is frequently used for applications that require high bandwidth but not high I/O rate, *i.e.* ones involving small numbers of large requests.

RAID 4 has the same throughput as RAID 0 under read workloads and large write workloads but its small write performance is restricted by the group size because of the dedicated parity disk. For each write request, the requested data units and the corresponding parity disk need to be updated, therefore, this parity disk can easily become a performance bottleneck.

RAID 5 eliminates the performance bottleneck of RAID 4 by distributing the parity over all disks. In addition, by so doing, RAID 5 allows all disks, including the parity disk, to participate in small reads. Thus RAID5 has the best small read performance. However, compared with RAID 0 and RAID 1, for each small write request, RAID 5 needs to perform four disk operations, including reading the data and parity unit and updating the data and parity unit, resulting in poor performance. This performance problem is called *small-write problem*; there is extensive research on how to improve small-write performance [69, 67, 93].

The performance of RAID 6 is similar to that of RAID 5 except that for each small write request, RAID 6 needs to perform 6 disk operations because two parities need to be updated. Therefore, the small-write problem for RAID 6 is even worse than for RAID 5.

2.1.2 Reading and Writing Operations

There are three operation modes for RAID systems: *normal mode*, *degraded mode* and *rebuilding mode*. Normal mode refers to the state in which all disks are available for

reading and writing. Degraded mode refers to the state in which one or two disks have failed. Rebuilding mode refers to the state in which a new disk has replaced the failed one and the data on the failed disc are being reconstructed from the surviving disks and written to the replacement. The reading and writing operations are different in these three modes. Since RAID 1 operations are straightforward and RAID 2, RAID 3 and RAID 4 are not so popular, this dissertation focuses on the operation of RAID 5.

2.1.2.1 Normal mode

Reading operations in the normal mode are simple. Based on the address of the user request, the RAID controller first calculates the disk number and the disk address of the stripe units being requested and then reads them from the disks (see Figure 2.2 (a)). Writing operations are more complex since the RAID controller needs to update not only the data units but also the corresponding parity units. In order to reduce the number of disk operations, there are two methods to compute the parity units. The first is called *parity increment.* When the stripe units to be written are less than half a stripe (usually called small write), the RAID controller first reads the old data on the units to be written and the old parity units of that stripe from the disks. The new parity units are then calculated by XORing the new data with the old data and old parity units (see Figure 2.2 (b)). Namely, $new\ parity = old\ data \oplus new\ data \oplus old\ parity$.

The second method is called *reconstruct write.* When the stripe units to be written are more than half a stripe (usually called large write), instead of reading the old data units, the RAID controller reads the other data units (units not to be written) on that stripe. Using these data units and the new data, the RAID controller computes the parity units (see Figure 2.2 (c)).

2.1.2.2 Degraded mode

In degraded mode, reading from the surviving disks is the same as in normal mode. Read requests to a failed disk are served by reconstructing the data from the other stripe units of that stripe, including both data units and parity units, as shown in Figure 2.3 (a). Write operations are even more complicated. Small write requests to a failed disk need to read the rest of the stripe to compute the parity; writes to the failed disk in degraded mode are just suppressed (see Figure 2.3 (b)). If the parity unit happens to reside on the failed disk, as shown in Figure 2.3 (c), the controller just writes to the data disks without computing the parity. For large writes, there are three cases to be considered.

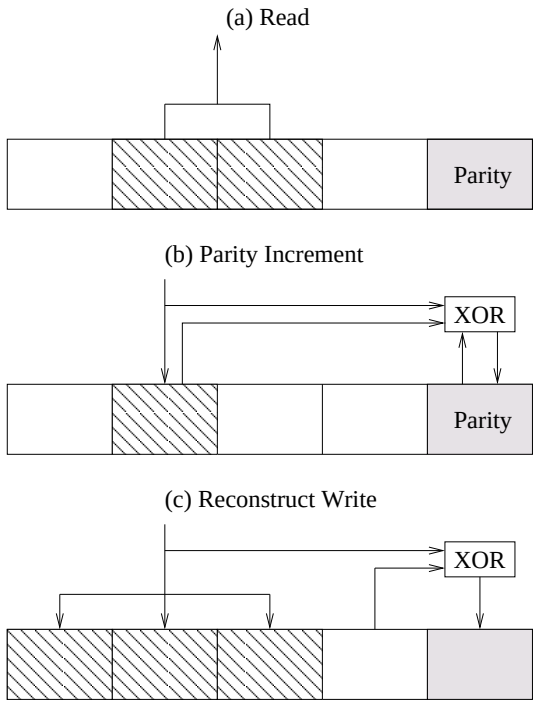


Figure 2.2: **Read/Write operation in normal mode.** Striped blocks refer to the data units to be read/written; grey blocks refer to the parity unit.

If the data units to be read for use in computing the parity reside on the failed disk, the RAID controller will read the old data and the old parity to compute the new parity, as shown in Figure 2.3 (d); if the data units to be written reside on the failed disk, the controller will compute the parity as in normal mode and just suppress the write to the failed disk, as shown in Figure 2.3 (e); if the parity unit resides on the failed disk, then as for small writes, the controller will just proceed with the write to the data units without computing the parity.

2.1.2.3 Rebuilding mode

RAID systems are able to serve user requests while rebuilding the failed disks. For operations on data on the surviving disks, the operations are the same as in normal mode. For user read requests to data that has not been rebuilt, the array operates as in degraded mode. For user write requests to the data being rebuilt, the controller will not write to that unit until it is rebuilt. In order to reconstruct the data on the failed disk, for each stripe unit, the RAID controller needs to read the rest of that stripe from the surviving disks, XOR them and send the reconstructed data to the replacement disk, as shown in Figure 2.4.

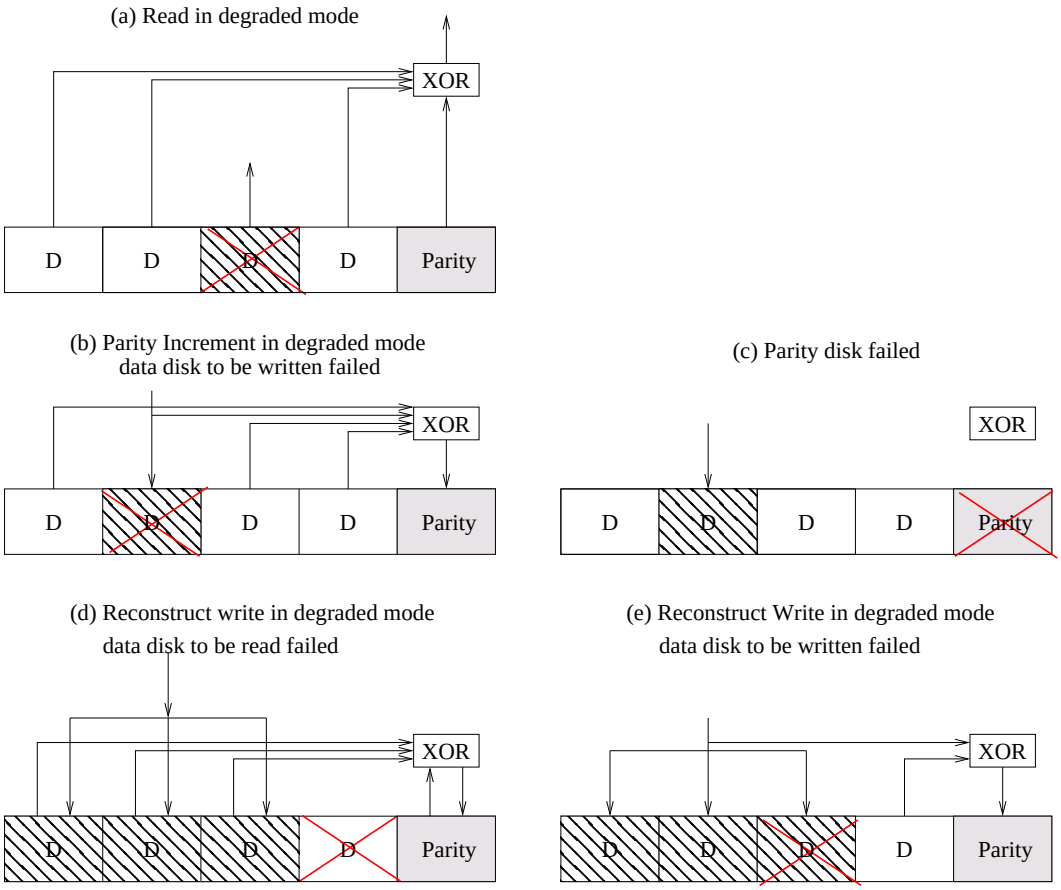


Figure 2.3: **Read/Write operation in degraded mode.** Stripped blocks refer to the data units to be read/written; grey blocks refers to the parity units. The failed disk is marked by a cross.

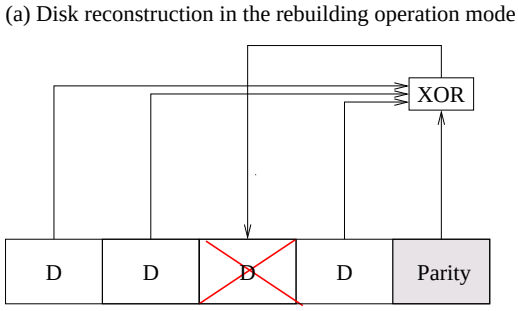


Figure 2.4: **Disk reconstruction operation.** Grey blocks refers to the parity units. The replacement disk for the failed disk is marked by a cross.

There are two methods to reduce the workload on the surviving disks during the reconstruction.

- Read redirection: User accesses to data that has already been rebuilt on the re-

placement disk are redirected to the replacement disk rather than invoking on-the-fly reconstruction. This optimisation reduces the number of disk read accesses during the rebuilding.

- Piggybacking of read: User data that is reconstructed on-the-fly is written to the replacement disk. This optimisation aims to speed up the reconstruction.

Holland & Gibson estimated the effect of these two method [41]. Their simulation results show that read redirection helps to reduce the user response time when the workload is heavy but the combination of piggybacking with read redirection leads to little difference in performance. Thus, this dissertation only exploits read redirection.

2.1.3 Data Layout

Data layout refers to the ways in which the data units and parity units are distributed over disks. It has a significant effect on system performance and reliability. A good RAID layout should not only be able to recover from disk failures but also be able to provide high performance. Holland & Gibson identified the following six desired properties that an ideal data layout should have [41].

1. Single failure correcting: no two stripe units in the same parity stripe may reside on the same physical disk.

2. Distributed parity: parity units should be evenly distributed over all disks.

3. Distributed reconstruction: the workload to rebuild a failed disk should be evenly distributed over all the surviving disks.

4. Efficient mapping: the function that maps user logical addresses to the stripe units in the array should be efficient in terms of both time and memory.

5. Large write optimization: a write to a contiguous full stripe data should not require pre-reading of the previous contents of any disk.

6. Maximal parallelism: requests that read n contiguous user units (where n is the number of disks in the array), should access all n disks.

However, Alvarez *et al.* later proved that ideal data layouts that satisfy all these six properties only exist under some special array configurations [11]. For most configurations, the data layout can only satisfy some of these six properties.

Early research on data layout were restricted to cases where the stripe width is equal to the number of disks on which the stripes are placed. Lee & Katz first studied the effect of the parity placement of RAID 5 on system performance [55]. Extending the RAID 4 data layout, Lee & Katz proposed six different parity placement schemes: right asymmetric, right symmetric, left asymmetric, left symmetric, flat left symmetric and extended left symmetric, as shown in Figure 2.5. The RAID 0 and RAID 4 data layouts are drawn here for comparison. Simulations were then carried out to study the performance of these six placements. Among them, the left symmetric placement has the best read performance under a low workload, whereas the right asymmetric has the best write performance under a low workload. Under a high workload, they have a comparable performance. Compared with other placement, the right asymmetric placement is easy to implement.

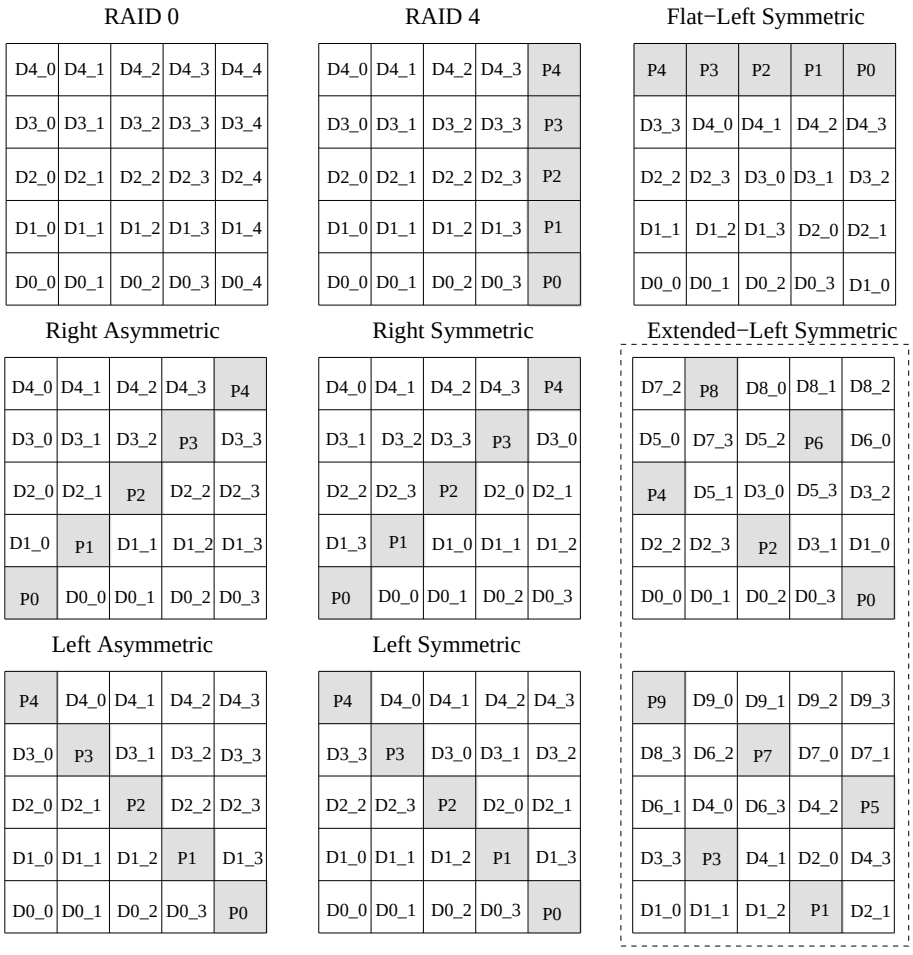


Figure 2.5: **Parity placement.** Each row is a stripe. White blocks represent data units and the numbers denote their logical addresses. Grey blocks represent parity units. Pi means that the parity belongs stripe i. (figure from [55])

However, as described in Section 1.1.4, setting the stripe width equal to the number of disks limits system performance in degraded and rebuilding operation modes. In the degraded operation mode, when a read occurs on the failed disk, the RAID controller has to read the surviving disks of that stripe to reconstruct the failed data (see Figure 2.3). In the rebuilding operation mode, the RAID controller has to read from the surviving disks of that stripe to reconstruct the failed disk (see Figure 2.4). In both cases, the workload on the surviving disks is increased. Increasing the user workload to the surviving disks leads to a reduction in the rebuilding rate, resulting in a longer reconstruction time. On the other hand, increasing the rebuilding rate will limit the user workload that the system can serve. To reduce the reconstruction time and support higher workload during reconstruction, Muntz & Lui [73] proposed *clustered RAID* data layout (Holland & Gibson later renamed it as *parity declustering* [41], which is more popularly accepted.) The parity declustering data layout sets the number of disks to be larger than the stripe width and distributes the stripes over all the disks. By so doing, the workload to the failed disk in the degraded mode only need access part of the disk array, therefore, it has a better degraded performance than RAID 5. In addition, the reconstruction workload of the failed disk just needs to access part of the surviving disk set. Thus the reconstruction workload on the surviving disks is reduced, resulting in a higher user workload under such a reconstruction workload. On the other hand, reducing the ratio of stripe width to the overall number of disks allows a higher reconstruction workload, leading to a shorter reconstruction time. A detailed review of existing parity declustering data layout is presented in Section 3.3.

2.1.4 Reconstruction Algorithms

To minimize the probability of losing data, RAID systems usually use on-line spare disk drives (called hot spares) so that data on the failed disk can be immediately rebuilt to the spare disk. A fast (online) reconstruction algorithm is crucial to RAID systems. A good reconstruction algorithm should not only shorten the rebuilding time of the failed disks but also have as little effect as possible on the system performance seen by the users.

The simplest reconstruction algorithm is stripe-oriented reconstruction [41]. In a single process stripe-oriented reconstruction algorithm, a process associated with a stripe sends *read* commands to the surviving disks. After reads to all surviving stripe units belonging to that stripe are completed, this process will execute XORing over

these units and send write commands to the replacement disks.

The drawback of the single process stripe-oriented reconstruction algorithm is that it cannot fully use the disk bandwidth that is not used by user requests. Two approaches have been proposed to overcome this drawback. The first approach is to rebuild multiple stripes in parallel [41]. In this approach, there are multiple independent reconstruction processes rather than one, each running the single process stripe-oriented algorithm with a different starting point. Parallel reconstruction substantially reduces the rebuilding time. The second approach is to exploit the disk-oriented reconstruction algorithm proposed by Holland & Gibson [42]. Instead of having a number of parallel reconstruction processes associated with stripes, the disk-oriented reconstruction algorithm creates *N* (the number of disks in the RAID system) processes, each of them associated with a disk. The process associated with each surviving disk reads the data units on that disk in order and submits the data to the RAID controller. Once all the data belonging to a particular stripe is ready, the RAID controller executes the XOR operation on all these data. The process associated with the replacement disk continually issues write commands to send the reconstructed data to the replacement disk. With this disk-oriented approach, the recovery process fully uses the array bandwidth and also reduces the time to reconstruct the failed disk significantly.

Compared with the parallel stripe-oriented approach, the disk-oriented approach is better at using the disk bandwidth during the process of reconstruction. Therefore, it leads to shorter rebuilding time but longer user response time. When the number of parallel reconstruction processes is large, these two approaches have very similar performance.

Beside these two basic reconstruction approaches, there has been extensive research trying to reduce the reconstruction time. One of the earliest proposals is *distributed sparing* [68]. Distributed sparing distributes the capacity of a spare disk across all the disks in the disk array. The distribution of spare capacity is similar to the distribution of RAID 5 parity units. Instead of having N data disks and one spare disk, distributed sparing has N+1 data disks, with $1/(N+1)$ of each disk containing the spare capacity. When a disk fails, the units on the failed disk are reconstructed to the corresponding spare unit. Because there are N+1 disks participating in the operation, distributed sparing has better performance in normal mode. Moreover, because each disk is only partially full due to the spare units, it takes less time to rebuild the failed disk. However, in order to maintain the original data layout, after rebuilding, the reconstructed data will eventually be copied to a permanent replacement. This process

creates extra work for the disk array. Furthermore, after rebuilding (before copy back to the permanent replacement), the reconstructed data that were originally on one disk are distributed across all disks, resulting in a concern for I/O intensive applications.

Tian *et al.* proposed Popularity-based multi-threaded Reconstruction Optimization (PRO) [98] to shorten the rebuilding time by exploiting the temporal and spatial locality of the workload. PRO divides the failed disk into a number of consecutive zones and rebuild the zones that contain high-popularity units prior to rebuilding other zones. After the data are rebuilt on the replacement disk, read requests to those data will be redirected to the replacement disk, as described in section 2.1.2.3. As data accesses to these zones are very frequent, PRO reduces the workload on the surviving disks, leading to shorter reconstruction time and smaller user response time. However, PRO is only suitable for read dominated workloads. For a workload that is not read-dominated, PRO does not seem to work well. Q. Xin has proposed a distributed recovery scheme called FAst Recovery Mechanism (FARM) [104] which reduces the recovery time by parallelising the rebuilding of the failed disk. An important feature of FARM is that the parity data and the spare space are required to be evenly distributed throughout the storage system. Once a disk has failed, the data on that disk will be reconstructed using several disks at once. It has only been applied to mirrored data layout (namely distributed RAID 1) so far.

2.2 Physical Components of RAID Systems

A RAID system consists of three basic physical components: hard disk drives, RAID controllers and back-end networks. Usually, these components are mounted in shelf-enclosures. This section briefly introduces these three physical components and shelf-enclosures.

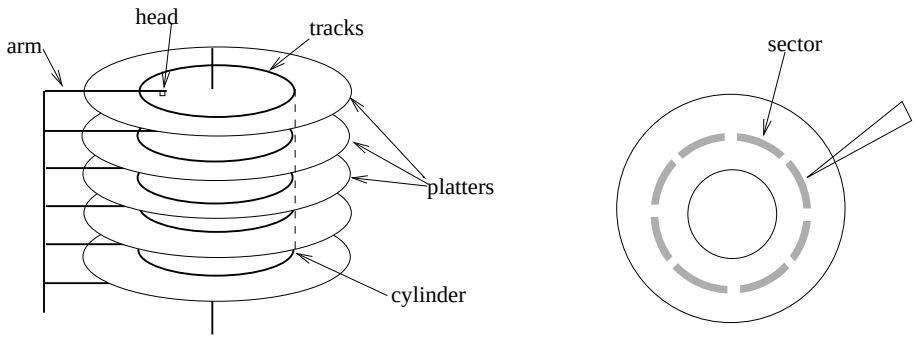
2.2.1 Hard Disk Drives

Hard Disk Drives (HDDs) have been the dominant non-volatile storage medium since 1965 [38]. As the detailed description of HDDs can be found in many books and in many disk manuals, this section only describes the essential feature of HDDs and the technology trend.

2.2.1.1 Physical Components

A modern HDD consists of three basic components: a communication interface, a disk controller and a set of platters. The communication interface manages the communication with clients. The market dominant interface protocols include Advanced Technology Attachment (ATA), Small Computer System Interface (SCSI) and Fibre Channel (FC). ATA is mainly used for personal disks, whereas SCSI and FC are mainly used for enterprise disks. The disk controller manages the interpreting and scheduling of the disk commands. A disk controller consists of a microprocessor and an embedded buffer which stores the disk commands to/from clients, enabling multiple request queuing at disks at a time. The buffers can also work as a read-ahead cache. The platters are the recording medium for storing information.

A platter is a metal or glass disc covered with magnetic material on both sides, rotating on a spindle at 3600 to 15,000 revolutions per minute (RPM), as shown in Figure 2.6 (a). Each platter is divided into a number of concentric circles, called *tracks* and each track is further divided into *sectors*, as shown in Figure 2.6 (b). A sector is the smallest unit that can be read or written. Sectors usually have fixed size of 512 bytes or 520 Bytes. The collection of tracks that have the same diameters on different platters are called *cylinders*. The disk controller reads or writes data through the heads (see Figure 2.6 (b)). Each head is connected with an arm and all these arms are connected together and move in conjunction.



(a) Side view of a disk (b) Top view of a platter

Figure 2.6: **Magnetic disk components.** (a) depicts the side view of a disk; (b) depicts the top view a disk.

The disk controller first stores the received disk commands in the buffer and then schedules them one by one according to its scheduling scheme, usually First Come

First Served (FCFS) or Shortest Seek Time First (SSTF) [40]. Reading data from or writing data to the disk media consists of three operations. First, the arms are moved to the track on which the data is located; the time to move the arms to the right track is called the *seek time*. Then, as the platters rotate, the first sector to be read comes under the head; this takes an amount of time that can vary between zero and the time for a complete rotation, so the average *rotation latency* is normally characterised as half the rotation time. Lastly, data are read/written from/to the disk media. The time to transfer data from/to disk media is called the *transfer time*. This time is a function of the data block size, the disk size, the areal density and the rotation speed. In summary, the disk access time is:

$t_{access} = t_{disk\ controlleroverhead} + t_{seek} + t_{rotation} + t_{transfer}$

Table 2.2 lists the main parameters of a Seagate 15.3 FC disk [90]. For sequential disk accesses, the access time is determined by the transfer time but for random accesses, the access time is dominated by the seek time and the rotation latency (2 ms for a disk rotating at a 15,000 RPM). Therefore, the average disk access time is around 5.8 ms.

Drive capacity	73.4 Gbytes
Rotation speed	15,000 RPM
Bytes per track	377, 770 Bytes
Transfer bandwidth	51.8-86.0 MBytes/s
Disk controller overhead	0.2 ms
Average seek time for read	3.6 ms
Average seek time for write	3.9 ms

Table 2.2: Disk features of a Seagate FC disk (2003)

2.2.1.2 Disk Technology Trends

Over the last two decades, disk capacity has increased dramatically. In contrast, the improvement in disk rotation latency and seek time is far slower [12]. Figures 2.7 shows the leading disk capacity and average access time over the last two decades. Disk capacity has increased more than 2000 times, whilst the disk access time just improved less than one order of magnitude. Compared with the CPU and memory latency, there is still a large performance gap between memory and hard disks.

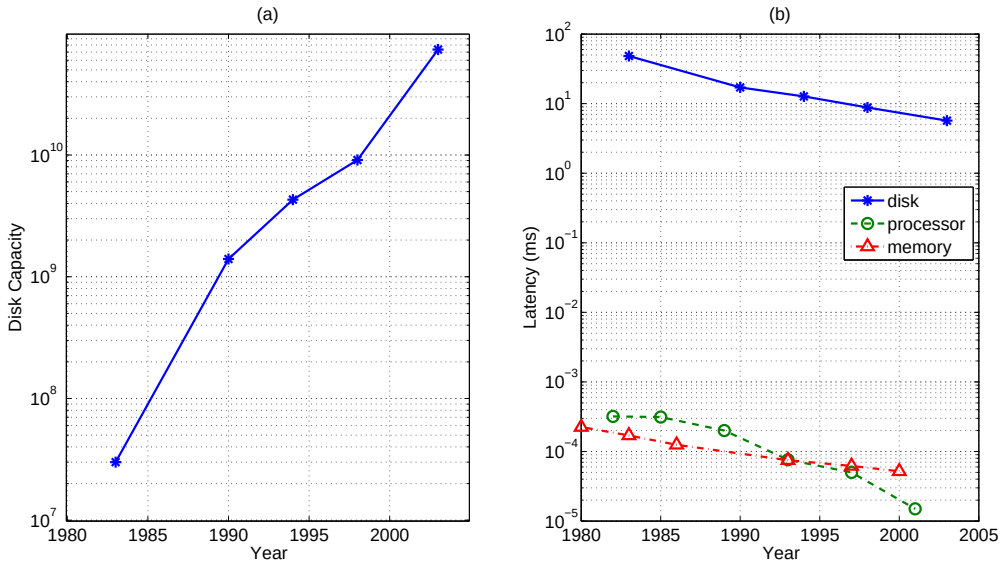


Figure 2.7: **Disk technology trends.** (a) depicts the disk capacity against the introduction year; and (b) depicts the average access time against the introduction year. The figures are generated from the data in [39], Chapter 1, page 16.

Over the last decade, numerous novel technologies which provide better performance have been introduced to replace hard disks, such as Solid State Disks (SSDs) [2] and Micro-Electro-Mechanical-Systems (MEMS) based storage [18]. SSDs use DRAM memory backed by battery or flash memory to store persistent data, leading to higher bandwidth and smaller access time. The average access time of DRAM-based SSDs is less than 0.15 ms access time, which is 250 times faster than HDDs. In contrast to SSDs, MEMS-based disks still use magnetic media to store data, but the access time is much faster than HDDs. Instead of rotating the surface, MEMS-based disks use an array of very small MEMS position probe tips which move linearly along X and Y directions to seek the data. The average access time of a MEMS-based disk is 0.67 ms [18], which is about 5 time faster than hard disks. However, performance is not the key factor to determine the future, and cost plays the most important role in the market. Due to their extremely high cost, there are still no MEMS-based products being shipped in large quantities. The price of SSDs is lower than MEMS-based disks and has kept decreasing since they appeared in the late 1990s. Due to decreasing of their price, there is increasing interest in using SSDs in RAID systems. They could be used in several ways: as a new layer of large cache, to store hot data or simply as substitutes for hard disks. However, compared with HDDs, they are still much more expensive.

Figure 2.8 compares the price of HDDs and SSDs over the last two decades. It can be seen that SSDs are about 1000 times more expensive than HDDs. Therefore, SSDs are now only used for applications that have critical requirements on performance but do not require large storage space. For large capacity storage systems, HDDs are still the first choice.

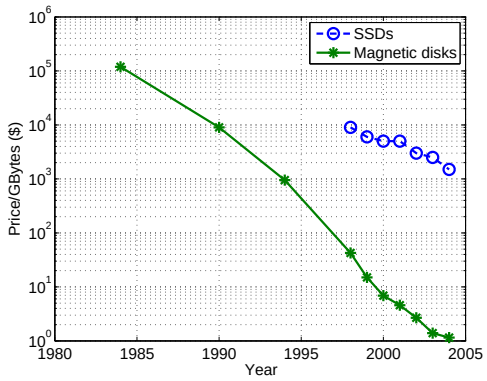


Figure 2.8: **Historical price comparison between hard disks and SSDs.** Data are quoted from [4] and [36].

2.2.2 RAID Controller

The function of the RAID controller is to make the disk array appear as a single big disk to hosts and to provide resilience mechanisms. RAID controller implementations contain components such as RAID tables defining the configuration of RAID arrays, data structures to store the descriptors for cached data, engine(s) for calculating redundancy information and the logic for handling I/Os to and from RAID arrays. They can be implemented in software, in hardware or in a hybrid way [8]. This section first discusses the difference between hardware RAID and software RAID. The architecture of hardware RAID controllers is then presented. Following that, the XOR operation implemented in hardware is described. Lastly, the write data mirroring process between RAID controllers is described.

2.2.2.1 Software RAID vs. Hardware RAID

Software RAIDs do not need any additional hardware and are usually implemented within the operating system of the host processor. Many operating systems (such as Windows XP/NT/Vista, Linux, Free BSD and MAC OS X) provide RAID functionalities. The main advantage of software RAID is its low price since it does not need

any extra hardware. However, as it shares the same CPU and memory resource with the operating system and other applications, the redundancy information calculation has a significant impact on the overall system performance. Hardware RAID requires special purpose hardware to run the RAID functionality. It costs more than software RAID but offers better performance. There are two main types of hardware RAID: (1) discrete RAID controller cards and (2) integrated hardware based on RAID-on-Chip (ROC) technology [9]. A discrete RAID controller card is a plug-in expansion card that usually has a built-in RAID processor (I/O processor) and its own interface to the disk drives. It usually plugs into the PCI-X or PCIe slots of the computer system's motherboard. This kind of hardware RAID is usually targeted at low-end application. The second type of hardware RAID, ROC, integrates the RAID processor, memory controller, host interface, I/O interfaces for hard disk drives and memory into one single chip. ROC-based RAIDs usually integrate two or more chips in one motherboard. These two chips are powered by two separate power supplies to improve reliability. In the case of one chip failing, the other one can take over its duty. In addition, the in-chip memory is usually protected by battery so it can work as a write-back cache to improve system performance. Due to their high performance and high reliability, ROC-based RAIDs are targeted at higher-end applications.

2.2.2.2 RAID Controller Architecture

A typical RAID controller chip [48] includes a main processor, a memory controller, an internal bus, two Direct Memory Access (DMA) controllers, an XOR engine, an Address Translation Unit (ATU), PCI-to-PCI bridge and PCI buses. Its architecture is shown in Figure 2.9[2]. The main processor is the core of the chip. It processes the incoming user commands and decomposes them into disk commands. The memory controller connects with an external memory subsystem and the RAID controller accesses the memory system through this memory controller. To improve performance, there is a direct high-speed interface between the main processor and the memory controller. The external memory usually works as controller memory and data cache. The XOR engine implements the XOR algorithm in hardware and performs XOR operations and parity calculations. The DMA controller allows low-latency, high-throughput data transfers between PCI bus agents (connected to I/O interfaces) and the local memory. The message unit/ATU allows PCI transactions direct access to the local memory.

[2] Modules that are not directly related to RAID functionalities are not introduced here

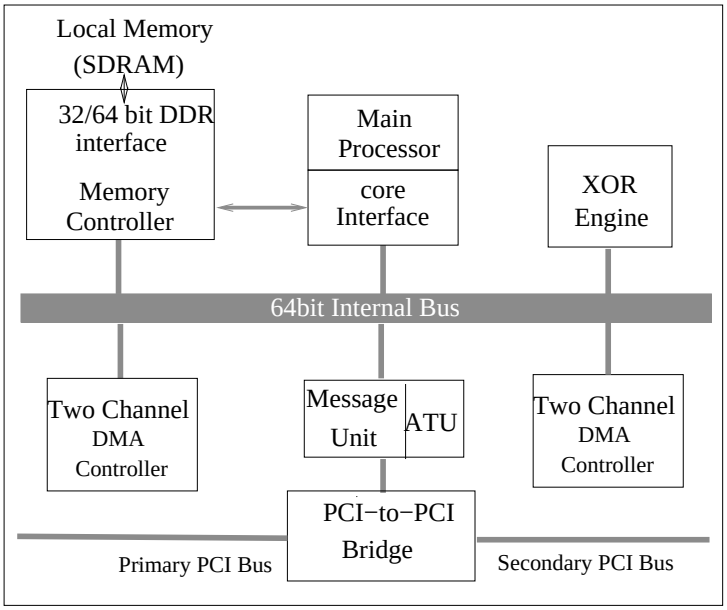


Figure 2.9: **Typical RAID Controller Architecture** (figure from [48]).

The PCI-to-PCI bridge and primary/secondary buses connect the I/O interface to hosts and disk drives.

2.2.2.3 XOR Operations

The XOR engine is the hardware dedicated to perform the XOR operations and parity calculations. It performs XOR operations on multiple blocks of incoming data and stores the result back in the local memory. The source and destination addresses are specified through *chain descriptors* that reside in the local memory. The chain descriptors contains the information about source addresses, destination addresses, number of bytes to transfer and some control information. Multiple source addresses can be specified by a chain descriptor. The chain descriptor also contains a *next chain descriptor address* field enabling multiple chain descriptors to be used for one operation.

The XOR engine consists of a boolean unit, a data queue, a packing and unpacking unit, a bus interface unit and some control registers, as shown in Figure 2.10. The boolean unit performs the XOR operation. The data queue is used for holding the result temporarily. The packing/unpacking unit enables data transfers from and to unaligned addresses in the local memory. The bus interface unit provides a low-latency connection with the internal bus. The control registers are used to control the XOR engine operation, including the status register, the address register etc. The XOR operation follows a seven-step operation as follows.

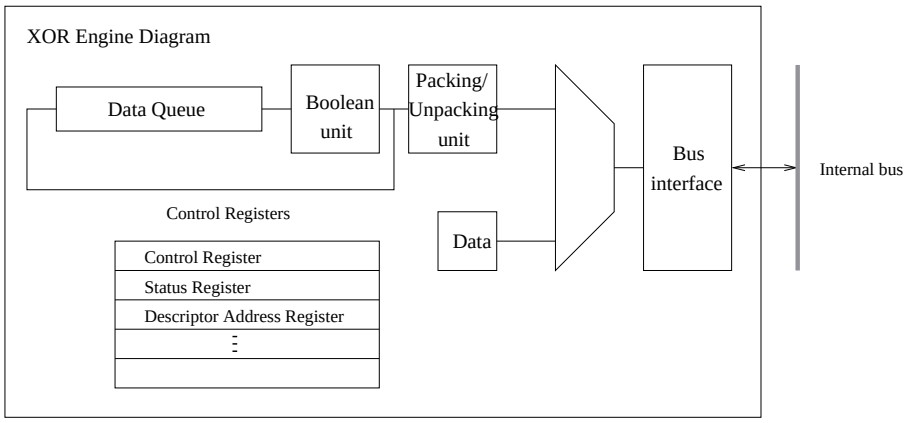


Figure 2.10: **XOR engine diagram**(figure from [48]).

1. The system software write the address of the chain descriptor to the XOR engine address register.

2. The XOR engine reads the chain descriptor at the address contained in the address register.

3. The XOR engine initiates the data transfer from the address pointed to by the first source address. The number of bytes transferred each time is 1 KBytes.

4. The boolean unit performs the XOR operation on the data currently existing in the store queue with the data being transferred from the memory.

5. The XOR engine initiates the data transfer from the second source address and XORs this data with the existing data.

6. The above steps are repeated until the first 1 KBytes of all source addresses are XORed and stored in the data queue. The XOR engine transfers the XOR result to the first 1 KBytes of the destination address.

7. The above operations are repeated until the data specified by the length register are all XORed and transferred back to memory.

2.2.2.4 Write Data Mirroring Between RAID Controllers

As introduced in section 1.1, usually there are two or more RAID controllers in one RAID system. To prevent data loss due to RAID controller failure, the write data received in the RAID controller is copied to the other controller before its acknowledgement is sent to the client. This process is called “write data mirroring”. Unlike the

cache coherency protocol in a distributed shared memory system, write data mirroring of RAID systems focuses on preventing single point failures. Under different caching policies, different write mirroring protocols will be applied.

2.2.2.4.1 Write Data Mirroring under Write-back Policy The advantage of the write-back policy is that the controller can acknowledge the completion of a write command before sending data to the disks. In order to prevent data loss due to a controller failure, before acknowledging the data as complete, the data are placed in two places in battery backed memory, one in each controller. Once it is confirmed that the system cannot lose data, the controller will acknowledge the command as complete. This is the so-called “write data mirroring process”. Assume R1 is the controller that receives write data from the host and R2 is the other controller. The write mirroring protocol for a write-back cache policy works as follows:

1. R1 receives data from a host, sends data to R2.
2. R2 receives data from R1; places it in memory and sends an acknowledgement to R1.
3. Once R1 gets the acknowledge from R2, R1 is able to inform the host that the volume-level command is complete.
4. R1 has all the read stripe units for one stripe that are used to compute the parity; R1 computes the parity unit and sends it to R2.
5. R2 gets the parity unit, puts it in cache and sends an acknowledgement to R1.
6. R1 gets the acknowledgement and is free to send write data to the disks.
7. Once all writes are complete, R1 sends a message to R2 to inform it that the volume command has completed.
8. R2 receives the completion message and sends an acknowledgement to R1.
9. R1 gets the acknowledgement; the command is complete.

2.2.2.4.2 Write Data Mirroring under Write-through Policy Under the write-through cache policy, the RAID controller acknowledge the completion of a write command after it receives the response from the disks. The data that need to be copied to the other controller are the disk-level write commands. Assume R1 is the controller that receives

write data from the host and R2 is the other controller. The write mirroring protocol for a write-through cache policy works as follows:

1. R1 receives write data from the host, decomposes the command into a sequence of reads and writes and forms the required read operations.

2. R1 receives all the read operations, generates the disk write commands and sends them over to R2.

3. R2 receives the disk level write commands from R1, stores them in cache and sends an acknowledgement to R1.

4. R1 the receives the acknowledgement, then starts sending disk-level write commands to the disks.

5. R1 receives responses from all the disks indicating that their write commands are finished, sends a command-completed message to R2 and acknowledges the completion of the volume level command to the host.

2.2.3 Back-end Networks

Enterprise class RAIDs typically use the Fibre Channel Arbitrated Loop (FC-AL) protocol [50] to move SCSI commands between disks and RAID controllers. The basic unit of an FC-AL communication is a 40-bit long FC word which is 8B/10B encoded and represents four 8-bit bytes [50]. The payload data is broken into frames with a known structure. Each frame contains up to 2048 bytes of payload data with a frame header and footer, each of pre-determined length. Each loop allows up to 127 attached devices, including one fabric element or 126 devices if there is no fabric element. In a storage sub-system, as each of the two controllers takes one port, at most 124 disks can be connected to the loop.

2.2.3.1 Topology

Initially, disks were organised in a daisy-chain style loop, known as Just a Bunch Of Disks (JBOD), as shown in Figure 2.11 (a). This architecture suffers from a single point of failure, as failure of any disk or link in the system results in the overall system being unavailable. To overcome this, a hub containing port-bypass and management functionality can be placed in the system. This architecture is commonly referred to as a Managed Bunch of Disks (MBOD), as shown in Figure 2.11 (b). This results in a

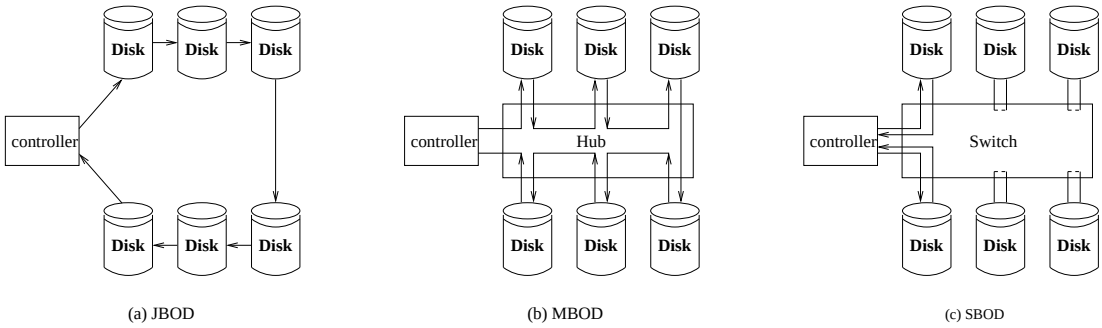


Figure 2.11: **JBOD, MBOD and SBOD architectures.**

system that is tolerant to failures at the expense of an additional delay introduced by the hub. In systems containing many devices, there is already a large delay as communications travel around the loop. To overcome these delays, a Switched Bunch Of Disks (SBOD) configuration was recently introduced [30], as shown in Figure 2.11 (c). In SBOD, a simple cross-bar switch replaces the port-bypass function of the hub. When two devices wish to communicate, the switch is used to create a virtual loop, just consisting of these devices with the switch as an interface. With such a system, the observed loop delay no longer scales with the number of devices in the loop but remains constant.

2.2.3.2 SBOD Operations

While the operation of the MBOD is fully defined by the FC-AL standard, the operation of the SBOD is not so intuitive, although it is designed to be fully interoperable with devices using the FC-AL standard. As opposed to the MBOD, the SBOD switch design considered in this dissertation is not transparent to the Fibre Channel protocol and plays an active part in the FC-AL arbitration process.

When a disk/controller wishes to arbitrate, it transmits ARB (arbitration) words to the SBOD switch. After a short delay, the switch returns these ARB words to the arbitrating disk/controller, thereby allowing that device to gain tenancy of the loop and to transmit an OPEN (open) word. This disk/controller is known as the Initiator. When the switch receives an OPEN word, it first determines whether the port of the target controller/disk is busy or not. If the target port is busy the switch is able to transmit a CLS (close) to the initiator to terminate the connection. If the target port is not busy, the switch will connect the initiator port and the target port together and forward the OPEN to the target device. This forms a virtual FC-AL with two devices in the loop, with a delay in the transmission path reduced to the latency across the switch.

2.2.4 Shelf-enclosures

Most RAID systems are rack-mounted, and the rack is called a shelf-enclosure. Figure 2.12 shows the diagram of a shelf-enclosure. It provides backplanes, power supplies, cooling system and other electronics for the disks and RAID controllers mounted in it. Note that the RAID controllers are optional. For small storage systems, the RAID controllers are also embedded in the shelf-enclosure, while for large storage systems, the RAID controllers are usually separated. In order to improve reliability, in a shelf-enclosure the non-disk components, such as power supplies and fans, have n+1 redundancy. For example, if the shelf needs 1.5 KW power and supplies are 1.5 KW, there are 2 supplies (1 for normal running plus 1 spare). If the supplies are 750W, there are 3 of them (2 for normal running plus one spare). The same principle applies to fans and any processors. However, even though redundancy is provided in shelf-enclosure, it is still possible that the shelf-enclosure fails because of a broken cable, power outage, *etc*. When a shelf-enclosure fails, all the disks mounted in it appear missing to the RAID controller.

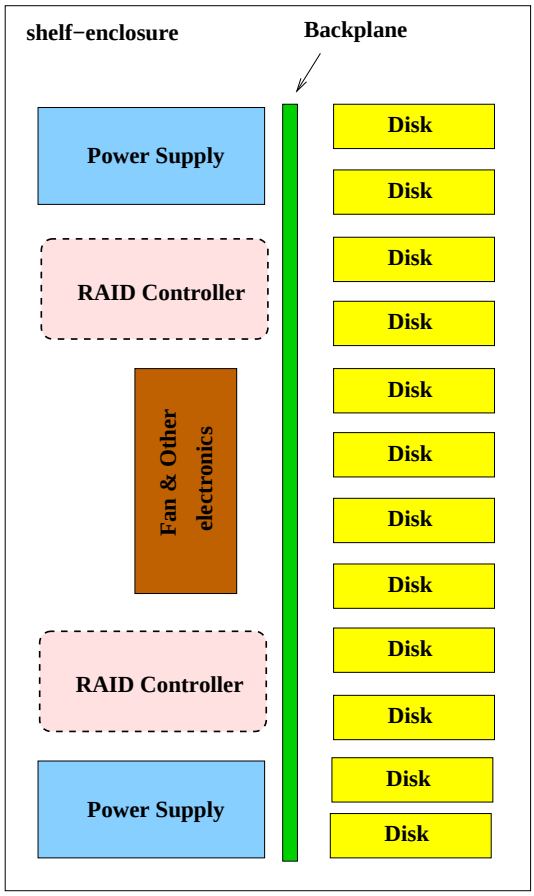


Figure 2.12: **Diagram of a shelf-enclosure.**

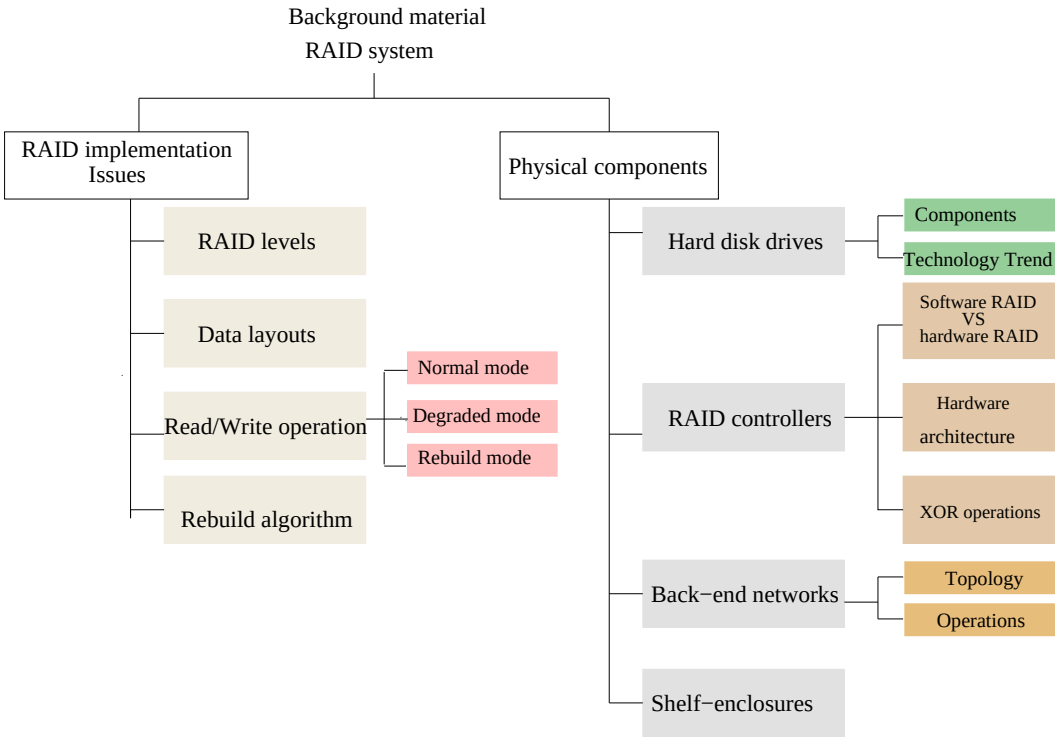


Figure 2.13: **Background Summary.**

2.3 Summary

This chapter has described the background material related to RAID systems. Figure 2.13 shows a chart summarising the contents presented.

The RAID system implementation has a significant effect on system performance. The issues involved in selecting RAID levels, read/write operations, data layout, and reconstruction methods have been discussed. Firstly, seven basic RAID levels were introduced. Among these seven levels, RAID 1 and RAID 5 are the most popular two levels. RAID 1 has better performance than RAID 5 but has lower storage efficiency. Because RAID 6 can protect against at least two disk failures at one time, it is expected to be used in large scale storage systems. Secondly, read/write operations in the three operation modes (normal mode, degraded mode and rebuild mode) were described. Thirdly, data layouts on disks were discussed. Because of its advantage in rebuilding mode, a declustered data layout is preferred. Lastly, reconstruction algorithms were discussed.

RAID systems consist of three basic physical components: hard disk drives, RAID controllers and back-end networks. Hard disk drives have been used as the non-volatile storage for a long time. A disk access operation includes a seek, a rotation and a trans-

fer operation. Over the years, disk capacity has increased dramatically, but there has been only a small improvement in disk access time. Compared with the processor and memory latency, there is a big performance gap between disk and memory. Although new technologies have been proposed to replace hard disk drives, because of their high price, hard disk drives are still the first choice for large capacity storage systems. The RAID controller makes the RAID system look like a single large disk and provides a single address space to hosts. RAID controllers can be implemented in software or in hardware. Hardware RAIDs cost more but have better performance than software RAIDS. High-end hardware RAIDs are usually implemented using the ROC technology. A dedicated XOR engine is included to accelerate the XOR operation. The back-end network connects the disks with the RAID controller. Enterprise RAID systems usually use FC-AL to move SCSI commands between controllers and disks. To meet the increasing performance requirement of data intensive applications, the topology of the back-end networks has evolved from JBOD and MBOD to SBOD. Disk Arrays are usually mounted in shelf-enclosures, which provide backplane, power supplies and a cooling system for the disks.

The background material introduced in this chapter will later be applied to the simulation model design in Chapter 4.

Chapter 3

Related Work

This chapter presents a survey of research work related to the study of the scalability of RAID systems. Due to their high popularity, there is a vast body of work on RAID systems. These studies roughly fall into four categories: system performance modelling and evaluation, system reliability, data layout design and system performance optimisation. Among them, system performance modelling, system reliability studies, and data layout design are related to the work presented in this dissertation.

This chapter is organised as follows: Section 3.1 gives a summary of the work that has been done on modelling and evaluating storage system performance, including studies of the overall performance of RAID systems and of the performance of FC networks; Section 3.2 describes work on RAID system reliability, including work to analyse system reliability and work to improve system reliability; Section 3.3 reviews previous parity declustering data layouts; Section 3.4 reviews distributed hot sparing schemes; Section 3.5 presents a summary of previous storage system simulators; Section 3.6 summarises all the related work and identifies research gaps.

3.1 Performance Modelling and Evaluation

This section reviews prior work on modelling and evaluating storage system performance, including studies of RAID system overall performance and that on FC network performance. This section focuses on performance analysis work while simulation work is reviewed in Section 3.5.

3.1.1 RAID System Performance

Since the invention of RAID systems, performance modelling of these systems has received a lot of attention. A survey of relevant papers reveals a progression of increasing complexity, as researchers attempt to model more features of real systems.

Early performance modelling work focused on parallel disks and ignored the array cache. Kim [51] derived the response time equation for synchronous byte-interleaved disk arrays by treating the entire disk array as an M/G/1 queuing system, *i.e.* the entire disk array is modelled as an open queuing system with an exponential interarrival distribution and a single server consisting of all the disks in the disk array. Kim & Tantawi [52] then presented an analytic method for approximating the disk service time of requests striped across n asynchronous disks (disks rotate independently of one another) and derived a service time equation. In this study, the disk seek time was assumed to have an exponential distribution and the rotation latency was assumed to be distributed uniformly. In these early papers, redundancy and queueing of outstanding requests were not considered.

Chen & Towsley [22, 23] subsequently incorporated both redundancy and queueing into their performance model of RAID 5 disk arrays in normal operation mode. In their study, they modelled RAID 5 performance using queuing analysis and determined the mean overall, mean read and mean write I/O request response times. They considered write synchronization, the effect of different request size distributions, and disk access skewing. Merchant & Yu [70, 107, 71] then analysed RAID 5 and RAID 1 disk arrays in both normal and rebuilding modes. Thomasian & Menon [96, 97] and Kuratti & Sanders [54] then extended to normal, degraded, and rebuilding modes in their RAID 5 performance model. Bachmat & Schindler [13] analysed reconstruction in RAID 1 disk arrays. Lee & Katz [56] extended the analysis of disk arrays to include synchronous I/O workloads.

After the fast cache architecture was proposed [67], research interests shifted to studying the effects of caching policies [66]. Menon [66] models explicit read-ahead and write-back by the array cache of a RAID 5 disk array. Uysal *et al.* [99] present an analytic model that predicts the mean throughput of disk arrays under asynchronous I/O workloads when the mean queue length at the disk array is known. Varki *et al.* [100] developed an analytical model that analyses the effects of caching policies and the effects of parallelism of disks along with the effects of array controller optimisations on the performance of a disk array when read-only and write-only workloads are

submitted to the array.

Although these models have involved increasing levels of complexity, none of them has considered the effect of the back-end network on RAID system performance. This dissertation is concerned with performance modelling, using back-end network bandwidth as the dependent variable.

3.1.2 FC Network Performance

The performance of storage area networks (SANs) has also been studied. Heath *et al.* [37] studied the performance of SANs connected by fibre channel arbitrated loop. Their results show that the fibre channel arbitrated loop is the performance bottleneck for a SAN under a large file transferring workload. However, since the FC network is used here as the front-end network, the controller and SCSI bus also affect network scalability. Moreover, the workloads on the front-end network and the back-end network differ significantly. Therefore, their results cannot be directly used for back-end networks. Ruwart *et al.* [82] focused on the effect of increasing the number of nodes and the physical length of the loop on overall performance. The results also indicated that 10 disks will saturate a 50-m long 1 Gbps FC_AL under file transfer workload. However, this sequential workload does not represent a normal workload and a 50-m loop is very unusual for storage sub-system back-end system. Chevalier *et al.* also [24] studied the back-end network saturation point under sequential accesses, but again, unlike SPC-1, this does not represent a 'normal' workload. In the client/server research field, Son & Kim [92] suggest that the approach to matching the number of clients to servers in a network is a matter of watching and learning. Their work introduces a way to do this matching in a fully switched network, but the RAID back-end network is not a fully switched network as the number of clients (RAID processors for our purposes) is smaller than the number of servers (disks). Therefore the problem addressed is distinct from that addressed in this dissertation.

3.2 RAID System Reliability Study

There are generally two categories of RAID system reliability study: analysis of system reliability and work to improve this reliability.

3.2.1 RAID System Reliability Analysis

Work on analysing RAID system reliability includes studies of disk failure characteristics, other system components failure characteristics and overall system reliability.

3.2.1.1 Disk Failure Characteristic Studies

Most RAID system reliability analysis work focuses on analysing disk failure characteristic. These work including both vendor studies and user experience studies. Seagate & Quantum studied long-term reliability characteristic through accelerated life tests of small populations and collecting statistics from their return unit database [34, 106]. Based on such test, they estimated the MTTF of disks to be more than one million hours. However, the disk reliability experienced by users is totally different. Pinheiro *et al.* [78] studied the failure trends in large disk populations by analysing disk replacement logs and they found that disks are replaced 2-4 times more frequently than the time specified by vendors. Schroeder *et al.* [86] found that the time between disk replacements in the same machine room does not follow the exponential distribution and exhibits significant levels of correlation.

Some research analyses the characteristics of disk latent sector errors (*i.e.* errors that go undetected until the corresponding disk sectors are accessed), which can potentially lead to complete disk failures. Bairavasundaram *et al.* [14] analysed data collected from production storage systems over 32 months across 1.53 million disks (both near-line and enterprise class). They found that disk size, disk age and disk vendor are factors that impact latent sector errors. They also found the there is a high correlation between latent sector errors, *i.e.* once a disk drive develops a latent sector error, it is very likely to develop a second one.

3.2.1.2 Other Component Failure Studies

Compared with research on disk failure characteristics, less work has been published on failure characteristics of other components. Schulze *et al.* [87] presented a reliability analysis on disk arrays. This study showed that besides disks, the rest of the system components, such as the host bus adapters, power supplies and fans cannot be ignored. However, their study was not based on real-world data but based on formula and datasheet specified MTTFs of each components. Jiang *et al.* [49] did a comprehensive study of the failure characteristics of components of RAID systems based on data collected from 39,000 commercially deployed storage systems. These systems consist of

about 1,800,000 disks hosted in 155,000 shelf enclosures. They found that in addition to disk failures that contribute to 20 – 55% of storage subsystem failures, other components such as physical interconnects (broken wires, shelf enclosure power outages, HBA failures, *etc.*) and protocol stacks (software bugs and compatibility issues) also account for significant percentages of storage subsystem failures. Their research indicates between 27-68 % of storage subsystem failures come from physical interconnects. Between 5-10 % are a result of protocol stack errors. Moreover, each individual storage subsystem failure type and storage subsystem failure as a whole exhibit strong self-correlation. In addition, these failures exhibit "bursty" patterns.

3.2.1.3 System Reliability Modelling

The overall reliability of RAID systems is most frequently modelled using continuous-time Markov chains. The failure and recovery of components in the system cause transitions from one state to another. Gibson [33] presented an analytical model for the MTTF of disk arrays and analysed the reliability of several alternative RAID designs.

3.2.2 Improving System Reliability

As described in Section 2.1.1.1, RAID 5 is the most commonly deployed RAID level. However, RAID 5 can only protect the system from one disk failure. If the system encounts a medium error during the reconstruction of the failed disk, the system will not able to recover that stripe. Therefore, much research has been carried out to design efficient RAID 6 code so that the system can survive two disk failures at the same time. These codes include Reed-Solomon (P+Q) erasure codes [79], EVENODD [15], RDP [26], Liberation codes [80], RM2 code [76] and X code [105]. Among these codes, Reed-Solomon (P+Q) erasure codes, EVENODD, RDP, and Liberation codes are horizontal codes, which store parity on separate disks from the data (not accounting for parity rotations). RM 2 and X code are vertical codes, which store the parity on the same disk as the user data.

Reed-Solomon (RS) code, also known as P+Q, is a very popular error control code. It has been used in a variety of applications, including communication, distributed computing and RAID systems. When it is deployed to protect against multiple disk failures in RAID systems, it is always optimal in terms of the number of redundancy disks. That is, it only uses m redundancy disks to protect against the failure of m disks. However, computing the RS code is time consuming as it involves Galois Field

[81] multiplications. EVENODD, proposed by Blaum *et al.* in 1994, is the first double disk failure protection scheme using only simple XOR operations. Thanks to this factor, EVENODD can be implemented on standard RAID level 5 controllers without any hardware changes. In addition, EVENODD employs only two redundancy disks. It is hence the optimal redundancy storage for protecting against double disk failures according to the theory of Singleton bound [62]. Row-Diagonal Parity (RDP) was proposed by Corbett *et al.* in 2004 to protect against double disk failures. Like EVENODD, it stores all data unencoded, and uses only XOR operations to compute parity and requires only two redundant disks. In addition, compared with EVENODD, it is more efficient in terms of computation complexity. Liberation codes, recently proposed by Plank, are a new class of RAID 6 codes. These codes encode, update and decode either optimally or close to optimally. Their modification overhead is lower than all other RAID 6 codes, and their encoding performance is often better as well.

Having been proposed by C. Park in 1995, RM2 [76] transforms the double disk failure protection problem into the data/parity placement problem. Given the number of disks N and the redundancy rate p, the question is how to place data and parity evenly across disks in order to tolerate double disk failures. The X-Code [105] proposed by Xu & Bruck is another vertical code. X-Codes depend on a prime p. The number of disks is p and the number of rows is p as well. Of these rows, two are parity units and $p-2$ are data units. The two rows of parity units are computed from diagonals (one up, one down) through the stripe.

These RAID 6 codes provide a variety of choices for implementing RAID systems that can survive two disk failures. However, they still cannot solve the data service loss problem caused by a physical interconnect failure.

3.3 Parity Declustering Data Layout Design

The basic idea of parity declustering is to set the number of disks to be larger than the stripe width and distribute the stripes over all disks. By so doing, the workload to the failed disk in degraded mode only needs to access part of the disk array, therefore, it has a better degraded performance than RAID 5. In addition, the reconstruction workload of the failed disk just needs to access part of the surviving disk set. Thus the reconstruction workload on the surviving disks is reduced, resulting in a higher rate of user workload servicing during reconstruction. In addition, setting the stripe width less than the overall number of disks allows a higher reconstruction workload and par-

allel reconstruction of different stripes, leading to a shorter reconstruction time. The fraction of each surviving disk that must be read during the reconstruction of a failed disk is called the *parity declustering ratio*. Because of its advantage in degraded and rebuilding mode, parity declustering data layouts have been the subject of extensive research. This section reviews and compares five parity declustering approaches: BCBD, BIBD, PRIME(and RELPR), PDDL and string parity declustering data layout.

To facilitate the discussion, the variables listed in Table 3.1 are defined.

Variable	Definition
n	Total number of disks
k	stripe width, namely the number of disks per stripe
f	number of redundant stripe-units per stripe
m	number of data disks per stripe, $m = k - f$
a	logic unit number of client data unit
α	Declustering ratio, $(k-f)/(n-f)$

Table 3.1: Variable Definitions

3.3.1 BIBD and BCBD

Using Balanced Incomplete Block Design (BIBD) to solve the parity declustering layout problem was suggested by Muntz & Lui [73], but they did not give any implementation examples. Holland & Gibson first demonstrated how to apply BIBD as a way of implementing parity declustered data layouts [41].

BIBD is a term in combinatorial mathematics. Hall [35] describes a BIBD as *an arrangement of n distinct objects into b tuples such that each tuple contains exactly k distinct objects, each object occurs in exactly r different tuples, and every pair of distinct objects occurs together in exactly* λ *tuples.*

As opposed to BIBD, Balanced Complete Block Design (BCBD) contains all combinations of exactly k distinct elements selected from the set of n objects. It is apparent that the result of a BIBD layout is a subset of BCBD. A BIBD can be represented by an $n \times b$ incidence matrix, which contains only 0's and 1's. The rows are labeled with the objects of the design and the columns with the blocks. The (i,j)-th cell is 1 if object i is contained in block j and 0 otherwise. Each row of the incidence matrix has r 1's, each column has k 1's and each pair of distinct rows has λ common 1's. These observations

lead to a useful matrix identity. If A is the incidence matrix of a (n,b,r,k,λ)-design, then $AA^T = (r-\lambda)I + \lambda J$ where I is the $v \times v$ identity matrix and J is the $v \times v$ matrix of all 1's. A BIBD can be found by using this matrix identity,

BIBD and BCBD algorithms are applied to the data layout problem by associating disks with objects and associating stripes with tuples. Figure 3.1 illustrates the procedure of using a BIBD table to construct a data layout with $n = 5$ and $k = 4$, namely the number of disks is 5 and the stripe width is 4. Firstly, each stripe is associated with a tuple in the BIBD table by considering each object as a disk. For the purpose of simplicity, the i-th stripe is associated with tuple i. Thus, the disks are allocated to each stripe. Then, allocate the first available stripe unit (*i.e.* one that has not been allocated to any stripe) on the disk to the corresponding stripe. According to this rule, stripe 0 is associated with tuple 0, which contains objects 0, 1, 2 and 3. Taking the first available stripe unit on each disk, stripe unit 0 on disks 0, 1, 2 and 3 are allocated to stripe 0. The selection of the parity unit is arbitrary at this stage. In this example, the last element of each tuple is chosen as the parity disk. Therefore, stripe unit 0 on disk 3 is allocated as the parity unit of stripe 0, whereas stripe unit 0 on disk 0, 1 and 2 are allocated as the data units of stripe 0. Following this rule, stripe unit 1 on disk 0, 1, 2 are allocated as the data unit of stripe 1 and stripe unit 0 on disk 4 is allocated as the parity unit of stripe 1. This procedure is repeated until stripes 2, 3 and 4 are all placed on the disks, as shown in the first quarter of Figure 3.1. These stripes will have a distributed rebuilding workload over all disks when one disk fails. Such an iteration, which contains the minimum number of stripes that form an evenly balanced parity declustering data layout, is called a *rotation*.

In Holland's paper, in order to meet the distributed parity criterion, this allocation procedure is repeated k times, with the parity disk assigned to different elements of each tuple in each rotation (see the right side of Figure 3.1). The whole data layout in Figure 3.1 is further repeated until all stripe units on each disk are mapped to stripes. One iteration of the data layout in Figure 3.1 is referred to as the *block design table* and the complete cycle (the entire data layout in Figure 3.1) is referred as the *full block design table*.

BIBD meets the first three of the six properties that an ideal data layout should have: single failure correction, distributed parity and distributed reconstruction. Depending on how the client data units are mapped to stripes, they can be made to satisfy either the large write optimisation or the maximum parallelism property. Although compared with BCBD, BIBD reduces the number of combinations, as the disk array

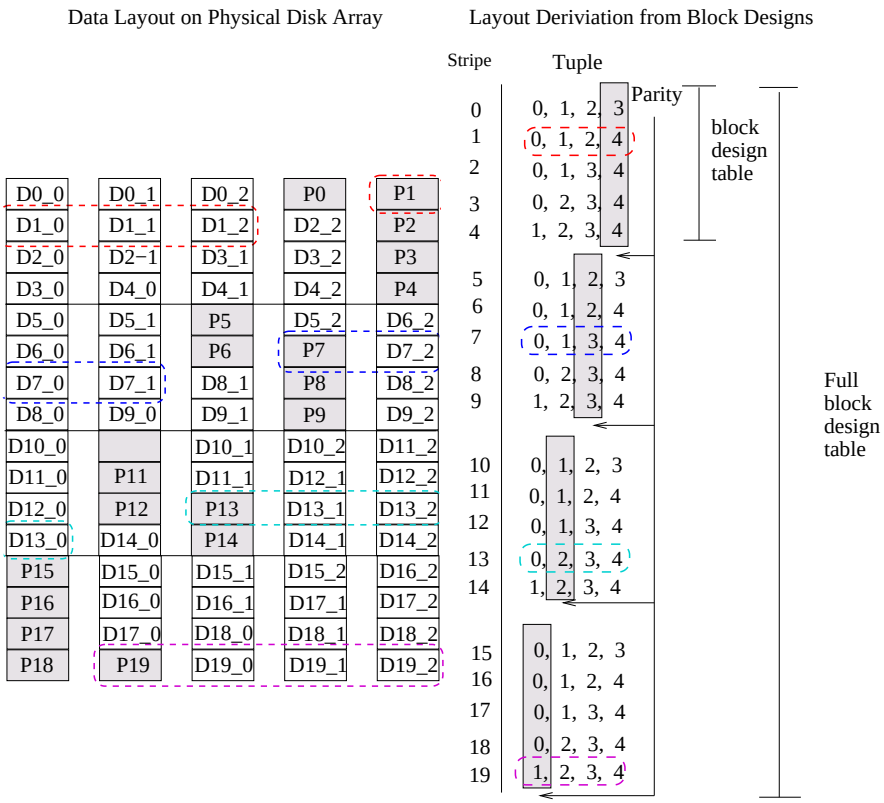


Figure 3.1: Parity Declustering using BIBD table, n = 5, and k = 4. (figure from [41]).

size increases, the block design table is still very large, which means that the RAID controller needs a large amount of memory to store it. Furthermore, BIBD layouts are not available for arbitrary n and k. When n is large, known BIBDs are rare. Therefore, BIBD is not a flexible approach for parity declustering in large-scale systems.

3.3.2 PRIME and RELPR

Schwabe & Sutherland proposed a Ring-based data layout [88], which reduced the computation complexity and the size of the data layout. Based on this ring-based data layout, Alvarez *et al* [11] proposed PRIME and RELPR. Both PRIME and RELPR use a direct computation approach to map user data to a physical location and have only $O(1)$ computation complexity. By slightly relaxing the maximal parallelism property, the PRIME data layout comes very close to being ideal. PRIME requires n to be a prime number; RELPR relaxes this constraint by removing the distributed reconstruction requirement. There are $n(n-1)$ stripes per rotation in PRIME layout and $n*\phi(n)$ [1] stripes per rotation in RELPR data layout. Algorithm 1 describes how to map a user

[1] $\phi(n)$ counts the number of positive integers less than n that are relatively prime to n.

data unit to its physical disk position when n is prime and Algorithm 2 describes the scenario when n is not a prime number. The input of the algorithm is the logical stripe unit number a and the algorithm calculates its physical disk number, the disk offset and those of the corresponding parity unit.

Algorithm 1 PRIME

$z = \lfloor \frac{a}{mn} \rfloor$;
$y = (z \mod (n-1)) + 1$;
$disk(a) = ay \mod n$;
$offset(a) = \lfloor \frac{a}{n} \rfloor + fz$;
$parity-disk_i(a) = (((\lfloor \frac{a}{m} \rfloor + 1)m + i) * y) \mod n$;
$parity-disk_i(a) = (z+1)k - f + i$;

Algorithm 2 RELPR

Require: $\phi(n)$ counts the number of positive integers less than n that are relatively prime to n;
Require: $Y = \{y_0, y_1, ..., y_{\phi(n)-1} | 0 < y_l < n \wedge gcd(y_l, n) = 1 \ for \ 0 \leq l < \phi(n)\}$;
Require: $g = gcd(n, m)$;
$z = \lfloor \frac{a}{mn} \rfloor$;
$l = z \mod \phi(n)$;
$j = \lfloor \frac{a}{m * (n/g)} \rfloor \mod g$;
$disk(a) = (a + j)y_l \mod n$;
$offset(a) = \lfloor \frac{a}{n} \rfloor + fz$;
$parity-disk_i(a) = (((\lfloor \frac{a}{m} \rfloor + 1)m + i + j) * y) \mod n$;
$parity-disk_i(a) = (z+1)k - f + i$;

Figure 3.2 shows an example of PRIME with n=5, k=4 and f=2. Figure 3.3 shows an example of RELPR with n=6, k=3 and f=1.

3.3.3 Nearly Random Permutation Based Mapping

Merchant & Yu proposed a parity declustering data layout based on a nearly random permutation [71]. This data layout is created by randomising the mappings for each stripe. The mapping created for each stripe is based on an almost random permutation generated by shuffling the identity permutation. Because of the randomness of generating the data layout, finding the address of a logic block needs the position of

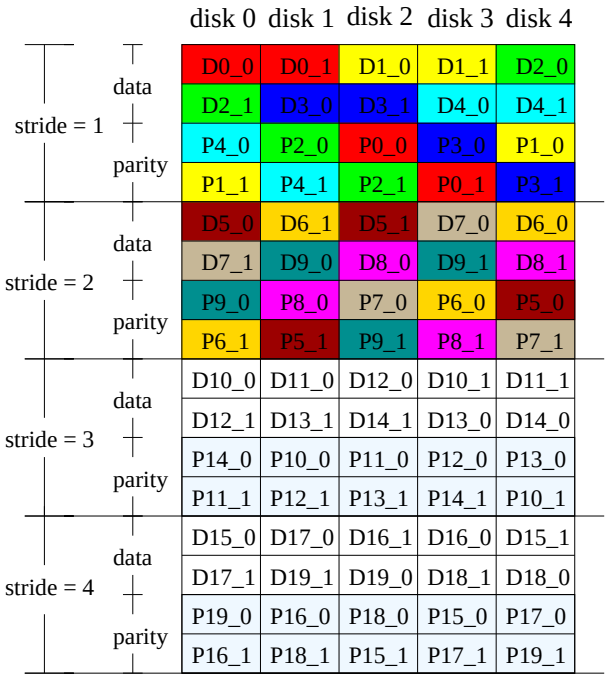


Figure 3.2: PRIME **data layouts.** n=5; k=4; f=2 (figure from [11]).

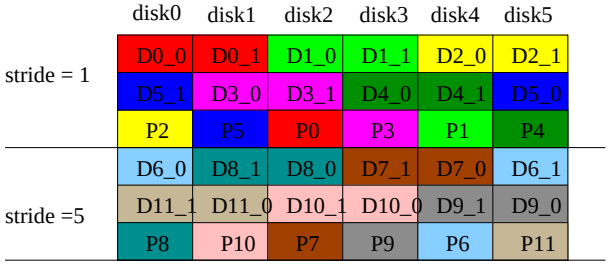


Figure 3.3: RELPR **data layouts.** n=6; k=3; f=1, $\phi(n)$ =2 (figure from [11]).

each block to be traced repeatedly after each shuffle. The time to find the address is $O(\log B + \log n)$, where there are B blocks per disk and n disks in the cluster. Although this data layout is applicable to any combination of n and k, it is not a run-time efficient scheme when n is large.

3.3.4 Permutation Development Data Layout

Schwarz *et al.* also proposed a permutation based data layout called Permutation Development Data Layout (PDDL) [89]. PDDL assumes that n disks can be virtually divided into g stripes and a spare disk. Each disk is assigned a virtual disk number between 0 and n. PDDL uses a *base permutation* as an initial mapping for the first row of stripe units. For all other rows, PDDL adds the row number to the disk num-

ber obtained from the base permutation. Figure 3.4 illustrates an example of PDDL mapping two virtual stripes (k =3) and one virtual spare disk to seven physical disks. This method is efficient in both computation complexity and storage, but again, not all configurations can implement PDDL because of the lack of a base permutation.

Virtual Disks

row number	spare	data	data	parity	data	data	parity
0	0	1	2	3	4	5	6
1	0	1	2	3	4	5	6
2	0	1	2	3	4	5	6
3	0	1	2	3	4	5	6
4	0	1	2	3	4	5	6
5	0	1	2	3	4	5	6
6	0	1	2	3	4	5	6
7	0	1	2	3	4	5	6
8	0	1	2	3	4	5	6
9	0	1	2	3	4	5	6
10	0	1	2	3	4	5	6

permutation base = {0, 1, 2, 4, 3, 6, 5},

physical disk number =

(permutation base[virtual disk number]+ row number)% 7

Physical Disks

row number							
0	0	1	2	4	3	6	5
1	1	2	3	5	4	0	6
2	2	3	4	6	5	1	0
3	3	4	5	0	6	2	1
4	4	5	6	1	0	3	2
5	5	6	0	2	1	4	3
6	6	0	1	3	2	5	4
7	0	1	2	4	3	6	5
8	1	2	3	5	4	0	6
9	2	3	4	6	5	1	0
10	3	4	5	0	6	2	1

Data Layout

0	S	D0_0	D0_1	D1_0	P0	P1	D1_1
1	D2_0	D2_1	P2	D3_1	D3_0	S	P3
2	D4_1	P4	D5_0	P5	D5_1	D4_0	S
3	P6	D7_0	D7_1	S	P7	D6_1	D6_0
4	D9_0	D9_1	P9	D8_0	S	P8	D8_1
5	D11_1	P11	S	D10_1	D10_0	D11_0	P10
6	P13	S	D12_0	P12	D12_1	D13_1	D13_0
7	S	D14_0	D14_1	D15_0	P14	P15	D15_1
8	D16_0	D16_1	P16	D17_1	D17_0	S	P17
9	D18_1	P18	D19_0	P19	D19_1	D18_0	S
10	P20	D21_0	D21_1	S	P21	D20_1	D20_0

Figure 3.4: **PDDL mapping.** Mapping two virtual stripes and one virtual spare disk to seven physical disks (figure from [89]).

3.3.5 String Parity Declustering Data Layout

Gang *et al.* [31] proposed a layout called *string parity declustering data layout*, which incorporates the advantages of both orthogonal data layout and weighted parity declustering data layout. In this paper, a physical interconnect is called a string. The data layout is first initialised using a random layout. A simulated annealing algorithm [53] is then used to optimise the data layout by setting a balanced reconstruction workload on each disk as the objective. The simulated annealing algorithm proceeds by swapping the positions of two units picked from the array at random. Obviously, the data layout simply generated by a simulated annealing algorithm does not guarantee to tolerate a single physical interconnect failure. In order to survive a physical interconnect failure, some constraints are placed on layout initialisation and transformation to guarantee that no physical interconnect contains more than one unit from one stripe.

Although this string parity declustering data layout is able to survive a physical interconnect failure and also has the advantage of a parity declustering data layout, the approach is random and the way to generate the data layout is time consuming. Moreover, the size of a rotation is random and unknown.

3.4 Distributed Hot Sparing

On-line spare disks, also called hot spares, allow the reconstruction of failed disks to start immediately, reducing the window of vulnerability during which another disk failure would result in data loss. If the spare space is on dedicated disks, these spare disks are idle most of time and do not participate in the normal operation of the system. In order to enhance system performance in normal operation, Memon & Mattson [68] proposed distributed hot sparing, which distributes the capacity of the hot spare disks over the whole disk array, as shown in Figure 3.5. Using distributed hot spares has the following advantages: (1) it allows all disks in the disk array, including the disk that would otherwise have been a dedicated spare, to serve requests, improving system performance; (2) because each disk is partially empty, each disk failure requires less work to reconstruct the data of the failed disk; and (3) it allows reconstruction of several disks at the same time so that the rebuilding speed is not restricted by the write disk bandwidth when the parity declustering ratio is low. The disadvantage of distributed hot sparing is that the reconstructed data must eventually be copied back to a permanent replacement for the failed disk, which creates extra work for the system.

However, since the copying back can be done during system idle time, it will not affect system performance.

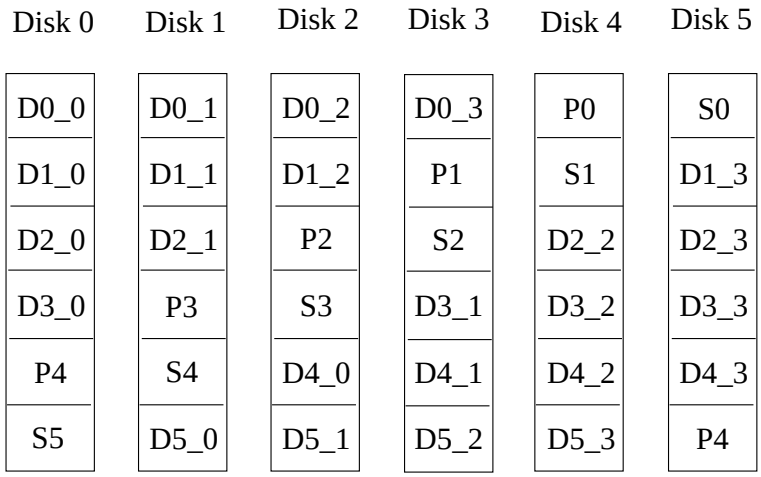


Figure 3.5: **Distributed Hot Sparing.** This figure illustrate a RAID 5 disk array with distributed hot sparing. 'D' denotes a data stripe unit, 'P' denotes a parity stripe unit, and 'S' denotes a spare stripe unit (figure from [68]).

In the data layout of the distributed hot sparing system described above there is a spare stripe unit in each stripe. This scheme works well for a data layout that has a stripe width equal to the number of disks. Unfortunately, this simple solution cannot be directly applied to parity declustering data layouts. Because parity declustered data layouts set the number of disks larger than the stripe width, having a spare unit in each stripe results in a waste of storage space. To solve this problem and simplify the mapping, Holland [41] proposed to allocate spare space in contiguous bands, as shown in Figure 3.6. The allocation of spare space is usually done statically to simplify mapping logic addresses to physical addresses. When a disk failure occurs, the problem is to assign the spare units as replacements for the units on the failed disk. Holland [41] proposed an algorithm to assign distributed hot spare units to the data units on a failed disk. Depending on the particular failed disk, the algorithm first lists the complement disks (namely the disks that are not involved in the stripe that the unit belongs to) for the units on the failed disk. It then assigns a unit from a complement disk to that unit. It is inevitable that there are conflicts during the assignment. This is because part of the complement disks of different units are the same, in the spare unit assignment process, unit that get assigned a spare unit later might find that all the spare unit on its complement disks has been assigned. The algorithm has to resolve this conflict by adjusting previous assignment. It keeps checking for conflicts and adjusting the assignment ac-

cordingly. In a large system, this checking and re-assignment approach could become very complicated.

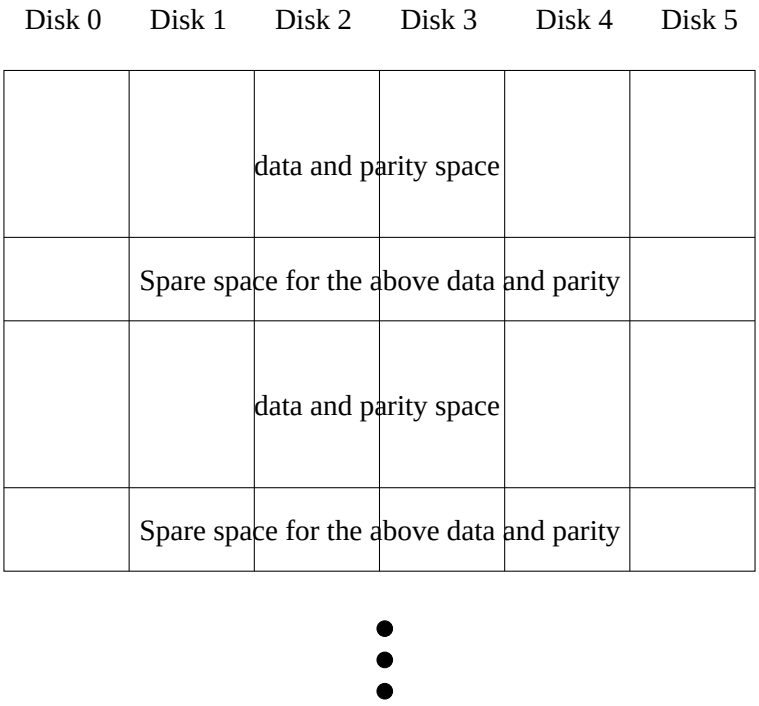


Figure 3.6: **Allocating spare space in contiguous bands**(figure from [41]).

3.5 Simulation of Storage Systems

This section reviews the simulation models build for storage systems, including simulation models of RAID systems and those of FC networks.

3.5.1 Simulation Models of RAID System

There have been a number of simulation tools developed for the purpose of storage system performance evaluation, such as Pantheon [102], Disksim [16], RAIDFrame [28], and DASim [95].

The Pantheon storage system simulator [102] was developed at Hewlett Packard. It is a simulator for performance modelling of parallel disk arrays and parallel computers. It has been used in a number of projects, such as the TickerTAIP parallel RAID architecture [17], the HP AutoRAID advanced disk array technology [103], and AFRAID [85]. Pantheon's simulation modules are written in C++, compiled and linked together

to make a single Pantheon executable program. The storage system components modelled in the Pantheon simulator include a module for controlling experiments, a host, an IO loader, a device driver, a disk channel and a disk drive. Unfortunately, Pantheon is not available for use outside Hewlett Packard for commercial reasons, so it could not be considered for use in the work described in this dissertation.

DiskSim [32] was developed at the parallel data laboratory of Carnegie-Mellon University (CMU). It is an efficient and highly configurable disk system simulator. It includes modules that simulate disks, intermediate controllers, buses, device drivers, request schedulers, disk block caches, and disk array data organisations. The most important feature of DiskSim is that its disk drive module simulates modern disk drives in great details and has been carefully validated against several production disks. However, as DiskSim focuses on disk simulation, it does not provide many interconnection network models, especially the latest developed serial technologies like FC, SAS and SATA.

RAIDframe [28], another storage system simulator developed at the parallel data lab of CMU, is a framework for rapid prototyping and evaluation of redundant disk arrays. Using a graphical programming abstraction and a mechanised execution strategy, RAIDframe allows users to quickly construct working prototypes which can be immediately evaluated in each of three environments: a device driver running against real disks, a user process running against real disks, or an event-driven simulator. Currently the architecture models implemented in RAIDframe include RAID levels 0, 1, 4, 5, 6, parity declustering [41], distributed sparing [42], and interleaved declustering. However, as RAIDframe focuses on the impact of data placement of RAID system on performance, like DiskSim, it does not provide many facilities for users to evaluate the impact of interconnection networks and RAID controllers on performance.

The DASim [95] is a discrete-event simulator written in C++. The purpose of DASim is to study the performance of different RAID implementations, for example RAID 5, EVENODD and RM 2. DASim generates multiple physical requests for logical requests using an execution plan, which is a Directed Acyclic Graph (DAG) determined by the type of logical request: the RAID level, the mode of operation. The disk model of DASim is called DTSim (Disk Timing Simulator) which provides disk timing information specified by DiskSim. Like other RAID simulators, DASim has no means of modelling back-end interconnection networks.

In summary, existing RAID system simulators focus on studying the impact of RAID algorithms and disk access time, therefore, they have limitations on exploring

the impact of other components in modern RAID systems, such as the other functions of the RAID controller and the interconnection network.

3.5.2 Simulation Models of FC network

Publicly available FC network simulation models include SANSim [101] and SimSAN [91].

SANSim [101], developed at the Data Storage Institute of NUS, is an event-driven FC SAN simulation tool. SANSim is aimed at studying the performance of fibre channel storage area network. It consists of four main modules: I/O workload module, host module, storage network module and storage system module. SANsim implements the FC protocol stack at a very low level, which allows the simulator to generate very accurate results. However, this low level simulation also makes SANsim very slow. Therefore, to run a large benchmark such as SPC-1 is very time consuming.

SimSANs [91], standing for *Simulating Storage Area Networks*, is another toolkit for designing, modelling, simulating, and evaluating SANs. It is written in C++ and based on the OMNeT++ [75] discrete event simulation framework. SimSAN has implemented major SAN related protocols, such as SCSI (SAM, SBC, SPC), FCP, and Fibre Channel (FC-FS, FC-LS, FC-GS, FC-SW). Its intended use is to explore I/O path performance limits from the architectural standpoint. However, like SANSim, SimSAN also implements the FC protocol stack in a very detailed way, leading to a very slow simulation speed.

In conclusion neither SANSim nor SimSAN is suitable for evaluating the back-end network of a RAID system that is running a large benchmark workload.

3.6 Summary

This chapter has presented a survey of research work that is related to this dissertation. This work includes storage system performance modelling and evaluation, RAID system reliability studies, RAID system data layout design and simulation of storage systems.

There has been a vast body of research work on modelling and evaluating storage system performance, including those on RAID system performance and those on FC network (SAN) performance. Previous RAID system performance modelling work focused on the performance of parallel disks and the effect of cache policies. None

of them has considered the effect of the back-end network. As the number of disks connected to the system keeps increasing, the effect and scalability of the back-end network cannot be ignored any longer. The performance of FC networks has also been studied. However, the scalability of the back-end network has only been studied under large sequential workloads, the result of which cannot be applied to real world storage system design.

Research on RAID system reliability generally includes two categories: analysis of system reliability and work to improve system reliability. Previous system reliability analyses shows that in addition to disk failures, physical interconnect failures also account for significant percentages of storage subsystem failures. Physical interconnect failures result in multiple disks from the same interconnect appearing to be missing from the system. However, current work on improving reliability is focused on designing efficient RAID 6 codes, which can survive only two disk failures.

Because of their advantage in degraded and rebuilding mode, parity declustering data layouts have been extensively researched. This chapter has reviewed existing parity declustering data layout, including BCBD, BIBD, PRIME(and RELPR), nearly random permutation mapping, PDDL and string parity declustering data layouts. Among these data layouts, the first five cannot survive physical interconnect failures or bursty disk failures. The string parity declustering data layout combines the advantages of both orthogonal and parity declustering data layouts. However, its generation is inefficient and the size of a rotation is random and unknown.

Distributed hot sparing has the advantage of improving system performance. In addition, it helps to eliminate write bottlenecks during disk reconstruction for parity declustering data layouts. The existing distributed hot sparing assignment algorithm for parity declustering data layouts uses re-assignment to solve the conflicts in the assignment process. This approach could become very complicated and inefficient when the system scale is large. Therefore, it is important to design an efficient distributed hot spare assignment algorithm for parity declustering data layouts.

There have been a number of simulation tools developed for the purpose of storage system performance evaluation, such as Pantheon, Disksim, RAIDFrame, and DASim. However, these simulators focus on studying the impact of RAID algorithms and disk access time, therefore, they have limitations when exploring the impact of other components in modern RAID systems, such as the other functions of the RAID controller and the interconnection network. FC network simulators, SANsim and simSAN, have been designed to investigate the performance of storage area networks (SAN), but they

have operated at a very detailed level of simulation with a very large ratio of real time to simulation time, which make simulation under real benchmarks impractical. In conclusion, there is a lack of suitable simulation tools for modern RAID system design.

Chapter 4

The SIMRAID Simulation Model

This chapter presents the design and implementation of the RAID system simulation model SIMRAID. In general, there are three different ways to evaluate performance: analytical modelling, empirical measurement, and simulation. The stochastic nature of realistic traffic streams and the complex interactions between commands coupled with the presence of multiple disks, large caches and sophisticated RAID controller designs, means that analytical modeling is insufficient to evaluate RAID system performance [100]. Empirical measurement ensures correctness and completeness of results by running benchmarks on an actual system; however, building the experimental framework in the real world is very expensive. Thus, simulation approaches are chosen as the main research method in this dissertation. Much of the contents of this chapter has been published in [57] and [27].

The remainder of this chapter is organised as follows: firstly, the simulation model development environment, the Hierarchical computer Architecture design and Simulation Environment, HASE, is introduced, including its facilities and capabilities to model RAID systems; secondly, the design and implementation of SIMRAID are presented; thirdly, the simulation process of the model is described; fourthly, the validation of SIMRAID is presented; and lastly, the performance of the model is discussed.

4.1 HASE

HASE is an integrated simulation development environment that allows for the rapid model development and exploration of computer architectures at multiple levels of abstraction [25]. There are three reasons why HASE was chosen as the model construction framework for this PhD project. Firstly, it provides a full framework to support

modelling of computer systems, including both an underlying Discrete Event Simulation (DES) engine and a graphical user interface (GUI). This framework facilitates the processes of creating, debugging, and running discrete-event simulation models. Secondly, the programming language of HASE is based on C++ (called Hase++, which is a superset of C++ with predefined functions describing the basic operations common to most simulation models). This C++ based programming language allows both low-level programming as well as object-oriented design. Lastly, it has been successfully used in a number of research studies including those by Alam [10] and Marurngsith [65], which confirms its capability for simulating computer systems at different levels of abstractions.

This section introduces the facilities and capabilities of HASE for modelling storage systems. Firstly, an overview of HASE is provided. Following that, HASE entity and hierarchical design, the HASE DES engine, HASE event scheduling and time advancing, and HASE clock mechanisms are introduced in detail.

4.1.1 Overview of HASE System

HASE consists of two levels of software: the underlying DES engine and the GUI. The underlying DES engine provides functions for inter-process communication, event scheduling and simulation synchronisation, while the GUI provides functionalities for parsing, editing, displaying and post-mortem animation. Figure 4.1 depicts a screen shot of the HASE application window while loading SIMRAID.

The basic model components of HASE are *entities*. Entities contain a number of ports through which entities are linked. Simulation events are passed from one entity to another through the ports. HASE uses three types of file to specify a simulation: EDL (project.edl), ELF (project.elf) and Hase++ (entity.hase) files. The EDL file describes the components of the architecture, their ports and parameters, the links between them and the hierarchical structure (in essence the logical structure of the experiment). The ELF file contains information relevant to the physical display, *i.e.* where components and ports are positioned on the screen, and any variables to be displayed. When creating the display, HASE uses icons in GIF format to represent the components on screen. The Hase++ files specify the behaviour of each of the entities. Simulation model development in HASE follows a four-step process: (1) model design, (2) construction of a simulation executable, (3) experimental control to set parameters and to run a simulation and (4) tracing and post-mortem animation.

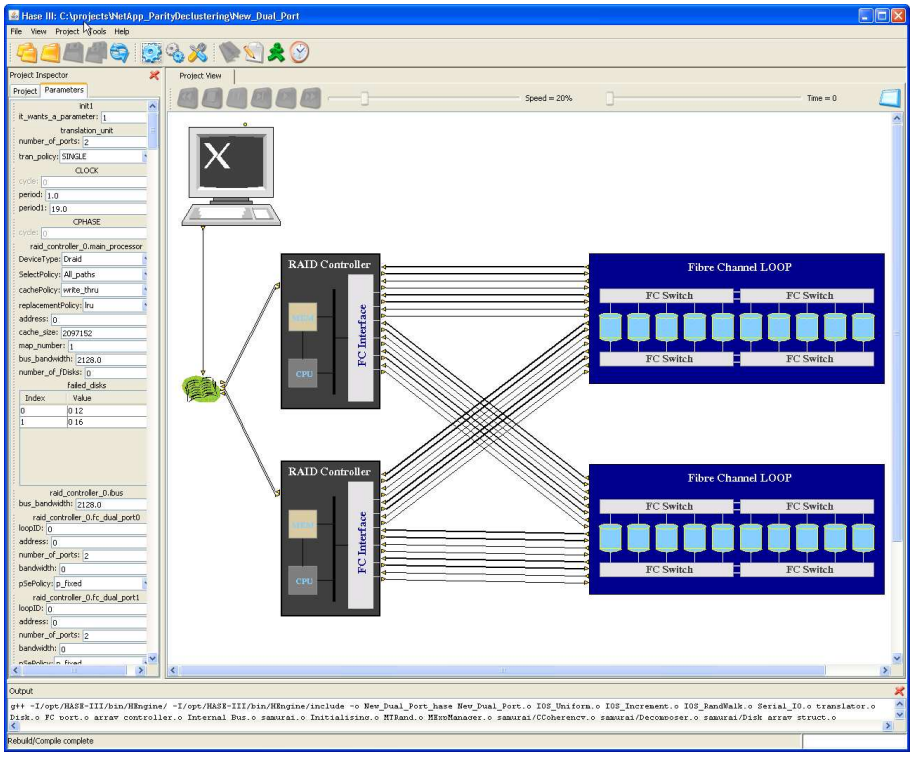


Figure 4.1: **HASE screen shot**

Figure 4.2 shows the software architecture of HASE and the process of creating and running a simulation model. The first step in creating a model in HASE is to define the architecture of the system being modelled (project.edl file), the corresponding displaying file (project.elf file) and the entity behavioural files (.hase files), as shown in Figure 4.2 (a). The model can then be loaded by HASE prior to the simulation being run. During the loading process, the HASE GUI and its embedded parser translate the definition of the model into an intermediate representation and also display on screen a graphical view of the model with its predefined icons (Figure 4.2 (b)). Prior to building the executable file, HASE generates a set of C++ files from the intermediate representation and the entity behaviour files (.hase file), as shown in Figure 4.2 (c). These C++ files are then complied and linked with user object files and HASE library object files to generate the executable simulation file (Figure 4.2 (d)). The executable simulation file acts as a platform for running simulation experiments. The input file is the user's parameter file. The system parameters are configured by updating this parameter file before simulation. Figure 4.2 (e) shows the simulation experiment process. A trace file that records the simulation run profile can be created during the simulation for animation and model verification purpose.

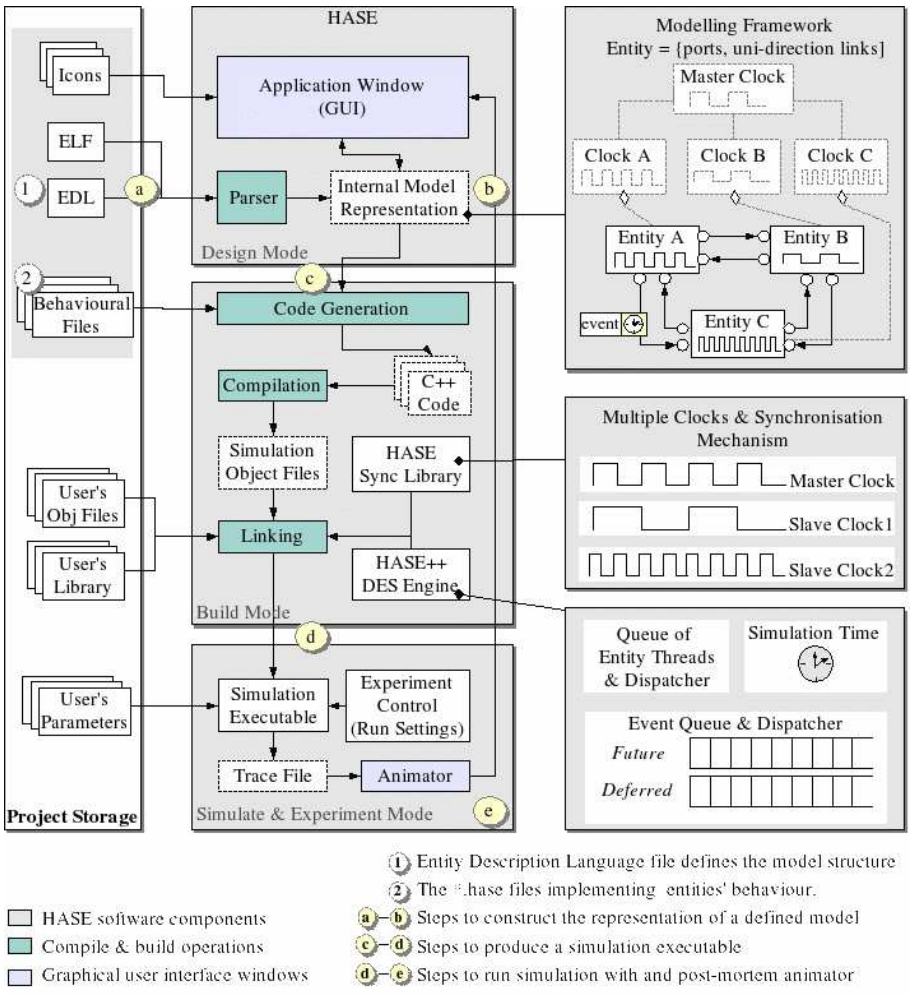


Figure 4.2: **HASE software architecture and model construction process**

1

4.1.2 HASE Entity and Hierarchical Design

The basic components of a HASE simulation model are entities that communicate by passing events via Ports. Each entity is composed of states, parameters and ports. The states and parameters of an entity define its properties at a particular point in time along with its internal attributes. The ports are the routes through which communication events are passed between entities; they are connected with ports of other entities in the STRUCTURE part of the EDL file through uni-directional links. The behaviours of the entities, such as creating, sending and processing events, are defined in the entity_name.hase file. An entity can be defined as a normal discrete event entity or a clocked entity (Section 4.1.5).

In addition to the basic entity, HASE provides for hierarchical model design by using a compound entity (COMPENTITY) or a design template to create a higher level

entity. The COMPENTITY groups together entities and/or other higher level entities (the COMPENTITY's descendants), along with their interconnections, into one entity in a child-parent fashion, allowing the operation of the lower level components to be abstracted out. When a project is first opened, a compound entity is displayed on screen by its own icon. Left clicking on this icon changes the display to show the next level of detail (*i.e.* a descendant may itself be a compound entity). A design template describes functional relationships among entities and also provides the means of connection between them. HASE design templates include BUSENTITY, NETWORKENTITY and MESHnD entity [63].

4.1.3 The HASE++ DES Engine

HASE entities run in separate lightweight threads during simulation. The HASE++ DES engine is the core that creates, manages and synchronises these threads. Its functionality includes generating the lightweight threads for each entity, scheduling events and synchronising simulation time [63]. The DES engine is implemented as a multi-threaded C++ library that provides core components for implementing the simulation logic of a model. All parts of the HASE++ library are written as classes and their facilities are presented as a set of headers. The HASE++ library is highly modular, and independent of the HASE applications (namely the simulation models). This independence of the HASE++ library enables it to be developed without relying on any components of the HASE environment and applications. On the other hand, HASE model developers gain the advantage of rapid model development by using the HASE++ library that is integrated into the model in the building process.

The HASE++ library includes nine basic classes: entities, threads, semaphores, ports, simulation events, event predicates, event queues, the DES system, and a DES Manager. Among these classes, the DES system class manages the creation, registration, scheduling and resource sharing of multiple entity threads, with the functions provided by threads and semaphores classes, while the DES Manager maintains simulation time and manages event queues. The entity and port classes are the base classes for implementing an entity. The simulation event class implements the simulation events passed between entities, while the event predicate provides functions for comparing events. The event queue class manages the queue of simulation events, like adding, inserting and deleting events from the queue.

4.1.4 Event Scheduling and Time Advancing

HASE maintains two events queues: a *future events queue* and a *deferred events queue*. The future queue stores events that contain timestamps referring to the time in the future when these events are expected to occur, whereas the deferred queue stores events that cannot be dispatched at the specified timestamps because the receiving entities are not yet ready to get any events. Events are sorted in both queues in ascending order of their timestamps.

HASE maintains a global simulation time, which represents an absolute or relative moment in time when the simulated events occur. This simulation time is initialised to zero at the start of the simulation. HASE++ uses a next-event time algorithm to advance to the next simulation time. A schematic diagram of HASE events scheduling and time advancing is shown in Figure 4.3. During a simulation run, entities create simulation events with a timestamp which represents the time when the event is supposed to happen. These events are pushed into the future event queue in ascending order of their timestamps. The DES engine pops up the event that has the smallest timestamp from the future event queue. HASE then updates the global simulation time to the timestamp of this event. Therefore the simulation time is advanced to the time specified by the timestamp of the most imminent event in the future queue rather than being progressed on the basis of a fixed time unit.

Following the updating of the simulation time, the scheduled event can either be dispatched to the receiver or inserted into the deferred event queue depending on whether the receiver is ready or not. In a handshake situation, when a blocking send/receive is implied, the receiver is normally ready to receive, therefore, the event is directly dispatched to the receiver (put into its event buffer). In a non-blocking situation, when events can be snooped at any point in time, the receiver will not be ready until a certain point (like next clock phase), so this event will be inserted into the deferred queue. Once the receiver is ready to receive the event, it will be dispatched to the receiver. After receiving an event, an entity will process this event and will keep creating new events. This process continues until either a set of pre-specified stopping conditions is satisfied, or a terminate signal is scheduled from one of the participating entities. This next-event time algorithm effectively saves computational resources by jumping the simulation time from event time to event time and enables the HASE DES engine to efficiently support simulations that use a handshaking communication protocol.

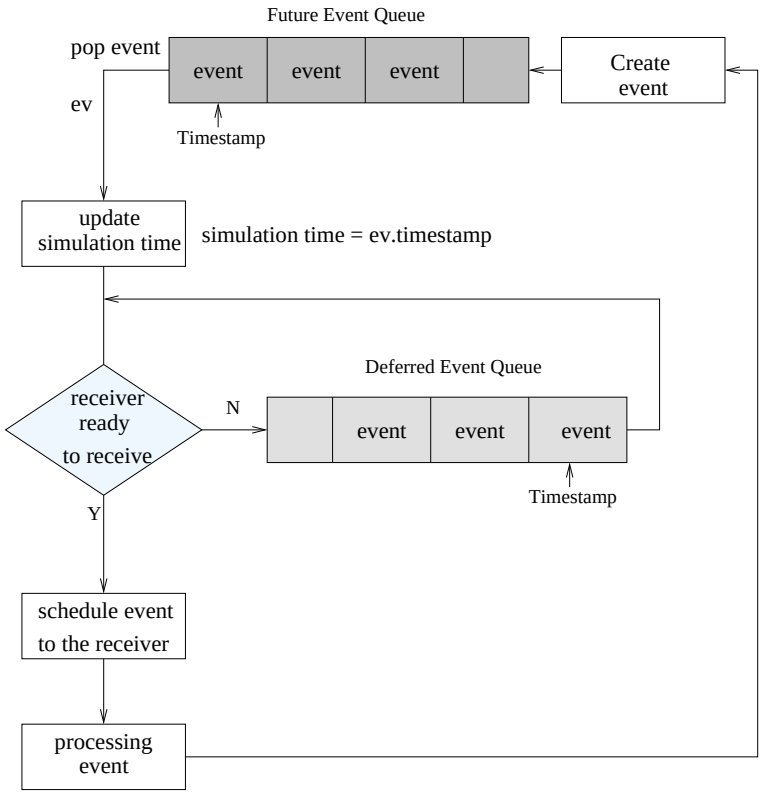


Figure 4.3: **HASE events scheduling**

4.1.5 Extensible Clock Mechanism

As most projects are simulations of synchronous systems, typically computer systems in which entities need to be clocked, HASE provides a number of built-in extensible clock mechanisms [63]. A clock in HASE serves two purposes: (1) to simulate the clock in the real-life version of the system being simulated; and (2) to synchronise the simulation code of the entities against ticks of the clock, in order to ensure an orderly transfer of data between entities.

The clock facilities are presented through the Sync library. The Sync library consists of a *Clock* class and a number of abstract *Clocked* classes (including classes for one-phase clocked models (Clocked), two-phase clocked models (Biclocked) and multi-phase clocked models (PLL and Multiclocked)). The clock class represents the component that emits the tick signals and the Clocked classes represent the basic behaviour of synchronous components. During simulation, each clocked entity first registers with the clock entity from which it wants to receive the tick signal. The clock entity sends a tick to each of the registered clocked entities and waits until it has received the status of each clocked entity before sending the next tick. This process

continues until the end of a simulation; this occurs when the clock entity detect errors in the simulation or the normal termination routine *stopSimulation* has been called by an entity. More details of the HASE clock mechanisms and implementation of a multiple clock mechanism can be found in [63].

To define a clocked entity involves two steps. The first step is to invoke these clock mechanisms in the ENTITY lib in the project definition file, which includes (1) declaring the Clock entity and the abstract Clocked entity in the ENTITY lib; and (2) defining an entity as a Clocked entity by using the EXTENDS option in the definition of an entity to link the entity to the clock. The second step is to code the entity behavioural file as an *n*-phased file, where *n* is the number phases that the user wants to define for the entity.

Figure 4.4 illustrates an example of defining a Biclocked entity in HASE. The first step in defining the CPU entity as a Biclocked entity is to declare a Biclocked CPU entity in the ENTITYLIB of the project file, as shown in Figure 4.4 (a). The declaration includes the declaration of a Clock entity, an abstract Biclocked entity, a Clockphase entity and the CPU entity extended from Biclcked entity in the ENTITYLIB. The second step is to code the CPU behaviour file as a two-phased entity by adding *$phase0* and *$phase 1* in the .hase file, as shown in Figure 4.4 (b). The Biclocked entity keeps track of the number of ticks sent by the clock entity. When the tick number is even, the activities defined in *$phase0* are invoked, whereas when the tick number is odd, the activities defined in *$phase1* are invoked.

```
ENTITYLIB(
    Abstrac Biclocked sync()
    ENTITY Clock sync( )
    ENTITY Clockphase sync()
    ENTITY CPU(
        EXTENDS (Biclocked)
    ...
    )
)
```

(a) .edl file

```
$phase0
    activities in phase0 ...

$phase 1
    activities in phase1 ...
```

(b) .hase file

Figure 4.4: **Defining a clocked entity in HASE** (a) the ENTITYLIB defined in the .edl file; (b) entity behaviour defined in the .hase file.

4.2 SIMRAID Model Design

SIMRAID was initially developed for the Storlite project [45], which was a collaborative industry-academia research project funded under the DTI /EPSRC LINK Information Storage and Displays (LINK ISD) programme and led by Xyratex, a disk storage company based in Havant. However, by the end of the Storlite project, SIMRAID was still not completed. As an extension of Storlite, this PhD project has continued to use the SIMRAID model as the simulation tool and both the functionality and performance of SIMRAID have been further enhanced. This section presents the design and implementation of the SIMRAID simulation model, as well as the various model enhancements. Firstly, an overview of SIMRAID is presented. Secondly, the transportation level abstraction technique created for SIMRAID to shorten simulation time is presented. Thirdly, the framework for modelling heterogeneous systems is introduced. Fourthly, the design of each components of SIMRAID is introduced. Fifthly, the method on how to model large scale systems is described. Lastly, the enhancements to SIMRAID are summarised.

4.2.1 OVERVIEW OF SIMRAID

Chapter 1 introduced five trends in modern RAID systems architecture design. A successful simulation model of storage systems should conform to these features. From analysis of these features, four key characteristics enabling efficient simulation of RAID system architectures can be identified. These are:

1. **Benchmark generated workload**: To create a simulation that provides a realistic performance estimate, it is necessary to provide a realistic traffic flow to it. However, given dramatic increases in storage system scale, extremely large trace files make trace file driven simulation of storage systems impractical. Thus a benchmark that reflects a real life workload is preferred. A recently built benchmark Storage Performance Council Benchmark 1 (SPC-1) [5] is such a benchmark meeting this requirement. It provides flows based on a statistical model derived from analysis of recorded traffic flows from systems running in real live settings. Therefore, the SPC-1 benchmark has been chosen as the major benchmark. Iometer [47], which reflects sequential workloads and is still often used in industry, is used occasionally in this dissertation for comparison purposes.

2. **Parameterisable modular implementation:** To enable a simulation model to

be scaled, all the component parts must be fully modular to allow scaling by the simple addition and removal of modules. This approach also allows simple operational models to be used for devices not under investigation alongside increasingly complex models of devices that are being tested. For instance, it helps to model increasingly complex RAID controllers. This 'plug and play' approach enables faster simulation due to the reduction in complexity. Furthermore, making each component of the model highly parameterisable enables design space exploration using a wide range of parameters.

3. **Framework for modelling heterogenous systems:** To model heterogenous systems, SIMRAID sets a framework which separates the interconnection and device models. By separating these models, it is possible simply to replace either the device behaviour model or the interconnect behaviour model without affecting other parts of the system. This enables isolation of the parts of the system under test for investigation purposes.

4. **Transport level abstraction of interconnection protocols:** Research work introduced in Section 3.5.2 shows that exact simulation of FC-AL results in very large ratios of simulation time over real time (on the order of 1,000,000). Since it is necessary for a storage system simulation to be run for a "warm-up" time of at least 5 minutes to remove start-up transients, exact simulation is not practical. Therefore, a set of abstraction techniques has been developed to extract the behaviour of the interconnection protocol without losing accuracy; this is introduced later as the "transport level abstraction".

Figure 4.5 shows a logical overview of SIMRAID. As can be seen in the figure, this model consists of five basic modules: a traffic generator, RAID controllers, disks, an interconnection interface and an interconnection network. The simulation model works as follows: the traffic generator generates volume level I/O commands in accordance with its rule-set; these are then sent to the RAID controllers. The RAID controllers decode these commands into a set of disk operations and send these to the disks for execution through the interconnection interface and network. The disk executes the commands and send the results back to the RAID controllers. There is also an "experiment controller" module used to initiate and control experiments and a "results collector" for collection and reporting of experiment results.

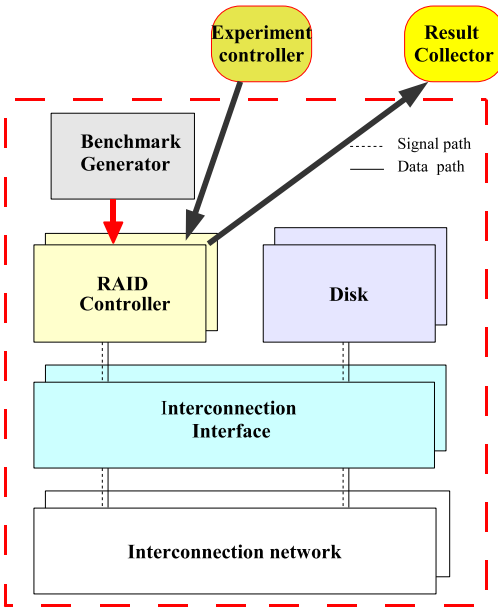


Figure 4.5: **Overview of** SIMRAID

4.2.2 Transport Level Abstraction Techniques

Previous research shows that direct simulation of the interconnection protocol (*e.g.* FC Arbitrated Loop (FC-AL)) or word level simulation results in very large ratios of simulation time over real time [101], which makes simulation using an SPC-1 benchmark workload impractical. Therefore, it is crucial to make an appropriate abstraction. Since the purpose of SIMRAID is to investigate the impact of architecture and configuration changes on overall system performance, it is of more interest to focus on the service time of disk commands than the transmission time of FC-AL data. To achieve this goal, SIMRAID uses a number of abstraction techniques to form a new abstracted interconnection protocol. This new interconnection protocol focuses on end-to-end control and allows accurate representation of disk command service times without full FC-AL simulation. Since it is much like the transport layer of the OSI reference model, it is called a "Transport level abstraction".

The abstraction approach consists of two techniques:

- **Removing redundant and deterministic transmissions:** As introduced in Section 2.2.3, the basic unit of FC-AL communication is a 40-bit long FC word which is 8B/10B encoded and represents four 8-bit bytes. In an accurate discrete event simulation, each communication word is handled as an event in the system. Therefore, reducing the number of words transmitted will reduce both the resources used and the time to handle these event. In doing so, and without af-

fecting the simulation results, SIMRAID removed three types of communication from the simulation.

Firstly, the IDLE data words which are transmitted in a real system to maintain synchronisation in the clock recovery circuitry of the receiver have been simply ignored in the model as they serve no end-to-end communication purpose. Secondly, deterministic communications that do not carry active data are implemented in SIMRAID by means of a communication channel time lock instead of by creating events. Since the topology of the system is known, the time that would have been taken up by these transmissions is known. Therefore this time can be set aside on the relevant communications links as the transmissions are eliminated. Thirdly, SIMRAID combines the separated frames of the payload data into just one transmission. In FC-AL, payload data is broken into frames with a known structure. Each frame contains up to 2 Kbyte of payload data with a frame header and footer, each of pre-determined length. Between frames of a single communication there are also 6 fill-words transmitted which can be used to deal with access control. As will be described below, SIMRAID has separated the data path from the signal path in the communications network, thus allowing access control to be taken out of band. Therefore these 6 fill words can be modelled as an additional transmission delay for the combined data transmission.

- **Single arbitration transmission:** In a physical system a device with data to send to another device will continually attempt to send it until it succeeds. In the case of FC-AL, this results in continuous arbitration with the possibility of rejection of connection even after arbitration is won. Obviously these processes result in unnecessary communication events. To remove these repetitive transmissions, instead of using the regular data path, SIMRAID adds a signal path into the model and makes the switch operate as a central access manager for the arbitration. The central access control manager mimics the access order that would be visible in a real system through tracking of all the communication requests and the status of all devices in the system. In a real implementation this is not possible as all the devices are generic in nature and cannot be programmed with a system topology. However, in simulation it is easy to build the system topology into the switch. In order to eliminate all unnecessary communication associated with arbitrating FC-AL devices, a virtual central controller entity was implemented in the simulation model. This entity is informed when

a device wishes to arbitrate for access to the system via an artificially created signalling link. Hence, a single message, containing the FC-AL address of the sender along with the address of the intended recipient, is passed to the central controller, which retains the information. This way, the central controller knows at any time all FC-AL devices that want to arbitrate. This single message, hence single simulation event, is a substitute for the thousands of messages sent in a real system. As the central controller has full knowledge of the system (such as topology information, history of connections and current list of desired connections), it is able to replicate the order in which they would be granted access in a real system using the FC-AL protocol.

In the case of the JBOD and MBOD, the central management system is simple to implement as it can deterministically follow FC-AL. In the case of the SBOD, fair access must be managed differently as the FC-AL protocol was not designed to operate in a switched manner. Since switch manufacturers guard their fairness algorithms closely, a suitable algorithm has been developed as part of this project, described in the SBOD fairness section below. In this way, a multitude of simulation events on the data path are replaced by a single access request event on the signal path. A short study using a model without this mechanism showed that the speed of a simulation tended very rapidly to become the same as a full FC-AL simulation.

The FC-AL protocol is described in terms of a state transition in table [1]. This dissertation has amended this transition table to remove sections that are present to enable the system to cope with transmission errors and with different topologies. Moreover, it removes states that are transitional and exist only to allow an FC-AL device to update internal variables. SIMRAID also removes the initialisation process from the state diagram as initialisation can be dealt with out of band (Section 4.2.5.1). The amended transition diagram is shown in Figure 4.6.

4.2.3 Framework for modelling heterogeneous systems

As introduced in Chapter 1, storage systems are becoming increasingly heterogeneous. Thus, SIMRAID is designed to be able to model heterogeneous storage systems. In order to do so, the models of devices (RAID controllers and disks) and interfaces are implemented as separate modules. These models are connected using a signal path connection and a data path connection, as shown in Figure 4.5. The data path is used

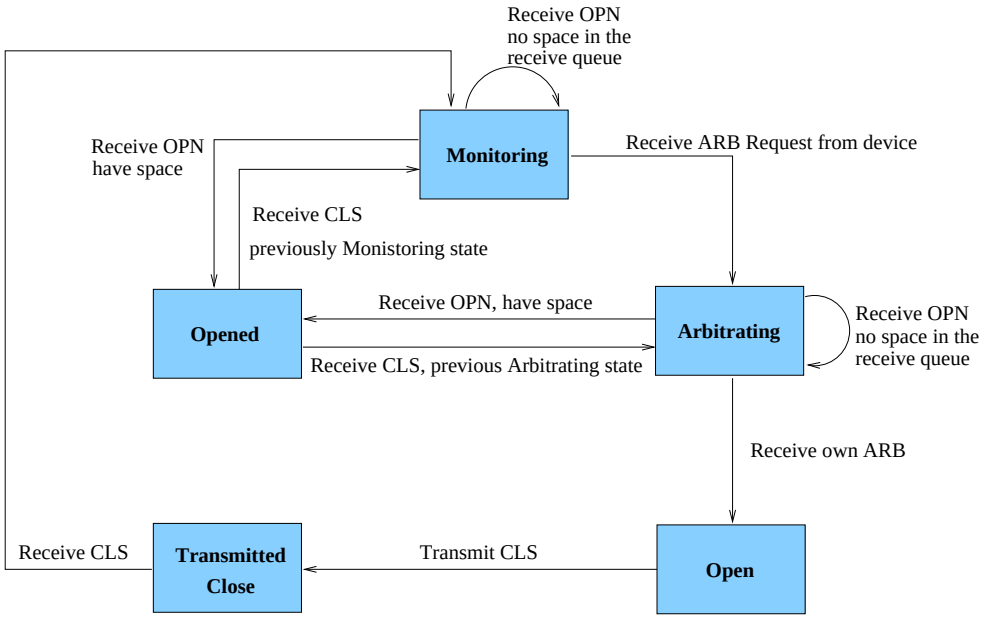


Figure 4.6: **State transition diagram for normal operation**

to transmit active data at specified times (these times are specified to mimic the times at which that data would be received in a real system). The signal path is used for all the access control data that is transmitted asynchronously. The following access control messages have been defined:

- *FCUnitTimeout:* sent by the initiator device to inform the interface that it has data to send
- *arbRequest:* sent by the interface to inform the network switch that it has data to send
- *arbGranted:* sent by the network switch to the interface informing that it is allowed to send data
- *GetData:* sent by the interface to the initiator device to get the data to be sent
- *DeleteData:* sent by the interface to the initiator device to inform it that the data has been sent to the target
- *SetData:* sends the data received by the target interface to the target device
- *UpdateDataStatus:* sent by the initiator interface to the initiator device to indicate that the communication is blocked because the target queue is full

Figure 4.7 illustrates the process of sending data from a RAID controller to a disk in an SBOD. When a device has data to send, it will send an *FCUnitTimeout* to its interface over the signal path. The interface module will then inform the central access control manager (in this case embedded within the SBOD switch model) through an *arbRequest*. When the connection can complete and all the other connections that would take place before this one in a real system have occurred, the central controller sends an *arbGranted* to the interface module. The interface then sends a *GetData* to the device module indicating that access is granted and that the required communication should commence. Following that, the RAID controller sends data to the interface through the data link. There is an appropriate delay to this data to mimic the access grant processing delay existing in a real system.

Following this process, there is a set of actions that follow the Loop Port State Machine defined for FC-AL [1]. Within this process, the signal *SetData* is sent from the interface to the device when the interface receives a *data* FC-AL word; this signal identifies the transmitted data structure and allows the recipient to act upon the command received (either add it to its queue in the case of a disk or mark it as complete in the case of a RAID controller). The central access control unit is looking for the second *close* (*cls*) word as this indicates that a communication is terminated. The central controller is located within the SBOD switch as the *close* will always traverse the switch and so is visible to this module. Once one communication is terminated, the central controller is able to grant access to another pending communication. The final signal is sent from the interface module to the device module *DeleteData* upon receipt of the final *close* by the interface to indicate to the device that the communication has terminated successfully.

4.2.4 Design of the Basic Modules

SIMRAID consists of five basic modules: a benchmark generator, RAID controllers, network interfaces, interconnection networks and disks. This section describes the design of each of these modules.

4.2.4.1 Benchmark Generator

The benchmark generator is able to generate two types of benchmark workload: the SPC Benchmark-1 (SPC-1) workload [5] and the Iometer benchmark workload [47]. The SPC-1 benchmark represents a real-world Online Transaction Processing (OLTP)

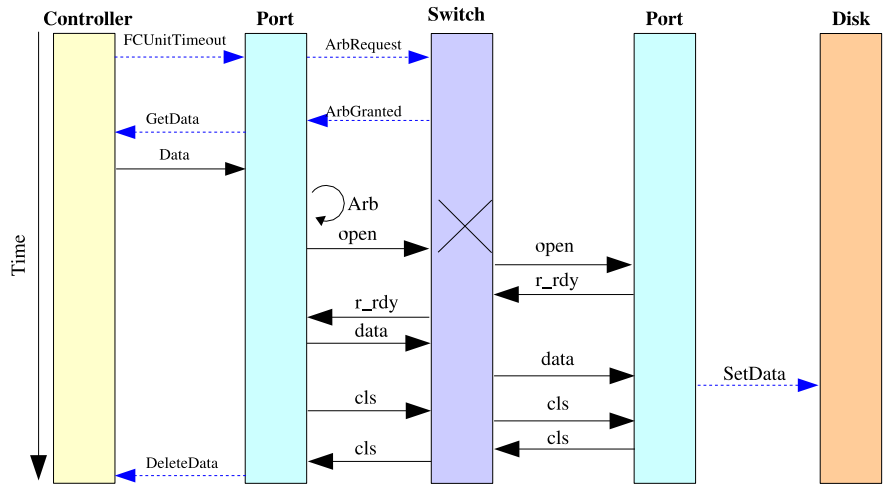


Figure 4.7: **Disk command communication process** Dashed lines represent the transmission of a signal message over the signal path and solid lines represent the transmission of data over the data path.

environment workload, whereas the Iometer benchmark represents a sequential workload environment. Figure 4.8 shows the structure diagram of the benchmark generator. The benchmark generator includes two layers of software: the workload generator and the workload splitter. During the simulation, the model can be configured to use either the SPC-1 benchmark or the Iometer benchmark. The workload generator generates I/O commands based on the benchmark rules. These I/O commands are sent to the workload splitter. The workload splitter sends the I/O command to the RAID controllers based on the controller selection policy. Depending on the policy, it can evenly split commands between both RAID controllers or just send commands to a single controller.

4.2.4.1.1 SPC-1 Benchmark The SPC Benchmark-1 [5] is the first standard industry storage benchmark based on application driven trace data and the first standard benchmark for Storage Area Networks (SANs). SPC-1 uses a highly efficient multi-platform and multi-threaded workload to emulate the precise characteristics of multi-user I/O in a multi-application environment. The most prominent group of applications that display a “core” of common I/O characteristics is represented by OLTP systems, database systems, or mail server applications. The “cores” of these applications are characterised by predominantly random I/O operations requiring queries as well as update operations and simultaneous threads of sequential I/O processing. Storage Performance council (SPC) also has two other type of benchmark: SPC-2 [6]

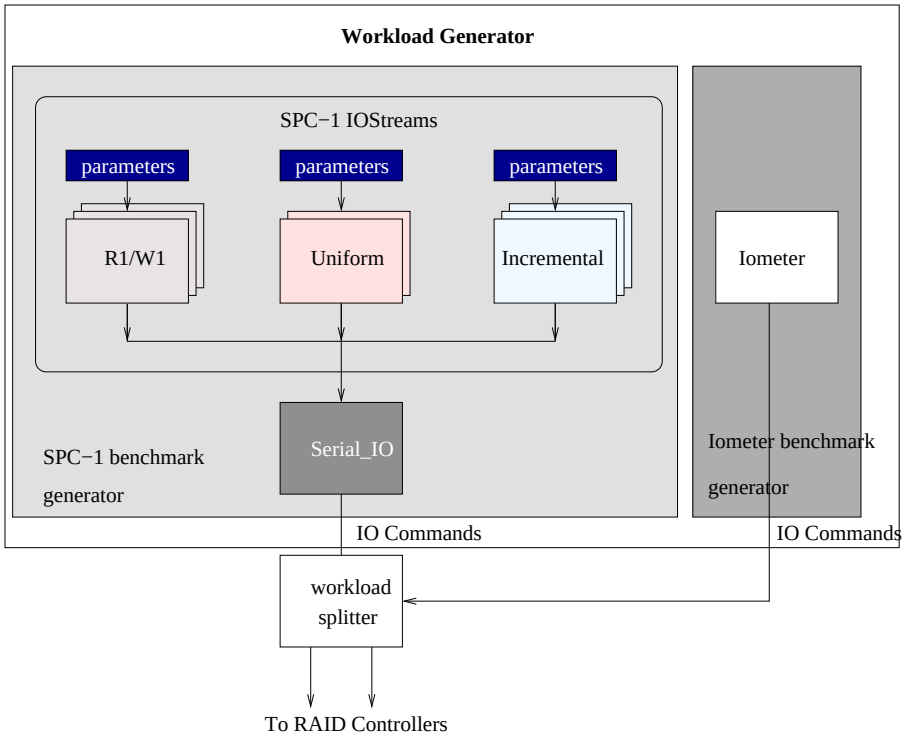


Figure 4.8: **Structure diagram of Benchmark Workload Generator**

and SPC-3BR [7]. SPC-2 represents sequential workloads and SPC-3BR is a content management benchmark to measure to the performance of backup/restore solutions. Since most commercial storage systems are used for OLTP, SIMRAID chooses SPC-1 as the workload. SPC-2 and SPC-3BR will be included in SIMRAID later.

Elements of the I/O profile include: read/write ratio, I/O size, locality, re-reference probability, sequentiality, and inter-arrival time. Central to the SPC-1 benchmark are two scaling units: the Application Storage Unit (ASU) and the Business Scaling Unit (BSU).

The SPC-1 benchmark synthesises a community of users running on storage that is organised into three logically separate Application Storage Units (ASUs), as would be encountered in a real-world installation. ASU-1 is a **data store**, holding raw incoming data from the application system. It contains four kinds of I/O stream. ASU-2 is a **user store**, holding information processed by the application system. It contains three kinds of I/O stream. ASU-3 represents a **log** written by the application system. It only contains one I/O stream. BSUs represent the I/O load imposed by the application's user population on the ASUs. Each BSU represents the aggregate I/O load created by a specified number of users. In the SPC-1 benchmark, one SPC-1 BSU corresponds to a community of users who collectively generate up to 50 I/O requests per second

(IOPS). The workload profile of each I/O stream that comprises the SPC-1 aggregate load is summarised in Table 4.1. All transfer addresses are aligned to 8 logic blocks.

To measure performance, the SPC-1 benchmark increases the offered IOPS by increasing the number of BSUs. The SPC-1 benchmark specifies that the reported average response time must not exceed 30 milliseconds (ms) for all test stages, otherwise the measurement is invalid. Therefore, most of the results from the experiments reported in this thesis are given in terms of the maximum BSU that can be used whilst maintaining an average response time of less than 30ms.

	ASU 1			
IOstream	1	2	3	4
Transfer alignment (KB)	4	4	4	4
Transfer Size (KB)	4	4	SMIX [a]	4
Rd Fraction	0.5	0.5	1.0	0.5
Intensity	0.035	0.281	0.07	0.21
	ASU 2			**ASU 3**
IOstream	1	2	3	1
Transfer alignment (KB)	4	4	4	4
Transfer Size (KB)	4	4	SMIX	SMIX
Rd Fraction	0.3	0.3	1.0	0.0
Intensity	0.018	0.07	0.035	0.281

Table 4.1: SPC-1 I/O stream parameters

[a] SMIX = Table: $\{4, 0.40\}$ $\{8, 0.24\}$ $\{16, 0.20\}$ $\{32, 0.08\}$ $\{64, 0.08\}$

4.2.4.1.2 Iometer Benchmark Iometer is an I/O subsystem measurement and characterisation tool. It issues a fully sequential workload to the storage system and keeps the outstanding number of I/O requests fixed at 75. Iometer tests determine two system parameters: I/O requests Per Second (IOPS) and throughput. Since this disserta-

[1] A logic block = 512 bytes.

tion uses Iometer to investigate network scalability under sequential workloads, system throughput is used as the metric.

4.2.4.1.3 SPC-1 Benchmark Generator As introduced in Section 4.2.4.1.1, the workload of the SPC-1 benchmark is measured by the number of BSUs. Each BSU represent 50 I/Os per second collectively generated by the eight I/O streams. To generate 50 I/O commands for each BSU every second, SPC-1 streams adopt the biclocked model. Each second is divided into 50 two-phased cycles, with phase 0 taking 1.0 ms and phase 1 taking 19 ms. In each cycle, there is one I/O command collectively generated from the eight SPC-1 streams. Phase 0 of each cycle is used for generating the I/O commands which conform to the stream definition. Phase 1 of each cycle is used to send the I/O commands to the Serial_IO module. Since there is only one I/O command for eight streams, each stream has a random number generator to control command generation. Before generating an I/O command, each stream generates a random number. Only when this random number is larger than the intensity parameter will an I/O command be generated. Usually, the workload of the simulation is larger than one BSU. To save computing resource, there is only one instance of each stream rather than having eight stream instances for each BSU. In phase 0 of each cycle, each streams repeats the aforementioned command generation process for each BSU and stores these I/O commands. In phase 1, each stream sends out the commands generated in phase 0 with equal intervals between each command.

The Serial_IO module receives the commands from each stream in phase 0 of the next cycle and stores them in a queue. In phase 1, Serial_IO sends these commands to the workload splitter with a delay equal to timestamp of the command plus a cycle period and the front end network transmission time. Table 4.2 lists the parameters of the SPC-1 generator.

Parameter	**Note**
Load_BSU	The number of SPC-1 BSUs
ASU capacity	The data capacity of the simulated system

Table 4.2: Parameters of the SPC-1 benchmark generator

4.2.4.1.4 Iometer Benchmark Generator The essence of IOmeter is to keep the outstanding commands fixed at 75. At the start of a simulation, the Iometer benchmark

generator generates 75 sequential commands and send them to the RAID controllers. Once it receives a complete acknowledgement from the RAID controller, the generator generates a new sequential command. The parameters of the Iometer generator are listed in Table 4.3. In practice, as the Iometer generator needs to receive acknowledgements from the RAID controllers, the Iometer generator is implemented in the workload splitter file.

Parameter	**Note**
Transfer size	The transfer size of each command
Read Fraction	The fraction of read commands
Outstanding IO	Fixed at 75 for Iometer

Table 4.3: Parameters of the Iometer benchmark generator

4.2.4.2 RAID controller

The model of the RAID controller is built based on the hardware architecture introduced in Section 2.2.2. According to the architecture and functionality of RAID controllers, the model of the RAID controller is divided into seven modules, as shown in Figure 4.9. These modules are linked together by the RAID controller behaviour file. The role of each module is as follows:

- Command store: Stores the received volume level commands and decomposes them into disk commands according to the RAID algorithm used.
- FIFO Queue: Emulates the time spent in the main processer of the RAID controller, including the command decomposition time and the time to store a command.
- Cache Table: Emulates the behaviour of cache; write through and write back cache policies have been implemented.
- Read/Write Queue: Manages the disk commands that have sent FCUnitTimout to the interconnection interface but have not yet been sent to disk.
- Read/write Blocked Queue: Manages the disk commands that cannot be sent to disk because the disk queue is full.

- Read/write Scheduled Queue: Manages the disk commands that are in the process of being sent to disk.
- Write Mirroring Controller: Controls write data mirroring between RAID controllers.

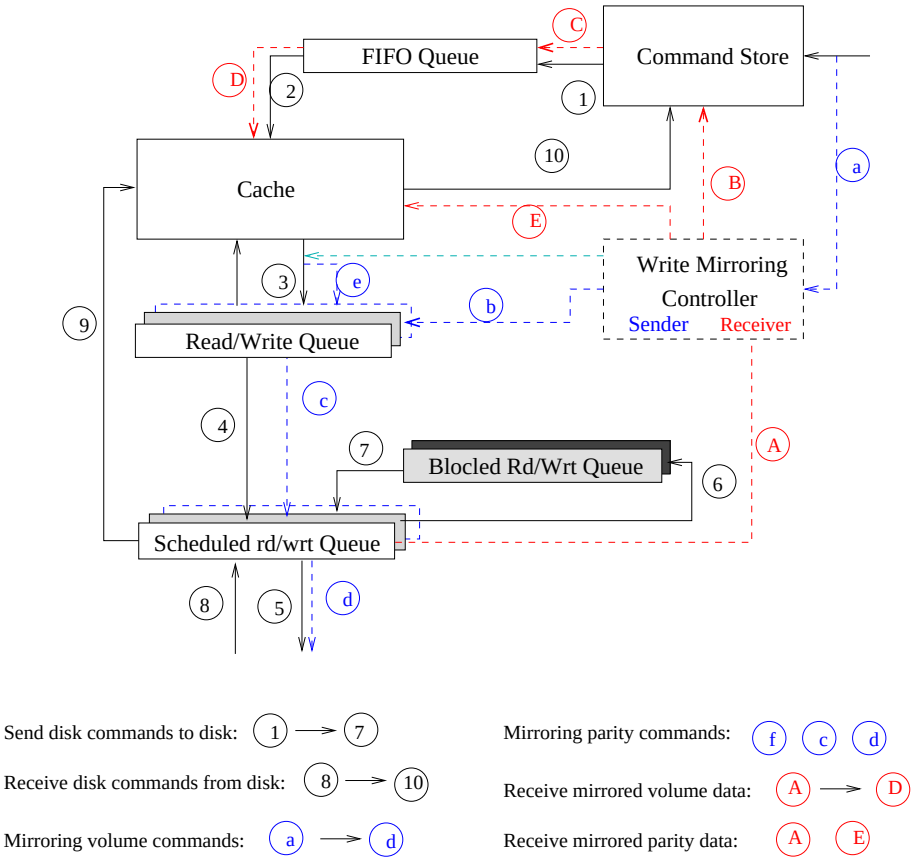


Figure 4.9: **Diagram of RAID controller model**

The main functionality of the RAID controller is to serve the volume-level commands received from the benchmark generator. This process works as follows (Figure 4.9 (1)-(7)): upon receiving a volume level command, the Command Store module stores the volume command and decomposes it into disk commands according to the selected RAID protection level and operation mode. These disk commands are then pushed to the FIFO queue, which has a time based exit strategy to mimic the time taken for command decoding. Once the time has advanced sufficiently, disk commands are inserted into the cache table. If there is a hit, the Command Store will be informed that the disk command is completed; otherwise the disk command will be sent to the Read/Write Queue. Once the disk commands are pushed to the Read/Write Queue

(Figure 4.9 (3)), the *FCUnitTimeout* will be sent to the interface. After *GetData* is received from the interface, the disk command will be popped from the Read/Write Queue and pushed to the scheduled queue (Figure 4.9 (4)). If the command cannot be sent to disk because the disk queue is full, the interface will inform the controller by sending an *UpdateDataStatus* signal. The command will then be pushed to the Blocked Queue (Figure 4.9 (6)). It will be sent to disk again once the disk queue is no longer full. Figure 4.9 (8)-(10) shows the data flow of receiving a disk command response. When a disk response is received, SIMRAID first gets the disk command pointer from the scheduled queue based on its disk number, address and command type. It then informs the cache table and Command Store that the disk response has been received for that disk command.

4.2.4.2.1 Command Store The main functionality of the Command store is to decompose incoming volume commands to corresponding disk commands and keep track the status of each volume commands. To fulfill this functionality, the command store keeps all outstanding volume commands in a hash table. Upon receiving a volume command, the command store assigns a unique sequential number to this volume command and stores the command in the hash table by using this number as the hash key. Based on the logic block address (LBA) of this command, the command store then decomposes the volume command into the corresponding disk commands following the operation rules described in Section 2.1.2 and the pre-defined data layouts. The system parameters that control this decomposition are listed in Table 4.4. The generated read disk commands and write disk commands are stored in two separate tables, with the stripe number and stripe unit number as the indexing key. Once the decomposition is completed, the command store sends the disk read commands and the disk write commands that form a full stripe write to the FIFO queue and waits for the response. When a disk command response is received, this disk command is deleted from the hash table. In the case of a partial stripe write, once the command store receives responses for all the read commands of that stripe, it sends write commands to the FIFO queue. The volume command is then deleted from the command store after all disk commands have completed.

4.2.4.2.2 Cache Table Cache has been widely used in modern RAID systems to improve system performance. In a real RAID system, the cache is divided into fixed size blocks of 4kbytes. Blocks are linked to form disk level accesses of multiple blocks (*i.e.*

Parameter	**Values**
RAID level	RAID 5 or RAID 6
Operation Mode	normal, degraded or rebuilding
Size of Stripe Unit	4kB, 8kB, 16kB, 32 kB or 64 KB
Strip width	any number

Table 4.4: Command Store Parameters

linked to form stripe units). Since all disk accesses are at stripe unit level, the cache line size is set to the size of the stripe units. In order to reduce the time to search the cache, the whole cache is implemented as a two-dimensional array of maps. Each map is called a cache set, which consists a number of cache lines. All cache sets have the same number of cache lines. The first dimension index of the array is the disk number while the second dimension index is (address/STRIPE_UNIT_SIZE)%NUMBER_OF_MAPS, where NUMBER_OF_MAPS is a parameter representing how many cache sets each disk has; it can be configured by the user. For instance, Cache_table[1][3] represent the third cache set of the first disk. A cache line is the basic unit of the cache table. Each cache line has multiple flags showing its status. It also has a disk command pointer which points to the uncompleted disk command that hit this cache line. Figure 4.10 shows the structure of the cache table. The parameters of the cache table include cache size, cache policy and NUMBER_OF_MAPS, are shown in Table 4.5.

Parameter	**Values**
Cache Size	any number larger than 0
Cache Policy	write-through or write back
NUMBER_OF_MAPS	any number

Table 4.5: Cache Table Parameters

The cache table implements two cache policies: write-through and write-back. Table 4.6 and Table 4.7 list the cache operations when a disk command is inserted into it under the write-through policy and write-back policy respectively. The cache operation depends on three conditions, namely whether the cache location exists, the disk command type and the cache status. The cache table adopts a Least Recently Used (LRU) replacement policy. Whenever a cache line is accessed, its timestamp is updated. When a disk command with a non-existent address is inserted into the cache

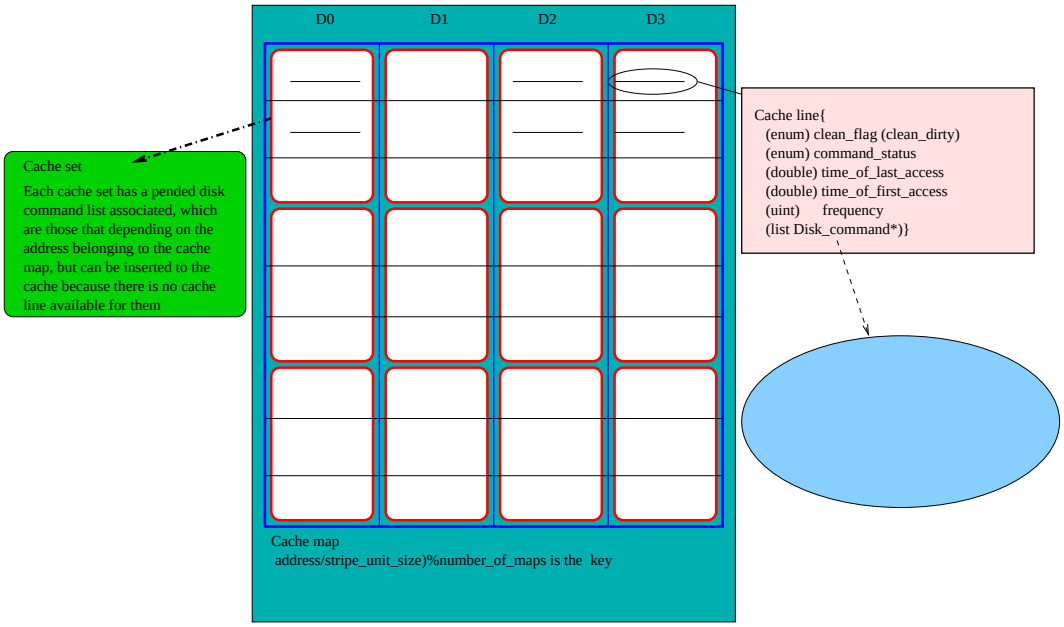


Figure 4.10: **Structure of the cache table**

table, it searches for the clean cache line that has the oldest timestamp. In addition to the cache lines, each cache set also contains a pending disk command list. When the cache location for a disk command does not exist and there is no free cache line available, this disk command is pushed into the pending command list. When the response from a disk command is received, the cache line is marked clean and the command is removed from the disk command list.

Cache Location Exists		Command Type	Cache Clean	Operation						
				Allocate Cache	Mark Dirty	Cache Hit	Send To Disk	Command Finished	Add to List	Return Result
Yes		Read	Yes	×	×	√	×	√	×	$hit\&\overline{send}$
			No	×	×	×	×	×	√	$\overline{hit}\&\overline{send}$
		Write	Yes	×	√	×	√	×	√	$\overline{hit}\&send$
			No	×	×	×	×	×	√	$\overline{hit}\&\overline{send}$
No	free line[a]	Read		√	√	×	√	×	√	$\overline{hit}\&send$
		Write		√	√	×	√	×	√	$\overline{hit}\&send$
	No	Add the command to the pending command list associated with corresponding map								

Table 4.6: Cache Operation of Inserting a Disk Command under Write Through Policy

[a]Free cache lines refer to both unused and clean ones. The program will first check if there is an unused cache line. if not, it will take the first clean line.

Cache Location Exists		Command Type	Cache Clean	Operation						
				Allocate Cache	Mark Dirty	Cache Hit	Send To Disk	Command Finished	Add to List	Return Result
Yes		Read	Yes	×	×	√	×	√	×	$hit\&\overline{send}$
			No	×	×	×	×	×	√	$\overline{hit}\&\overline{send}$
		Write	Yes	×	×	√	√	√	×	$hit\&send$
			No	×	×	×	×	×	√	$\overline{hit}\&\overline{send}$
No	free line[a]	Read		√	√	×	√	×	√	$\overline{hit}\&send$
		Write		√	×	√	√	√	×	hit& send
	No	Add the command to the pending command list associated with corresponding map								

Table 4.7: Cache Operation of Inserting a Disk Command under Write Back Policy

[a]Free cache lines refer to both unused and clean ones. The program will first check if there is unused cache line. if not, it will take the first clean line.

4.2.4.2.3 Write Data Mirroring SIMRAID can be configured to work either with or without write mirroring.The write mirroring controller controls the write data mirroring process. When the RAID controller is working with write data mirroring (Section 2.2.2.4), the process of sending disk commands to disk happens only after the necessary data has been sent to the other controller. Under the write-back cache policy, when a volume-level command is received from the host, the command store starts to decompose it. In the meantime, the write mirroring controller sends it to the other controller, as shown in Figure 4.9 (a)-(d). Once acknowledgement of the mirroring has been received and time has advanced sufficiently to pop the commands from the FIFO, disk-level read commands are sent to the disks (Figure 4.9 (2)-(7)). After the responses to the read commands are received from the disks, the disk-level write commands are generated. Before sending these write commands to the disks, the parity commands are mirrored to the other controller (Figure 4.9 (e) (c)(d)). Once the parity commands have been acknowledged, the disk-level commands are sent to the disks. Under the write-through cache policy, only the generated disk-level write commands need to be mirrored and the process also follows Figure 4.9 (e) (c)(d).

The process of receiving mirrored data is simpler. When it receives a mirrored volume command (under write-back cache policy), the receiving controller first stores it in its command store, then decomposes it and stores it to the cache table (Figure 4.9 (A)-(D)). When a disk-level command is received, since the controller does not need to decompose it, the disk command is just stored in the cache table (Figure 4.9 (A)(B)).

4.2.4.3 Interconnection interface ENTITY

Each device maintains its own interconnection interface unit to the communication network. The interconnection interface ENTITY consists of a number ports connected together by FC_port.hase. The interface ENTITY communicates with the FC device and the network ENTITY. For the FC interface it usually contains two ports with each of them being connected to the network to provide communication path redundancy. Figure 4.11 shows its structure and communication interface. Each port is an independent object implementing its own Loop Port State Machine (LPSM). This implementation facilitates both changing the number of ports for a device and changing to the state transition table for new protocols. The LPSM works with accordance with the Transport Level Abstraction techniques introduced in Section 4.2.2. Communication with the FC device and the network was introduced in section 4.2.3. The parameters of the interconnection interface ENTITY are listed in Table 4.8.

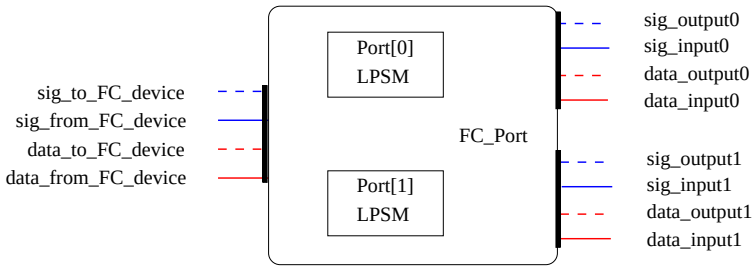


Figure 4.11: **FC Port ENTITY**

Parameter	**Definition**
port_bandwidth	
number_of_ports	the number of ports that the interface has
port_selection_policy	Single, Alternate or Random

Table 4.8: Interconnection Interface Parameters

4.2.4.4 Interconnection Network ENTITY

Figure 4.12 shows the structure and interface of the Interconnection Network Entity. It has a number of PORTS (including both signal path and data path ports) each of which connects to an interconnection interface ENTITY. This network ENTITY is configurable either as a switch or a hub. This allows the unit to mimic the behaviour of the three main system topologies, JBOD, MBOD and SBOD. The interconnection network entity controls the arbitration of the interfaces and their connections. The parameters of the interconnection network entity are the number of ports it has and its architecture, as shown in Table 4.9.

Parameter	**Value**
architecture	SBOD or MBOD
number_of_ports	

Table 4.9: Interconnection Network Parameters

4.2.4.4.1 Fairness Control In SIMRAID the central arbitration manager is embedded within the SBOD switch and so is able to see the FC-AL communications as well as the management signalling. This allows it to monitor ongoing communications and determine when a connection is terminated. When a new request for arbitration is received, a counter of the number of outstanding arbitration requests between a given

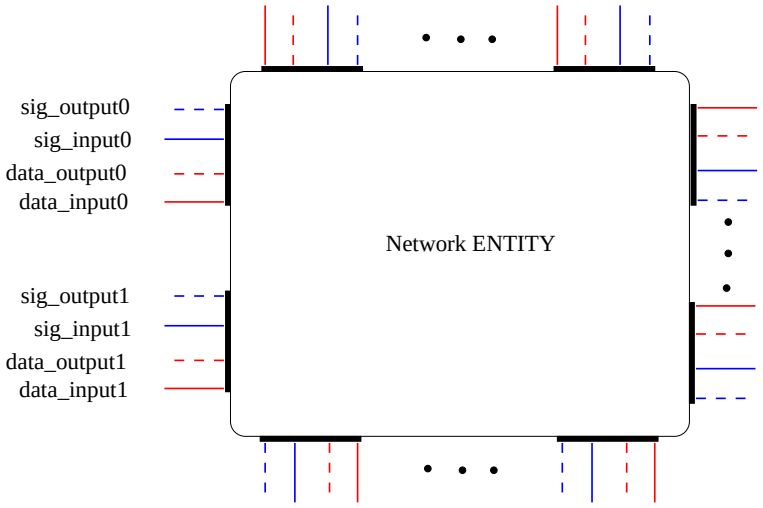


Figure 4.12: **Interconnection network ENTITY**

source and target is incremented. If the connection can be made immediately, the switch responds with a permission signal and the source is able to initiate FC-AL communications with its target. At this point the counter is decremented. This process is synchronised using the HASE clock to match the timings of a real SBOD FC-AL system. Once a communication is terminated, it is necessary to determine if a new connection can be established. There are 4 options that are tested in the sequence indicated in the flow diagram of Figure 4.13. Note that an arbitration request can only be granted if the switch ports for both the source and target devices of that arbitration request are not currently involved in an active communication. To ensure fairness in the system, it is also necessary to decide which of two outstanding arbitration requests is granted in the 2nd and 4th scan actions in Figure 4.13. This is achieved through the concept of priority of access.

At initialisation, each port of the switch is given a priority list. This is initialised to have all the ports in numerical order starting with the next numbered port. For this discussion, the port owning the priority list is known as the 1st port. When a port is granted access to the 1st port (the accessing port being known as the 2nd port) the 2nd port's number is sent to the bottom of the priority list. When searching for an arbitration request to grant in scan actions 2 and 4, ports are scanned in the order in which they appear in the priority list. This system helps to ensure fairness of access and to prevent starvation.

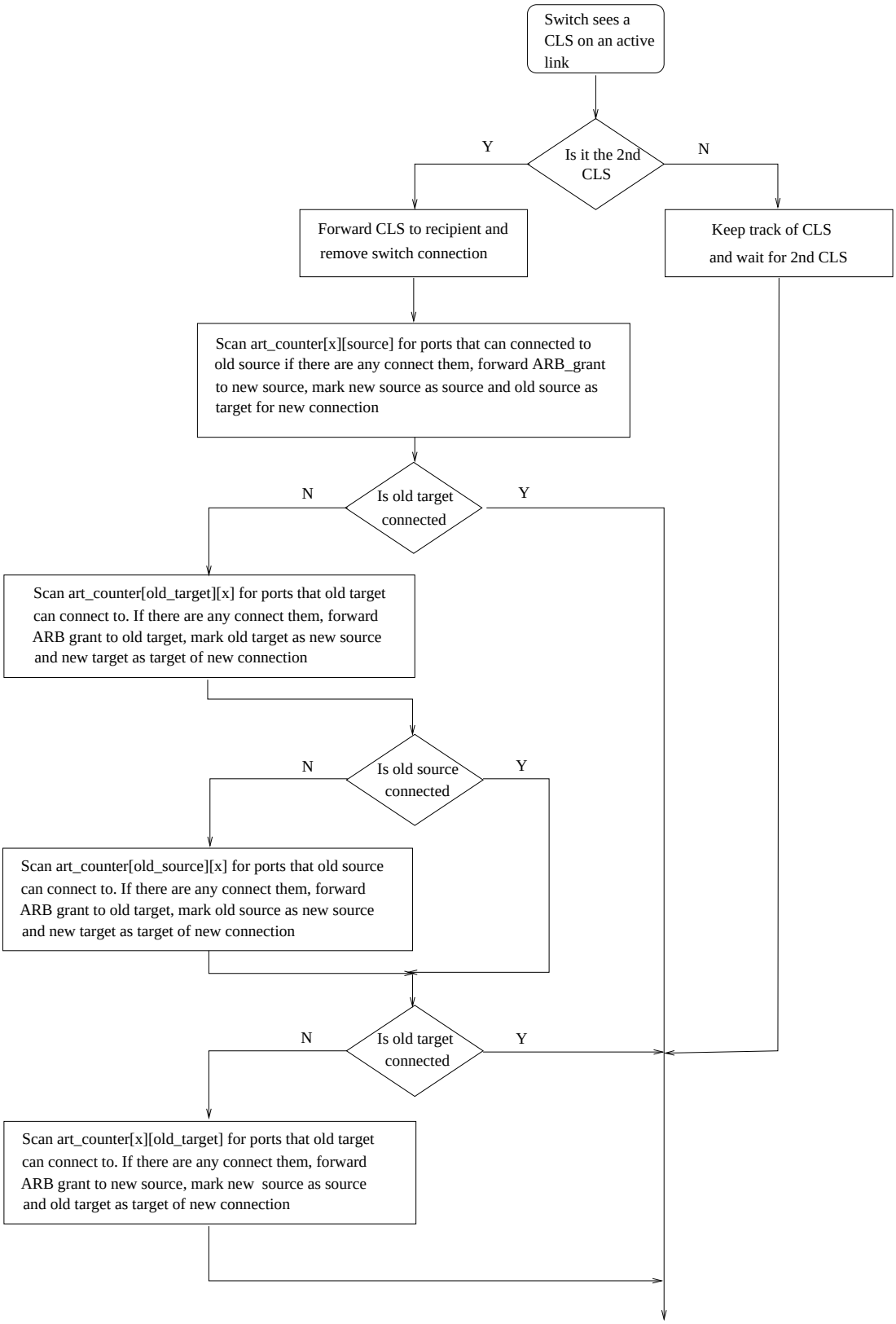


Figure 4.13: **Flow diagram of connection termination**

4.2.4.5 Disk ENITTY

The normal disk is modelled as a queue with maximum length of *max_queue_size* which is a parameter of the model. Data is obtained from experimentation on real disks to determine the response time of the disks. In this experiment, different workloads are injected to disks to gather information of the response time of each disk command. It first gathers the trace data of each disk commands including the time stamps, the number of commands currently in the queue, the sequentiality of these commands, the command type and the information on whether a disk cache is used. The statistic of response time is the then generated from these trace data. The mean response time for each set of parameters (the number of commands currently in the queue, the sequentiality of these commands, the command type and whether a disk cache is used) is then calculated. The disk response time is organised to a table and the indexes are those aforementioned parameters. When the disk model receives a disk command, it will check the table to determine the response time for that command based on its type, the number of commands in the diks queue, the sequentiality of those commands and whether disk cache are used.

The disk entity uses a signal path and a data path to communicate with its interconnection interface, as shown in Figure 4.14.

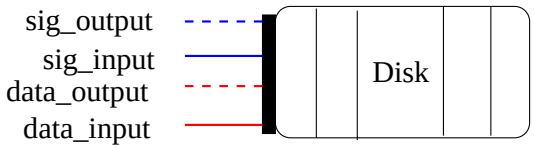


Figure 4.14: **Disk ENTITY**

4.2.5 Modelling Large-Scale RAID Systems

Large scale RAID systems usually consist of more than one FC loop and RAID controller. Each FC loop comprises two FC switches for redundancy and a number of disks. The raid controllers connect to each FC loop through an FC dual port. In order to model such a complicated large scale system with least effort, SIMRAID deploys a hierarchical model design method by using the COMPENTITY template provided by HASE and the basic modules described in Section 4.2.4. Three types of compound entity are defined: RAID_Controller_system, Disk_sub_system and FC_Loop, as shown in Figure 4.15. The RAID_Controller_systems COMENTITY comprises two types of

basic entity: a raid controller entity and *n* FC dual port entities, where *n* is the number of loops in the system. The Disk_sub_system COMPENTITY consists of a disk entity and an FC dual port entity. Each FC_Loop COMPENTITY comprise two FC switches and a number of Disk_sub_system entities. In addition, a model initialisation protocol is designed to allocate an FC address to each component, which avoids allocating FC addresses manually in the EDL file. This initialisation process is launched at the beginning of the simulation. This initialisation process also builds the topology information in the RAID controller and the switch. With these compound entities and the model initialisation protocol, simulating a RAID system with multiple RAID controllers and FC loops is just a matter of instantiating the corresponding number of RAID controllers and FC loops and connecting them in the structure part of the EDL file. The following subsection described the initialization protocol in more detail.

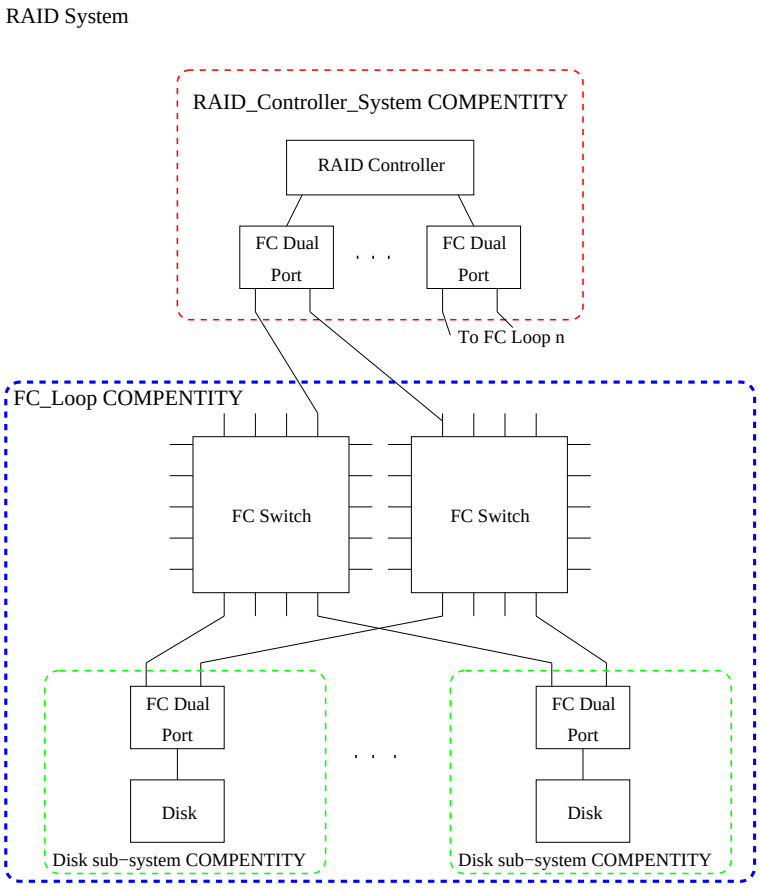


Figure 4.15: **Hierarchical Model Design**

4.2.5.1 Model Initialisation Process

In the FC-AL protocol, initialisation occurs at system start-up and may then be instigated by any device in the system at any time to overcome perceived errors. Initialisation must terminate any ongoing communications with immediate effect (to be restarted after initialisation); it must also perform address assignment. The initialisation of FC-AL systems has four stages: negotiation of an initialisation master, assignment of previous addresses, assignment of hard addresses and finally assignment of soft addresses. The previous address assignment phase is used during re-initialisation and allows previously connected devices to maintain a constant address across the initialisation.

In SIMRAID the initialisation process is different. Firstly, there is no negotiation of a master; this role is taken by the RAID controller. Secondly, the initialisation only occurs at the start-up of the model and its purpose is to simplify the address allocation process and build the network topology information in the SBOD switch and the RAID controllers. The initialisation process is started by the pre-designated main RAID controller. This main RAID controller starts the initialization process by sending predefined primitive messages *StartInitialisation* to the FC switches of its first FC loop through the FC dual port, as shown in the first step of Figure 4.16. After receiving this message, the main switch[2] sends a primitive message *GetDeviceType* to all RAID controllers and disks connected to it (Step 2 in Figure 4.16). After receiving *GetDeviceType*, all RAID controllers (including the main RAID controller) and disks connected to that switch send back their device type through message *SetDeviceType* (Step 3 in Figure 4.16). Based on the device type (RAID controller or disk), the main switch allocates FC addresses to the RAID controllers and disks, and then sends the FC addresses to them through *SetAddress* messages (Step 4 in Figure 4.16). While forwarding the address to the device, each FC port also records its own FC address.

After receiving their FC addresses, all devices connected to that switch acknowledge receipt by sending messages *AcceptAddress* to both switches of that FC loop with their allocated addresses included (Step 5 in Figure 4.16). Since these *AcceptAddress* messages are sent to both switches in the loop, the non-main switch also gathers the device FC address information at this step. The main switch then sends the address of each disk connected to it to all controllers so that the RAID controllers are able to get the disk information (Step 6 and 7 in Figure 4.16). The controllers acknowledge

[2] One of the two SBOD switches of each FC loop is designated as the main switch

receipt of all disk addresses by sending *AcceptDiskAddress*, which is the last step of the initialisation process for this FC loop. If there is more than one FC loop in the system, the main RAID controller then launches this same process for each loop, one by one. It is worth noting that all these initialisation messages are sent through the signal path. In so doing, all devices and FC ports in the system get an FC address. In addition, the RAID controllers and the SBOD switches gather all the address and topology information of the system for later simulation use.

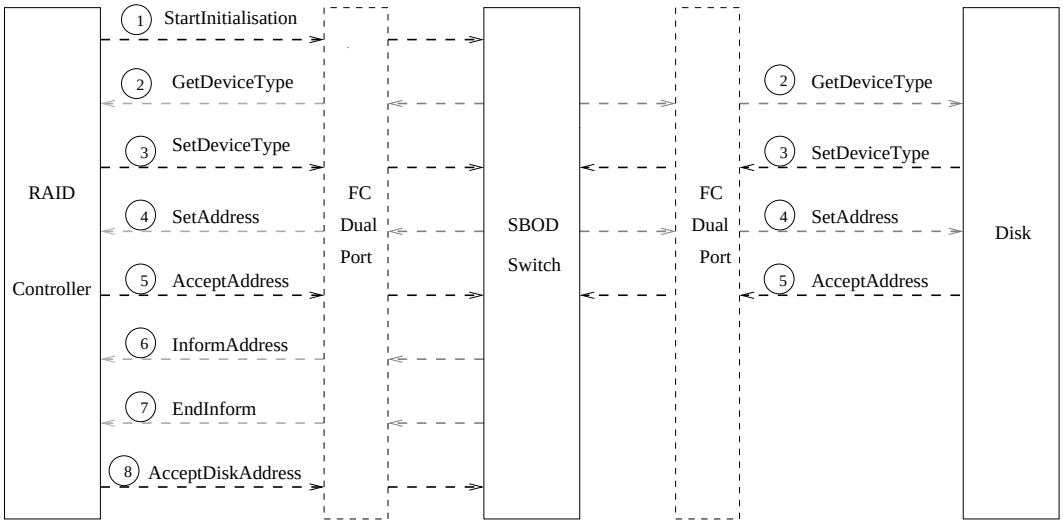


Figure 4.16: **Model Initialisation Process**

4.2.6 Summary of Model Enhancement

As stated at the beginning of this section, SIMRAID was initially developed for the Storlite project. This PhD project has improved and enhanced the model's functionality and performance. In summary, the enhancements to the model include:

- **Redesign and implemention of the RAID controller ENTITY.** The original RAID controller ENTITY had only the decomposition function. The current module has added more functionality to the controller model.

- **Redesign of the interconnection interface ENTITY.** The original interconnection interface was designed as a complex compound ENTITY. In this design each port was a separate ENTITY and ports were connected together by another ENTITY. This kind of design is less flexible and scalable than that of the current interface ENTITY. In addition, the current interface ENTITY reduces the

number of simulation events compared with the original, resulting in faster simulations.

- **Verification and correction of the communication protocol.** There was an error in the original communication protocol state transition design. If an FC port received an "OPN" when the queue of the device was full, it sent a "CLS" back to the initiator and "DeleteData" to the FC device. This design led to a false deletion of commands in the FC device and resulted in a segmentation fault. This error has been corrected in the SIMRAID model.

- **Debugging and verification of the whole model.** The SIMRAID model inherited from Storlite project was in the early stages of development and had not yet been debugged and verified. Each module of the model was therefore debugged and verified (and also ported from Linux 9 to Fedora Core). The RAID Controller and the Interface ENTITY were completely redesigned and the whole model then tuned and validated.

- **Improvements to the model's simulation speed.** This work included profiling and tuning the model, improving the effectiveness and efficiency of the simulation events scheduling class. After these changes, SIMRAID runs 4 times faster than the original model.

4.3 Simulation Process

The SIMRAID model goes through five phases during the simulation process: pre-simulation phase, model initialisation phase, start up phase, steady phase and termination phase. Table 4.3 describes these five phases in detail.

4.3.1 Simulation Results Collection

Simulation results are collected in the steady state. The average response time of the system is collected every minute. The final average response is the average of the all the collected response times, which consists of tens of simulation runs with standard deviation less than 0.5. Based on the system average response time, the maximum SPC-1 BSU that has an average response time of less than 30 ms is then obtained. In order to find the maximum number of SPC-1 BSUs for under 30 ms response time, short simulations are first carried out to find a BSU number for which the response

Phase	**Description**
Pre-Simulation	The model initialises its parameters by reading the values from the parameter file and performs some error checks. An error in this phase leads to abortion of the simulation
Model Initialisation	The model performs the initialisation protocol described in Section 4.2.5.1. In this phase each device is allocated an FC address. In addition, the system topology information is built in the RAID controllers and FC SBOD switches.
Start Up	Each component of the model executes its normal operation code. The benchmark generator sends out I/O requests and the rest of the model services these requests in accordance with its code. However, as the components have just started running, they have not yet reached their steady state, for example, the cache is not full yet.
Steady	Each component of the model executes its normal operation code and the model reaches the steady state. Measurements are taken to collect experiment results, for example the average response time and utilization of each component.
Termination	The model parameter file specifies a Termination_Time. Once the simulation time reaches this termination point, the model enters the Termination phase. Some statistical results from the model execution are reported.

Table 4.10: Phases of SIMRAID Simulation Process

time is close to 30 ms over a short period. Long simulations are then carried out to determine the average steady-state response time. To accelerate the searching process, binary search was used to find the maximum number of SPC-1 BSUs with response time not exceeding 30 ms.

4.4 Model Verification and Validation

SIMRAID has been verified and validated at two different levels. The first level involves verifying and validating the design and implemention of each component. The second level involves validating the whole simulation model against a real system at

the application level. This section first presents the verification results of the SPC-1 benchmark generator. It then presents the application level validation process and results.

4.4.1 Verification of the SPC-1 Benchmark Generator

Table 4.11 lists the statistics on the total number of I/O commands generated by each stream during a 60 second simulation. These figures are rounded to two digits. It can be seen that the maximum error in the number of I/O commands per stream is less than 3%. Table 4.12 lists the statistics on a portion of each command size for the stream that uses the SIM distribution. The maximum error in these statistics is less than 0.3%. Table 4.13 lists statistics on the write command fractions. These statistics also have a maximum error less than 3%. The above statistics show that the benchmark generator is highly accurate.

Streams	**Actual**	**Expected**	**Error (%)**
IOS1_1	19356	19320	0.19
IOS1_2	166733	166745	-0.01
IOS1_3	38650	38640	0.03
IOS1_4	124570	124613	-0.03
IOS2_1	10019	9935	0.85
IOS2_2	42419	41538	2.12
IOS2_3	19190	19320	-0.67
IOS3_1	154666	155112	-0.29

Table 4.11: Statistics on Total Number of I/O Commands per Stream

4.4.2 Application Level Validation

SIMRAID has been validated against an IBM Enterprise Storage Server (ESS) F20 system [19] at the application level. As shown in Figure 4.17, the F20 system contains 64 36 GB 15k RPM disks. These disks are connected to a RAID controller through four Serial Storage Architecture (SSA)[3] loops, with each loop containing 16 disks.

[3]SSA is a high performance, serial connection technology for disk drives. SSA is a full duplex loop based architecture, with two physical read paths and two physical write paths to every disk drive attached to the loop. Data is sent from the adapter card to the first disk drive on the loop and then passed

Stream	Command Size (blocks)	Actual (%)	Expected (%)	Error (%)
IOS1_3	8	39.81	40	-0.19
IOS1_3	16	24.24	24	0.24
IOS1_3	32	19.86	20	-0.14
IOS1_3	64	8.013	8	0.013
IOS1_3	128	8.080	8	0.08
IOS2_3	8	39.95	40	-0.05
IOS2_3	16	24.27	24	0.27
IOS2_3	32	19.98	20	-0.02
IOS2_3	64	7.806	8	-0.194
IOS2_3	128	7.983	8	-0.017

Table 4.12: Statistics on command size

IOS	Actual	Expected	Error(%)
IOS1_2	11651	11633	0.15
IOS1_4	8669	8694	-0.29
IOS2_2	3955	4057	-2.51

Table 4.13: Statistics on write commands fraction

Each SSA loop connects to the RAID controllers through two Device Adapters (DA). The total bandwidth of each loop is 320 MB/s. The RAID controller is coupled with a read/write cache and a Non-Volatile Storage (NVS) unit. Cache is used to store both read and write data to improve ESS performance as seen by the attached host systems. The NVS is used to store a second copy of write data to ensure data integrity in the case of a power failure or a RAID controller failure and the cache copy being lost. The F20 system operates as follows. When a read request arrives, if the data is in the cache, the data is sent back to the host straight away. Otherwise, a staging request is sent to the device adapter (DA) to fetch the requested data from disk. This read operation is the same as the read operation in the SIMRAID model. When a host sends a write request, it sends the data to the cache of one controller and the NVS of the other. Once the data is written to the cache, the request is considered completed. The cached copy

around the loop by the disk drives until it arrives at the target disk. Each read or write path on the loop operates at 40MB/s, providing a total loop bandwidth of 160MB/s.

of data will remain in the cache of the same cluster processor complex (CPC) until the LRU algorithm of the cache (of this CPC) or NVS (of the other CPC) determines that space is needed, and the data is scheduled to be destaged. All modified data for the same track is sent to the device adapter at the same time to maximize the destage efficiency. This write operation is also similar to the SIMRAID write operation under write back policy. The major difference that will affect the performance is that the IBM F20 system destages write commands when the cache line is replaced whereas SIMRAID does that immediately after the command is stored in the cache.

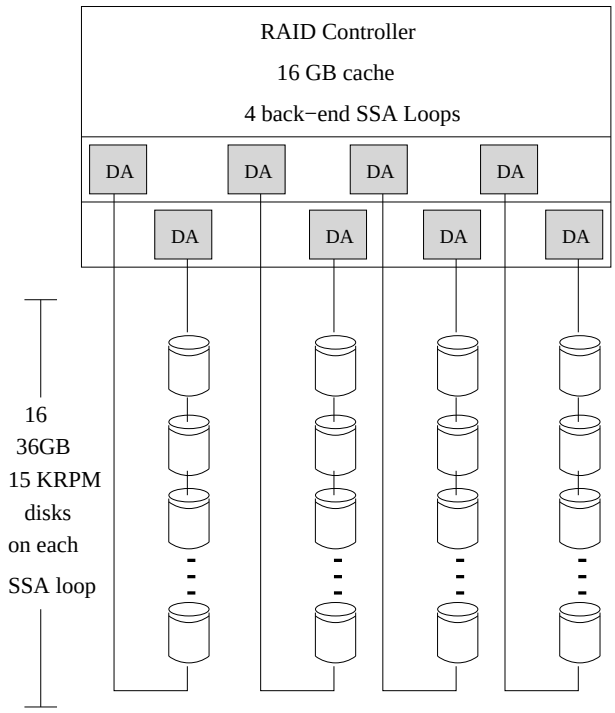


Figure 4.17: **Architecture of the IBM ESS F20 System** (from [19])

Since the system architecture and operation of SIMRAID are very close to IBM ESS F20, this system was chosen as the validation standard for SIMRAID. It is also worth mentioning that another reason to choose IBM ESS F20 system that the real system that SIMRAID is based on has not yet been completed. SIMRAID was initially built to evaluate Xyratex's new generation product, Samurai. However, due to some unforseen reasons, that project was terminated.

Although the system architecture and operation of SIMRAID are not totally identical to those of the IBM ESS F20 system, SIMRAID is very close to the IBM ESS 20 system in many aspects, including its RAID protection algorithm, disk features, cache type (both use mixed read and write cache) read/write operations and cache replace-

ment algorithm. Although SIMRAID models the FC_AL protocol rather than the SSA protocol, FC-AL working in MBOD mode is very similar to SSA in the way that each disk command has to pass through the loop. The major difference between the IBM ESS F20 system and SIMRAID is that the former destages write commands when the cache line is replaced while the latter does that immediately after the command is stored in the cache. The former has the advantage of sending a whole track that is in the cache to the disk at one time. Therefore, it can lead to a higher system performance.

Based on the IBM ESS F20 system architecture and its published SPC-1 results [3], SIMRAID is configured using the parameters listed in Table 4.14. The maximum SPC-1 BSU number of SIMRAID under such a configuration is 155, which is only 3.125% less than that of the IBM ESS F20 system. The different cache destaging algorithm is the main reason that SIMRAID performs less well than the F20 system. This validation confirms that SIMRAID is capable of modelling RAID systems accurately.

Parameter	**Value**
Total ASU capacity	1.12 TB
RAID type	RAID 5
Stripe unit size	16 KB
Cache size	16 GB
Cache policy	write back
Cache replacement policy	LRU
number of disks	64
number of loops	4
switch work mode	MBOD
FC port bandwidth	160 MB

Table 4.14: Total Number of I/O Commands per Stream

4.5 Model Performance Evaluation

A good simulation should be efficient in terms of both simulation speed and memory usage. This section presents the model performance evaluation results, including both simulation speed and memory requirement.

4.5.1 Simulation Speed

Figure 4.18 shows the simulation time multiplier (number of times real time for simulation run) with respect to workload and disk numbers for SIMRAID. It can be seen that workload is the main factor that affects the simulation speed. As the workload increases, the time multiplier increases linearly. Increasing the number of disks in the system also increases the time multiplier slightly, since it takes more time to synchronise these disks. For the simulation of a small size RAID system (7 disks) under full workload (namely maximum workload with average response time less than 30 ms), the time multiplier is 3.9. That of a large size RAID system (72 disks) is 52. Compared with other FC simulators, SIMRAID has sped up the simulation speed by a factor of 1000.

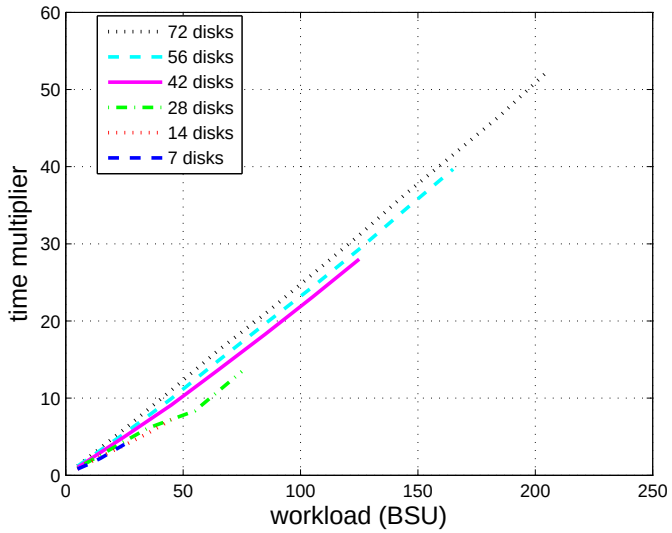


Figure 4.18: **Time multiplier vs. load for the system under test**

4.5.2 Model Memory Requirement

Figure 4.19 shows the physical RAM memory and virtual memory usage by SIMRAID with respect to workload and number of disks. These data are collected from the cluster job report. The physical memory used by SIMRAID is determined by the workload when the workload is small. However, as the workload increases, the physical memory usage will keep constant until the workload exceeds the full workload that the simulated system can serve. Increasing the number of disks increases the memory usage slightly, whereas the virtual memory used by SIMRAID is simply determined by the

number of disks. The virtual memory usage stays constant as the workload increases, unless the workload exceeds the full workload of the simulated system. The reason that the memory and virtual memory usage increase dramatically once the workload exceeds the full workload of the system is that once this happens, the volume commands that cannot get served in time are queued in the command store. This accumulation of unserved volume commands means that running SIMRAID needs more physical memory and virtual memory. For simulation of a large-size RAID system under full workload (namely maximum workload with average response time less than 30 ms), SIMRAID needs about 67 MB memory and 1.4 GB virtual memory, which is just a small portion of the memory of a modern desktop machine. Therefore, it is fair to say that SIMRAID is also very efficient on memory usage.

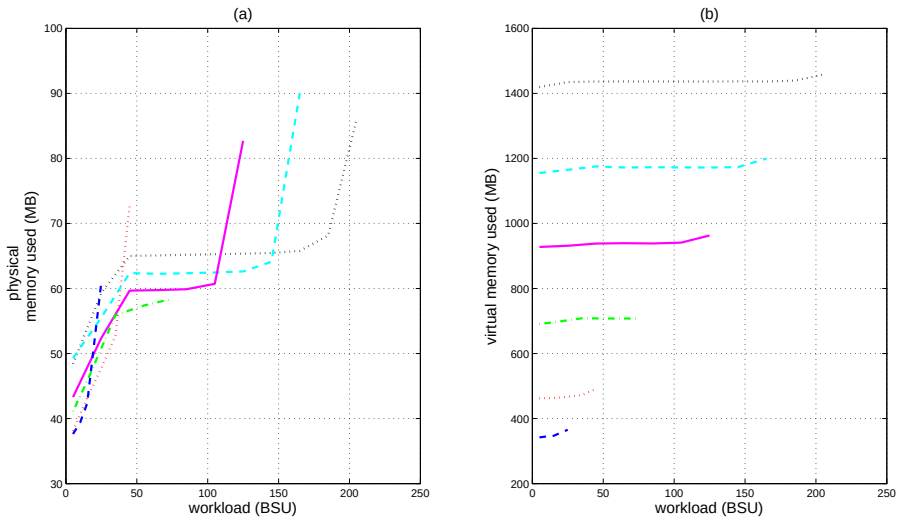


Figure 4.19: **Memory requirement vs. load of** SIMRAID**;** (a) physical memory requirement; (b) virtual memory requirement

4.6 Summary

This chapter has introduced the design and implementation of the SIMRAID simulation model. Firstly, an overview of the HASE model development environment and its facilities was presented. Secondly, the design and implementation details of the SIMRAID simulation model were presented. Thirdly, the simulation process of SIMRAID was described. Fourthly, model verification and validation result were pre-

sented. Lastly, the performance of the SIMRAID model was evaluated. The use of SIMRAID to study the scalability of the back-end network of the RAID system is presented in next chapter.

Chapter 5

Scalability of the Back-end Network

The design and implementation of the storage system simulator SIMRAID were described in Chapter 4. This chapter studies the scalability of the back-end network of storage sub-systems in terms of the number of disks that can be linked to the network. In particular, Fibre Channel (FC) Switched Bunch of Disks (SBOD) [30] has been chosen as the research subject, since it represents the current state of the art in scalable back-end RAID systems. This chapter aims to answer the following two questions: first, given a number of disks, which factors affect back-end network bandwidth requirements of disks; second, given an interconnection network, how many disks can be connected to the system. The contents of this chapter have been published in [59] and [58].

Due to the cost associated with making a dedicated test system, analytical and simulation approaches are used in this chapter. In the first approach, a queueing network model is built for storage sub-systems to study how the limited network bandwidth affects the performance of scaled-up systems under a small random workload. This analysis identifies the key factors that affect that network scalability. In addition to the queueing model, an equation to calculate the bandwidth needed to mirror write data is derived using a probability model. The purpose of the analytical study is to guide the simulation study rather than provide accurate results, since it is hard to predict disk behaviour under real-world workloads by using a purely analytical approach. In the second approach, simulations with various combinations of SIMRAID model parameters are conducted to study network scalability. The simulations are first configured with a network bandwidth that is higher than any currently available to study the factors that affect the bandwidth requirement for a given number of disks. After obtaining these results, the bandwidth is then reduced to study the saturation characteristics. Both

Iometer and SPC-1 benchmarks are used. Iometer represents a sequential workload environment, whereas SPC-1 represents a real-world Online Transaction Processing (OLTP) environment workload.

The rest of this chapter is organised as follows: Section 5.1 presents the analytical model; Section 5.2 provides the experimental methodologies; Section 5.3 presents the simulation results and discussions; lastly, Section 5.4 concludes the chapter.

5.1 Analytical Model

To simplify the analysis, the following assumptions are made in the analytical model:

- All disks have the same parameters. The Seagate Cheetah 15K.3 FC disk was chosen, since it is widely used in enterprise level RAID systems.
- The access address of each user request is aligned to the stripe unit boundary.
- Disk commands are uniformly distributed over all disks.

Figure 5.1 shows the queueing network model of a RAID system. To facilitate the analysis, a number of variables are defined (Table 5.1). In contrast to previous work, besides the conventional components such as controllers, caches and disks, the interconnection network delay is also included in this model. The controller is simply modelled as a $G/D/1$ queue. A fixed delay is added to every volume command. The controller cache is modelled as a splitter. Only disk write commands and disk read commands that miss in the cache are sent to the disks. The network is modelled as a $G/G/1$ queue with average service rate μ_1. The controller can use a dual port to connect to the disks, but for the purpose of simplicity, it is just modelled as one network; it is easy to predict the scalability of a dual port system once the result for a single port is known. Each disks is modelled as a $G/G/1$ queue with an average service rate μ_2. The aggregate service rate of the disk array is therefore $N\mu_2$. Obviously, the more disks in the array, the higher the bandwidth required. In order to support N disks, the network service rate μ_2 must be larger than the aggregate service rate of N disks. Thus, the maximum number of disks that a network can support to get the best performance is μ_1/μ_2. The disk service rate μ_2 depends on the disk command size, workload features and disk characteristics. Given that the disk transfer rate is 50 MB/s, it is easy to determine that a 2G FC port can support at most 4 disks under large sequential workloads, though it is not so straightforward for random access workloads. The following section will analyse the transmission process under a small random read workload in detail.

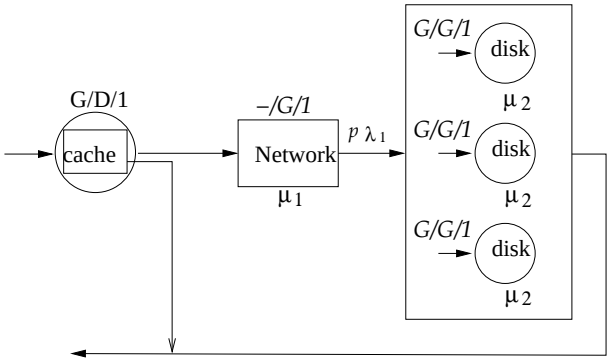


Figure 5.1: **Queueing network model of storage sub-systems**

Variables	Description	Values
N	number of disks	
D	number of data disks in a stripe	
M	number of parity disks in a stripe	
λ_1	volume commands arrival rate	
μ_1	network average service rate	
μ_2	disk average service rate	
p	cache miss rate	
V	size of a I/O request	(Bytes)
S	size of a stripe unit	(Bytes)
S_{ack}	size of the acknowledgement in the write mirroring	(Bytes)
a	alignment address	(Bytes)
R_x	transmission rate of the SBOD port	2.125 Gbps[a]
T_{dsk}	average seek time of disks	3 ms
T_{rl}	average rotation latency	2 ms
R_{dpt}	port transfer rate of disks	400 MB/s
R_{dm}	average disk media rate	50 MB/s
T_v	volume command completion time	
T_x	transmission time of a disk command	
T_d	disk service time for a disk command	

Table 5.1: Model Variable Definitions

[a]This is usually called 2G FC

5.1.1 Case Study - Small Random Access

For a random access workload, in order to improve the IOPS, the RAID systems will simultaneously serve as many I/O requests as possible. Two situations are considered, as shown in Figure 5.2. When there are not too many disks, the back-end network will be fast enough to send all disk commands to the disks before a disk command is completed. Thus, the number of simultaneous I/O requests being served is $\frac{N}{\lceil V/S \rceil}$, namely all data disks are serving I/O requests simultaneously. The average response time of a disk I/O request is $T_v = T_d + T_x$ (Figure 5.2 (a)). Thus, the IOPS of the system is $\frac{N}{\lceil V/S \rceil} * \frac{1}{(T_d + T_x)}$. If there are too many disks in the system, the network is still busy sending disk commands to other disks when a disk command is completed (Figure 5.2 (b)). The response time of a disk I/O request is then $T_v = N * T_x$. Due to the saturation of the network, some disks are idle; the number of active disks is $\frac{T_d}{T_x}$. Thus, the number of simultaneous I/O requests being served is $\frac{T_d}{T_x} * \frac{1}{\lceil V/S \rceil}$ and the IOPS of the system is $\frac{T_d}{T_x} * \frac{1}{\lceil V/S \rceil} * \frac{1}{N * T_x}$. In summary, the IOPS of the system is:

$$\textbf{IOPS} = \begin{cases} \frac{1}{T_d + T_x} * \frac{N}{\lceil V/S \rceil} & N * T_x < T_d \text{ or } N = 1 \\ \frac{T_d}{T_x} * \frac{1}{N * T_x} * \frac{1}{\lceil V/S \rceil} & N * T_x \geq T_d \text{ and } N > 1 \end{cases} \tag{5.1}$$

where $T_d = \frac{S}{R_{dpt}} + \frac{S}{R_{dm}} + T_{dsk} + T_{rl}$, and $T_x = \frac{10S + 22 * 40 * S/2048}{R_x}$ [1]

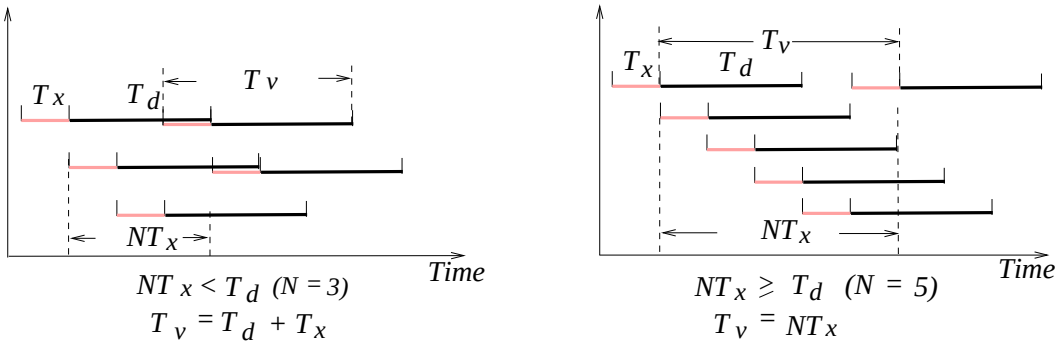


Figure 5.2: **Transmission process of disk commands** (a) depicts the situation that $NT_x < Td$; (b) depicts the situation that $NT_x \geq T_d$

[1] $10S$ is the length of an encoded S-Byte disk command in bits;
$22 * 40 * S/2048$ is the length (in bits) of the frame headers and idle words between frames to transmit an S-Byte disk command.

Figure 5.3 depicts the IOPS with respect to the number of disks in a system under 8 KB random read workloads with the stripe unit size equal to 8 kB, 16 kB, 32 kB and 64 kB respectively. It can be seen that as the number of disks increases, the IOPS will increase until the number of disks reaches 136, 71, 41 and 21 respectively. The selection of the stripe unit size has a significant effect on network scalability. A smaller stripe unit size allows a larger system size. Nevertheless, compared with a sequential workload, more disks can be accommodated under a random workload with the same network. Because there is a seek and rotation process for random accesses, the disk service rate for random accesses is lower than that for sequential accesses.

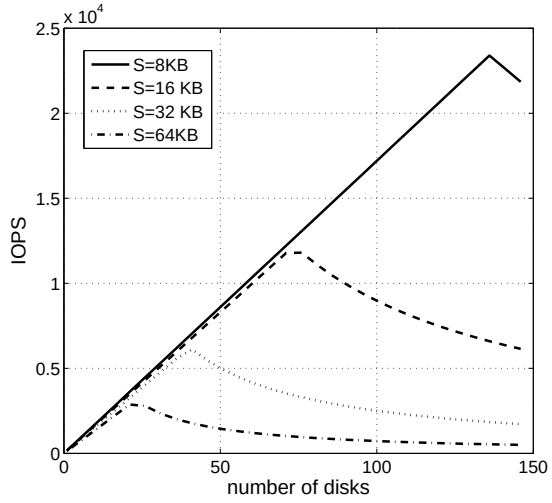


Figure 5.3: **IOPS vs. number of disks under random read workloads** $bandwidth = 2.125 Gbps$ and V = 8 kB.

The above analysis shows that the more disks there are in a RAID system, the more network bandwidth is required to support the system. Moreover, workload feature and stripe unit size have a significant effect on the bandwidth requirement. However, under a synthetic workload, it is difficult to predict the number of disk commands that can be served by a RAID system. Simulations were therefore carried out to find out the relationship between the system size and bandwidth requirement under an SPC-1 benchmark workload. Furthermore, the effect of cache size and RAID algorithm have also been explored through simulation.

5.1.2 Bandwidth for Write Mirroring

In a system with two controllers, write data mirroring can be used to prevent data loss (Section 2.2.2.4). Since a write-back cache policy is usually employed in modern RAID systems to improve performance, this section analyses the network bandwidth required to mirror write data under the write-back policy.

There are four types of data transmitted between the two controllers:

- Volume-level write commands;
- Parity units for each stripe of the write commands;
- Command completion messages of the write commands;
- Acknowledgements of the above messages;

Given the distribution of the command sizes, the average size of a volume-level write command is:

$$\overline{\mathbf{V}} = \sum_{i=0}^{n} V_i \times P_i \tag{5.2}$$

where P_i is the distribution of the sizes of volume-level write commands.

The average number of parity units is determined by the average number of stripes in each write command:

$$\overline{\mathbf{N_s}} = \sum_{i=0}^{S/a-1} \frac{1}{S/a} \times \left(\sum_{j=0}^{D-1} \frac{1}{D} \times \left\lceil \frac{j+Nd_i-1}{D} \right\rceil + 1 \right) \tag{5.3}$$

Thus, the average number of parity units for each volume-level write command is:

$$\overline{\mathbf{N_p}} = M \times \overline{N_s} \tag{5.4}$$

The command completion message and all acknowledgement messages have the same length S_{ack}. Thus, the average mirroring data for each volume-level write command is:

$$\overline{\mathbf{K_m}} = \overline{V} + M \times \overline{N_s} + S_{ack} \times (\overline{N_s} + 3) \tag{5.5}$$

Taking the FC transmission overhead into account, for a workload that generates W volume commands per second, the bandwidth to transmit the write mirroring data is:

$$40W \cdot P_{write} \times \left(22 \times \frac{\overline{K_m}}{2048} + \frac{\overline{K_m}}{4} \right) \tag{5.6}$$

where P_{write} is the fraction of volume-level write commands.

Since the data to be transmitted between two controllers is independent of the communication network, equation 5.5 can also be applied to compute the communication cost for other types of network.

5.2 Experimental Set Up

This sections describes how experiments were set up to investigate network scalability. Firstly, the parameters that might affect the network scalability are discussed. Secondly, the system architectures used in the experiments are introduced. Lastly, the experimental procedure is presented.

5.2.1 Parameters of Interest

Based on the analysis in section 5.1, the experiments explored the effects of the following four parameters:

- **Workload feature.** Simulations were conducted under Iometer benchmark and SPC-1 benchmark respectively. As described in section 4.2.4.1, the Iometer represents a sequential access workload, and the SPC-1 benchmark represents an OLTP environment workload.
- **Cache size.** The cache was increased from zero (no cache) to 16 GB to investigate the effect of increasing cache size.
- **Stripe unit size.** Five conventional stripe unit sizes: 4 KB, 8 KB, 16 KB, 32 KB and 64 KB, were tested.
- **RAID Algorithm.** The bandwidth requirements of RAID 5 and RAID 6 were studied.

5.2.2 System Architecture

Experiments were first carried out using architectures containing only one RAID controller with a single FC port and one SBOD to study the scalability of the back-end network in systems without write mirroring (Figure 5.4 (a)). Later experiments used architectures containing two controllers with a dual FC port connected to each of them and two switches to study the bandwidth requirement for write data mirroring (Figure 5.4 (b)). The system size was expanded by increasing the numbers of SBOD ports

and disks. The experiments began with a network bandwidth higher than any currently available, to investigate the effects of aforementioned parameters on bandwidth requirements of disks. The bandwidth was then reduced to study saturation characteristics. Table 5.2 lists the setting of the parameters that were fixed throughout the experiments.

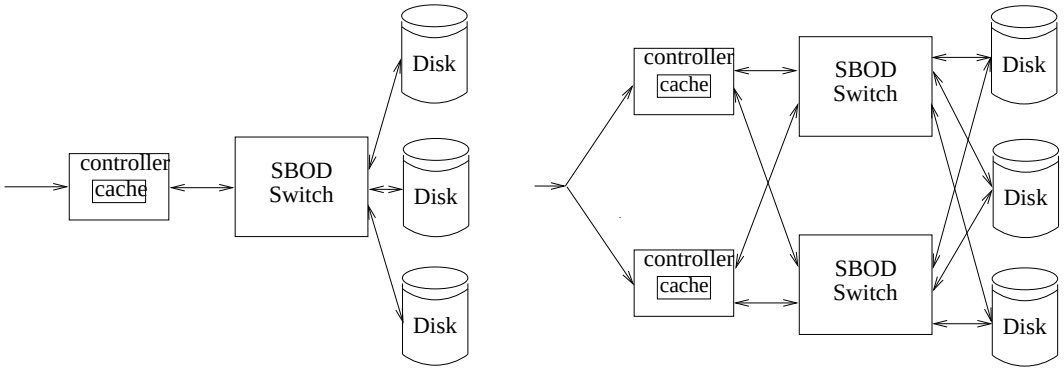


Figure 5.4: **Architectures for experiments** In (a) the controller communicates with disks through a single FC port and one SBOD; in (b) dual FC port and two SBODs are used. The two controllers communicate with each other and disks through the FC port and SBODs.

Parameters	Value
file capacity	2 TB
cache policy	write back
cache replacement policy	LRU
maximum disk queue length	20

Table 5.2: Parameter Settings in the Experiments

5.2.3 Experimental Procedure

For each set of input parameters, a sequence of performance measurements is generated in a single simulation run. Since only steady-state performance is interesting, the initial transient phase from the sequence is eliminated. The remaining steady-state sequence is partitioned into subsequences. The performance measurements presented in this dissertation are the averages of the subsequence mean estimates, which consist of tens of simulation runs with standard deviation less than 0.5.

In order to find the maximum number of SPC-1 BSUs for under 30 ms response time, short simulations were first carried out to find a BSU number for which the response time is close to 30 ms over a short period. Long simulations were then carried out to determine the average steady-state response time. To accelerate the searching process, binary search was used to find the maximum number of SPC-1 BSUs with response time not exceeding 30 ms.

5.3 Simulation Results

This section presents the simulation results. Firstly, the factors that affect the back-end network bandwidth requirement of the disks are investigated. Secondly, the scalability and saturation characteristics of the back-end network are discussed. Lastly, simulation results on bandwidth required for write data mirroring are presented, as well as results on network scalability.

5.3.1 Factors Affecting Bandwidth Requirement

This section studies the effect of four parameters on the back-end network bandwidth requirement of disks.

5.3.1.1 Effect of Workload Feature

The impact of workload feature on bandwidth requirement was first investigated. Figure 5.5 (a) shows the network bandwidth requirement under the Iometer and SPC-1 benchmark workloads. It can be seen that for the Iometer workload, when the system is smaller than 16 disks, each disk requires about 0.5 Gbps of network bandwidth. This confirms the analysis in Section 5.1. With a 50 MBps data transfer rate and the 8B/10B encoding method, 1 Gbps is able to support two disks. Due to the fact that Iometer keeps the outstanding IOs fixed at 75, it is not able to keep the disks 100% busy when there are too many disks. Moreover, because a disk command received in the idle state takes longer to complete, in these simulations the bandwidth requirement and performance begin to level out at 16 disks. For SPC-1, each disk requires only about 0.2 Gbps bandwidth. Figure 5.5 (b) shows that for both cases the disk utilisation is close to 1. The reason that the disks require less bandwidth under the SPC-1 workload is that for most of the time the disks are seeking, not transmitting, which is not the

case for the sequential Iometer workload. Since it is easy to compute the bandwidth requirement for Iometer, the rest of this chapter will only discuss the SPC-1 workload.

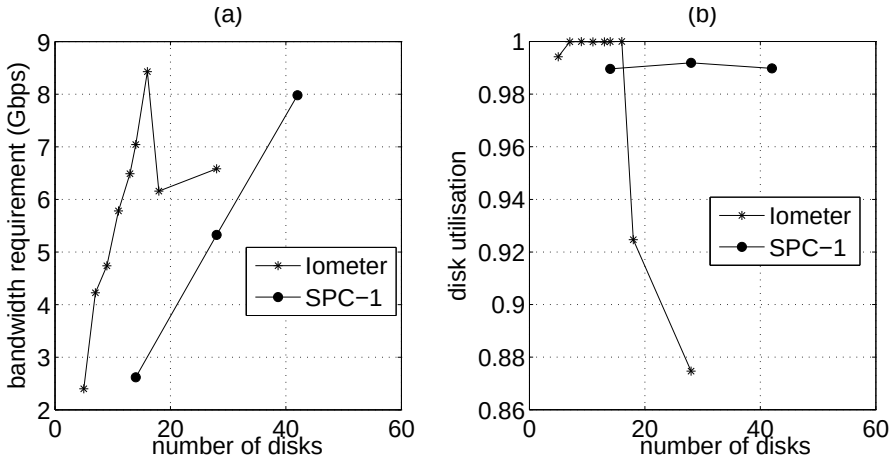


Figure 5.5: **Effect of workload** Stripe unit size = 64 KB. (a) bandwidth requirement; (b) disk utilisation.

5.3.1.2 Effect of Cache Size

Figure 5.6 shows the effect of cache size on the network bandwidth requirement with respect to the number of disks in the system. It can be seen from Figure 5.6 (a) that systems without any cache require less network bandwidth than systems with cache, whereas there is no significant difference between systems with 2 GB caches and systems with 16 GB caches. This can be explained by the disk utilisation shown in Figure 5.6 (b). When there is no cache in the controller, in order to meet the SPC-1 maximum 30 ms average response time requirement, disks can only be 90% busy, whereas systems with cache are able to provide service with an average response time less than 30ms with the disks 100% busy. When the disks are 100% busy, given a certain number of disks, the number of disk commands they are able to serve is determined by the workload and stripe unit size. Therefore, once the disks are 100% busy, adding more cache only increases the system performance, but does not increase the back-end network bandwidth requirement.

5.3.1.3 Effect of Stripe Unit Size

Figure 5.7 (a) shows the back-end network bandwidth requirement of cached systems under a number of conventional stripe unit sizes with respect to system size. The reason to use cache is to drive the disk utilisation close to 100%, as shown in Figure 5.7 (d).It

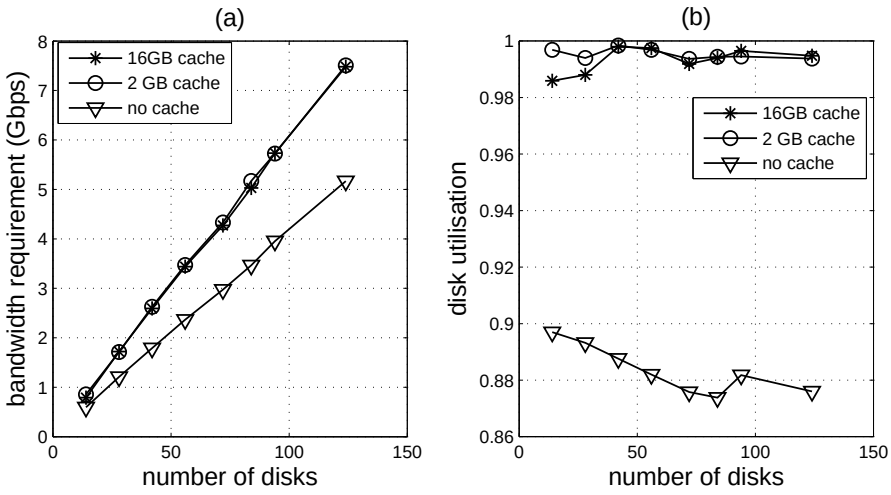


Figure 5.6: **Effect of cache size** Stripe unit size = 16 KB; RAID level = RAID 5; workload = SPC-1 benchmark workload.

can be seen that the stripe unit size has a significant effect on the bandwidth requirement. Systems with larger stripe unit size require more back-end network bandwidth to support the system. This is because a large portion of the SPC-1 benchmark workload is 4 KB random access, as described in Section 4.2.4.1. This random access is decomposed into one disk command in the case of a read access or four disk commands in the case of a write access. As the number of disk commands transmitted across the back-end network is the same, a large stripe unit size requires more bandwidth. However, regarding performance, 32 KB rather than 4 KB is the optimal stripe unit size, as shown in Figure 5.7 (b). This is because with large stripe unit size, each disk command brings more data into the controller, leading to a higher read hit rate (Figure 5.7 (c)). Nevertheless, if the stripe unit size is too large, *e.g.* 64 KB, the disks spend much more time processing disk commands, leading to some reduction in performance. Therefore, 32 KB is a good performance balance point.

5.3.1.4 Effect of RAID Algorithm

Figure 5.8 shows the effect of the RAID algorithm. It can be seen from Figure 5.8 (a) that in systems without any cache, RAID 5 needs more bandwidth to support the same number of disks, whereas if systems contain enough cache, there is no significant difference in the bandwidth requirement of RAID 5 and RAID 6. This can also be explained by the disk utilisation, shown in Figure 5.8 (b). As described in Chapter 2, RAID 5 takes four disk accesses for a small write request, whereas RAID 6 takes six

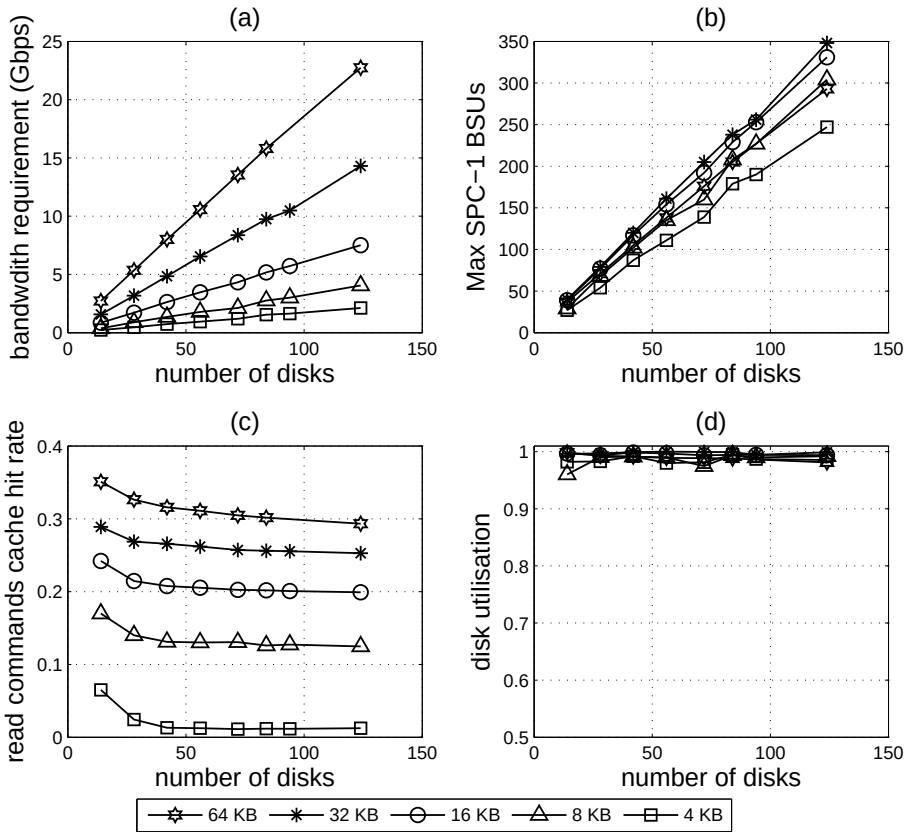


Figure 5.7: **Effect of stripe unit size** Cache size = 2GB; RAID level = RAID 5; Workload = SPC-1 benchmark workload.

disk accesses. In order to meet the SPC-1 maximum 30 ms response time requirement, RAID 6 has to keep the disks less busy than RAID 5. Therefore, fewer disk commands are sent to disks in RAID 6 systems than in RAID 5 systems. Thus, less back-end network bandwidth is required. However, if the system contains enough cache, *e.g.* a RAID 5 system with 2 GB cache and a RAID 6 system with 8 GB cache, the disks in both RAID 5 systems and RAID 6 systems work 100% busy. Given the stripe unit size and workload, the same number of disks is able to serve the same number of disk commands no matter whether the system is configured as RAID 5 or RAID 6. Therefore, there is no significant difference between the back-end network bandwidth requirement for RAID 5 and RAID 6. The reason that RAID 6 with 2 GB cache needs slightly less bandwidth is that a 2 GB cache is not enough for a RAID 6 system to provide service with maximum 30 ms average response time while disks work 100% busy.

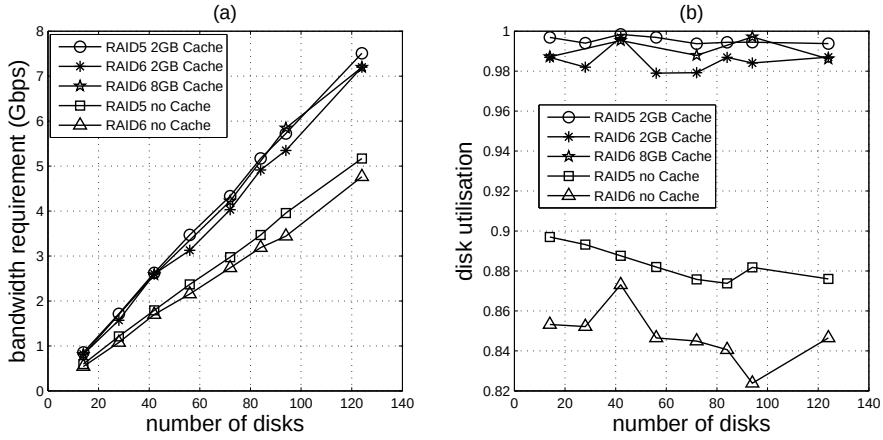


Figure 5.8: **Effect of RAID Algorithm**

5.3.2 Network Saturation Characteristic

Figure 5.9 shows the scalability and saturation characteristics of RAID systems with 2G FC networks. It can be seen that 11, 16, 32, 64 and 118 disks are the saturation points for 64 kB, 32 kB, 16 kB, 8 kB and 4 kB stripe unit size respectively. After the system size reaches this point, there will be no further gain in the Max SPC-1 BSU number by adding more disks to the system. Before the network saturates, the disk utilisation is close to 1. Adding more disks to the system will increase the network utilisation. Once the network utilisation is close to 1, adding more disks to the system only reduces the aggregate disk utilisation. Therefore, the system cannot serve

more requests. However, at this point, adding disks has a good side effect of speeding up the corresponding response speed. The simulation results shows that after the network saturates, adding disks to the system will cause the response time to decrease (see Figure 5.10). This figure shows an instability because of measurement inaccuracy in experiments. The average response reported in this figure is the maximum average response time that does not exceed 30 ms. Due to small experiment errors and measurement inaccuracy, this maximum average response time measured for different experiments which uses different number of disk shows an instability. These saturation points are smaller than the analytical result for the small random accesses. This is because SPC-1 is a mixed workload of 38.6% sequential access and 71.4% random access, and the disks serve requests faster than under pure random access workload. Therefore, under an SPC-1 benchmark workload fewer disk can be accommodated in the same network. Moreover, a smaller stripe unit size has better network scalability and overall system performance. By comparing Figure 5.7 (b) and Figure 5.9, it can seen that when there is no back-end network bottleneck, 32 KB is the optimum stripe unit size. However, given 2G bandwidth, 4 KB stripe unit size achieves maximum system performance. Considering that for the near future back-end network costs will be greater than disk costs, smaller stripe unit sizes should be considered in designing scalable storage systems.

Figure 5.11 shows the scalability of a 4G FC[2] network in a cached system with 16-kB stripe units compared to that of a 2G FC network. It can be seen that when the network is not saturated, using 4G FC instead of 2G FC does not lead to any obvious increase in system performance, because the disk performance determines the overall system performance. However, the saturation point of a 4G FC network is nearly twice of that of a 2G network. The maximum performance that a 4G network can achieve is therefore twice that of a 2G network.

5.3.3 Bandwidth Requirement for Write Data Mirroring

Figure 5.12 depicts the bandwidth required to transmit write mirroring data, including both simulation results and analytical results. This bandwidth requirement includes the bandwidth to transmit and receive write mirroring data. The simulation results match closely with the analytical results. Due to the small inaccuracy of simulation model,there a small difference between the simulation results and analytical results.

[2]The bandwidth is 4.25 Gbps

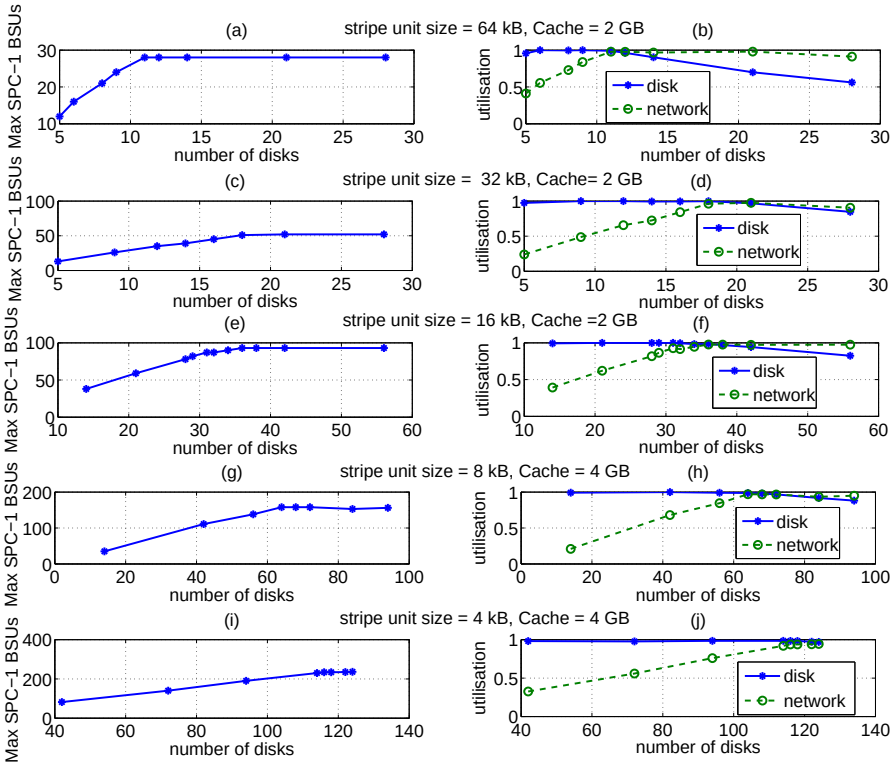


Figure 5.9: **Scalability of 2GFC network in cache system** Left hand side graphs depict the maximum number of SPC-1 BSUs for 30 ms response time with respect to the number of disks; right hand side graphs depict the corresponding average disk utilisation and network utilisation with respect to the number of disks.

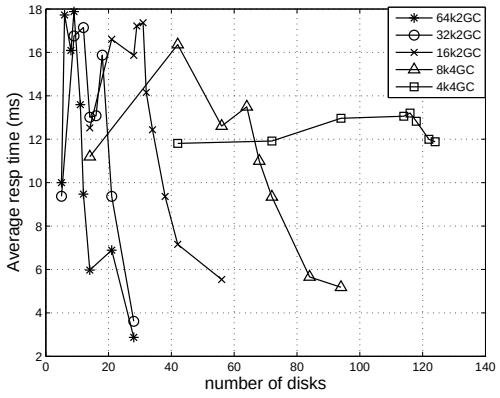


Figure 5.10: **Response time VS number of disks** Network bandwidth = 2.125 Gbps. Legend '64K2G' means that the stripe unit size is 64 KB and the cache size is 2 GB.

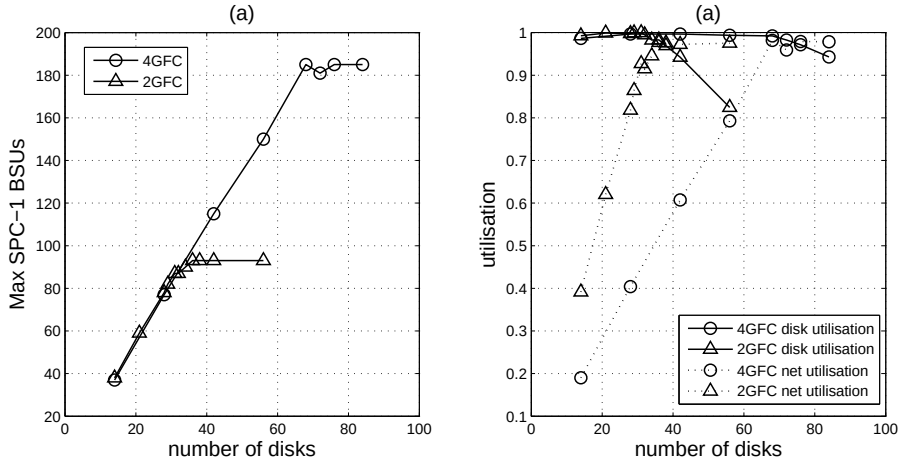


Figure 5.11: **Scalability of 4GFC network in cache system** Size of stripe unit = 16 kB, cache = 2 GB. (a) depicts the maximum number of SPC-1 BSU for 30 ms response time with respect to the system size in disks; (b) depicts the corresponding overage disk utilisation and network utilisation with respect to the system size in disks.

Note that the result represents the minimum bandwidth requirement for the system with larger cache size. This is because adding more cache requires more data to be mirrored, which in turn requires more bandwidth. Nevertheless, it provides a useful way to estimate the system bandwidth requirement.

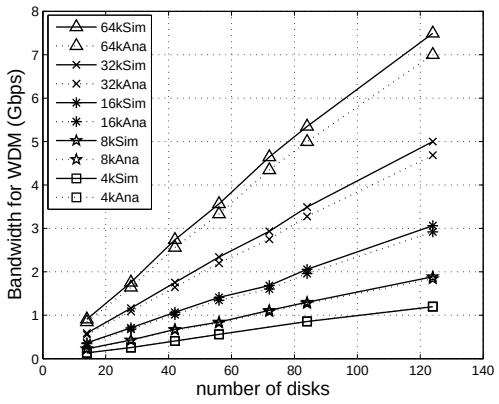


Figure 5.12: **Bandwidth requirement for write data mirroring in RAID5 systems**, Cache size = 2 GB, legend '64kS' means the simulation results for 64-kB stripe unit size. '64kA' refers to the analytical results for 64-kB stripe unit size.

5.3.4 Network Scalability in Systems with Write Data Mirroring

Figure 5.13 shows the network scalability of a 2G FC dual port SBOD[3] in a system with write data mirroring. Fixed port selection and alternating selection mechanisms are considered. In the former mechanism, each controller sends FC words through only one of the dual ports and the other port is used to receive mirroring data, while in the latter mechanism, the controller will send commands through each of the dual ports alternately. As a result, the former scheme leads to the four FC ports not being fully used, and the number of disks in a system is restricted to 11, 28, 52 and 84 with stripe unit size being 64 kB, 32 kB, 16 kB and 8 kB respectively. The 4 kB case is not shown in Figure 5.13; for 4 KB stripe unit size the system size can be up to a full loop size of 124 for both port selection mechanisms (FC-AL can only support 126 devices in a loop; as the two RAID controllers use two ports, only 124 disks are in a full loop). In contrast, the latter scheme allows the number of disks to be 21, 38, 64 and 118 respectively. The results also show that even under this scheme, the utilisation of each port of the two controllers can only be about 85% due to contention among ports.

5.4 Summary

This chapter has studied the scalability of the RAID system back-end network through analytical and simulation approaches. The analysis identifies the key factors that affect that network scalability and provide guideline for the simulation study. In particular, it has solved two problems: (1) it has determined the factors that affect the back-end network bandwidth requirement of the storage system; and (2) it has identified the network scalability and saturation points for a number of conventional system configurations.

The major results achieved include:

- The bandwidth requirement per disk is primarily determined by workload features and stripe unit size. Cache size and RAID algorithms have very little effect.
- The number of disks that a system can scale to is limited by the back-end network bandwidth. A smaller stripe unit size has better scalability than a larger one.

Table 5.4 lists the network bandwidth requirements and the maximum number of disks that a 2G FC port can support for a number of conventional system configu-

[3] In total there is 4*2.125 Gbps bandwidth available for controllers as each controller connects with a dual FC port.

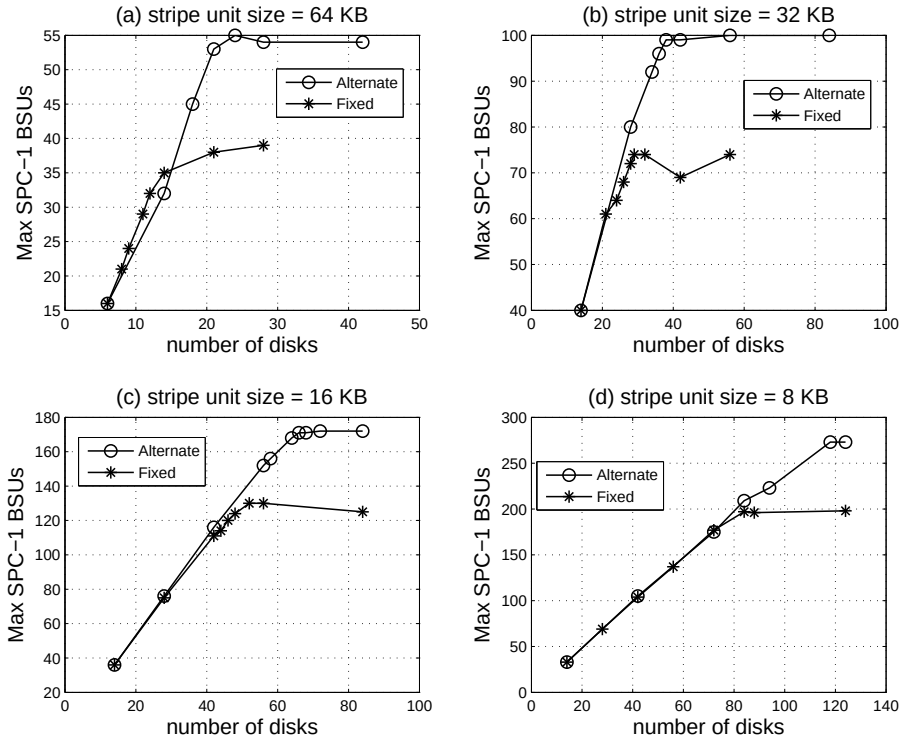


Figure 5.13: **Scalability of 2G FC dual port SBOD in systems with write data mirroring** Cache size = 2 GB. The network bandwidth available is 8 Gbps since each controller is connected to a 2G FC dual port.

rations. Considering that the workloads represent real-life workloads, such accurate simulations provide practical guidance for RAID system design.

Stripe unit size	Without mirroring		With mirroring	
	bandwidth requirement per disk (Gbps)	disks supported by each 2G FC	bandwidth requirement per disk (Gbps)	disks supported by each 2G FC [a]
64KB	0.1913	11	0.3146	5
32KB	0.1135	16	0.1967	9
16KB	0.0602	32	0.1114	16
8KB	0.0311	64	0.0638	29
4KB	0.0171	122	0.0367	31

Table 5.3: Network Scalability Summary

[a] This is what a 2 Gbps FC can support. The values shown in Figure 5.13 are 4 times larger than these values since it displays the number of disks that two 2Gbps FC dual ports can support.

Chapter 6

Improving the Reliability of Large Scale RAID systems

Chapter 5 studied the scalability of RAID systems from the performance perspective. In particular, it studied the scalability of the back-end network. This chapter studies the scalability of RAID systems from the reliability perspective. In particular, it proposes several approaches to improve system reliability and scalability, including a novel parity declustering data layout that can survive dependent disk failures caused by physical interconnect failures or correlated disk failures, a system architecture and rebuilding mechanism for fast disk reconstruction, and an efficient distributed hot spare allocation algorithm for general parity declustering data layouts.

As introduced in Chapter 1, physical interconnect failures and correlated disk failures cause multiple disks to go missing from the system. If the system is simply organised using a RAID 5 or RAID 6 data layout, the system will lose data service when such failures happen. An orthogonal RAID system, which maps reliability groups orthogonally to the physical interconnect, so that only one disk is affected when such a physical failure occurs, can survive physical interconnect failures. However, it has very bad degraded and rebuilding performance. Parity Declustering data layouts offer good degraded and rebuilding performance by setting the number of disks within a RAID group to be larger than the stripe width and distributing stripes over all the disks. This chapter thus presents a parity declustering data layout called Parity deClustering Data layout for Surviving Dependent disk Failures (PCDSDF). This data layout combines the advantages of both parity declustering and orthogonal data layouts.

As system reliability is inversely proportional to the disk reconstruction time, improving the disk reconstruction speed leads to a higher system reliability. Although

there has been a variety of research carried out on disk reconstruction algorithms, ways to improve the disk reconstruction speed from the system architecture aspect have not been studied. This chapter describes the design and evaluation of a system architecture for fast disk reconstruction.

Lastly, to solve the write speed bottleneck during disk reconstruction under general parity declustering data layouts, an efficient distributed hot spare allocation algorithm for general parity declustering data layouts is proposed[1]. This algorithm avoids the conflict and adjustment problems of assigning distributed hot spare space to data units that exist in previous algorithms.

The remainder of this chapter is organized as follows: Section 6.1 presents the PCDSDF data layout; Section 6.2 describes the system architecture and rebuilding mechanism for fast disk reconstruction; Section 6.3 presents a distributed hot spare allocation algorithm for general parity declustering data layouts; Section 6.4 describes the experimental set up; Section 6.5 presents the performance evaluation results achieved by using simulation on SIMRAID; Section 6.6 summarises the chapter. To facilitate discussion, Table 6.1 defines the variables used in this Chapter.

Variable	**Definition**
n	Total number of disks
k	stripe width, namely the number of disks per stripe
f	number of redundant stripe-units per stripe
α	Declustering ratio, $(k-f)/(n-f)$
D	Disks per shelf
L	Number of shelves

Table 6.1: Variable Definitions

[1]NetApp^{TM}is applying for patents for the PCDSDF data layouts and the distributed hot spare allocation and assignment algorithm.

6.1 Parity Declustering Data Layout For Surviving Dependent Disk Failures

This section presents a novel parity declustering data layout called Parity deClustering Data layout for Surviving Dependent disk Failures (PCDSDF)[2]. This data layout combines the advantages of both parity declustering and orthogonal data layouts - it can not only survive dependent disk failures resulting from physical interconnect failures or correlated disk failures, but also has a good degraded-mode and rebuilding performance. The generating process of PCDSDF is simple, deterministic and efficient in terms of both storage and time. The size of a rotation, which is the minimum number of stripes that form an evenly balanced parity declustering data layout, is small.

To simplify the discussion, shelf failure is used as an example of a dependent disk failure. In addition, for simplicity, f is restricted to 1 in the following discussion. It is also assumed that each shelf contains the same number of disks. To survive a shelf failure, the disks that participate in a stripe must be selected from different shelves. The total number of such combinations is $\binom{L}{k} \times D^k$, where L is the number of shelves, k the stripe width, and D the number of disks contained in each shelf. When L, k and D are large, this number is very large. The proposed PCDSDF data layout reduces the number of stripes required per rotation and its generation is deterministic. This section first describes the simple PCDSDF scheme which works for the scenario where the number of shelves L is equal to the stripe width k, k is not larger than the number of disks in each shelf D, and D is a prime number. Following that, the complete PCDSDF is presented; this works for any disk array configuration.

6.1.1 Simple PCDSDF

First, a simple case in which $k = L$, $k \leq D$ and D is a prime number is considered. A novel block design technique called *index permutation* is developed to combine disks from different shelves together. In the data layout generated by this block design, for each rotation, each disk shares the rebuilding workload with all disks that are not in the same shelf. In addition, any two disks appear in the same stripe only once in one rotation. The simple PCDSDF block design process contains D iterations and each iteration contains D sub-block designs. Assume that shelves are numbered from 0 to

[2] In this dissertation both disk failures caused by physical interconnects failures or correlated disk failures are defined as dependent disk failures

$L-1$ and the disk in each shelf has an index number from 0 to $D-1$ (the real disk number of the jth disk in the ith shelf is $i*D+j$). The principle of the design is that for the ith iteration and the jth sub-block design, the $(j+m*i)\%D$th disk (namely the disk with index number equals to $(j+m*i)\%D$) is selected from shelf m.

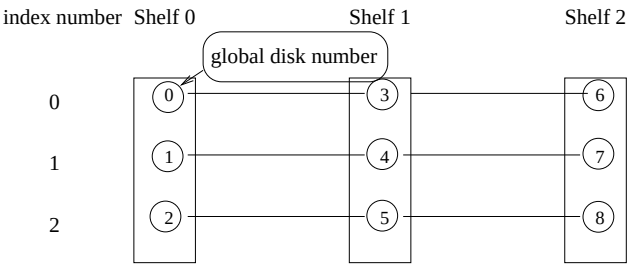
Figure 6.1 illustrates the process of the simple PCDSDF block design. It gives an example where there are three shelves and each shelf contains three disks, as shown in Figure 6.1 (a). Each disk has a index number within a shelf and a global disk number. Index numbers are numbered from 0 to $D-1$ and the global disk number are numbered from 0 to $L*D-1$.The objective of the block design is to create a data layout in which stripe units that form a stripe are selected from three different disks and each of these disks is from a different shelf so that it can survive both a single disk failure and a single shelf failure. Figure 6.1 (b) demonstrates the aforementioned index permutation technique. As seen in Figure 6.1, the block design includes three iterations and each iteration generates the layout for three stripes. The right side of Figure 6.1 (b) lists the disk index number and disk number that the stripe resides on. In the first iteration, when i is equal to 0, disks with the same index number are combined to form a stripe group. The index number combinations of the disks that the stripes reside on are (0, 0, 0), (1, 1, 1) and (2, 2, 2) respectively. In the second iteration, when i is equal to 1, disks that have the index number of disks from adjacent shelves differing by 1 are combined together. Namely it generates disk combinations (0, 1, 2), (1, 2, 0) and (2, 0, 1) respectively. In the third iteration, when i is equal to 2, the index number of disks from adjacent shelves to be combined together differ by 2. Namely it generates disk combinations (0, 2, 1), (1, 0, 2) and (2, 1, 0) respectively. To tell the difference between disks of difference shelf, the index number shown the the block design are denoted as shelf_number.index_number. Figure 6.1 (c) shows the data layout of a rotation generated by associating stripes with those combinations allocated to the first available stripe unit (*i.e.* one that has not been allocated to any stripe) on the disk to the corresponding stripe.

Algorithm 3 describes the process of the simple PCDSDF block design. The input to the algorithm is the number of disks per shelf D and the number of shelves L. The output of the algorithm is a two-dimension array *index_vector* which stores the block design. Each array element is an array of disk index numbers, which are the index numbers of disks on which each stripe resides.

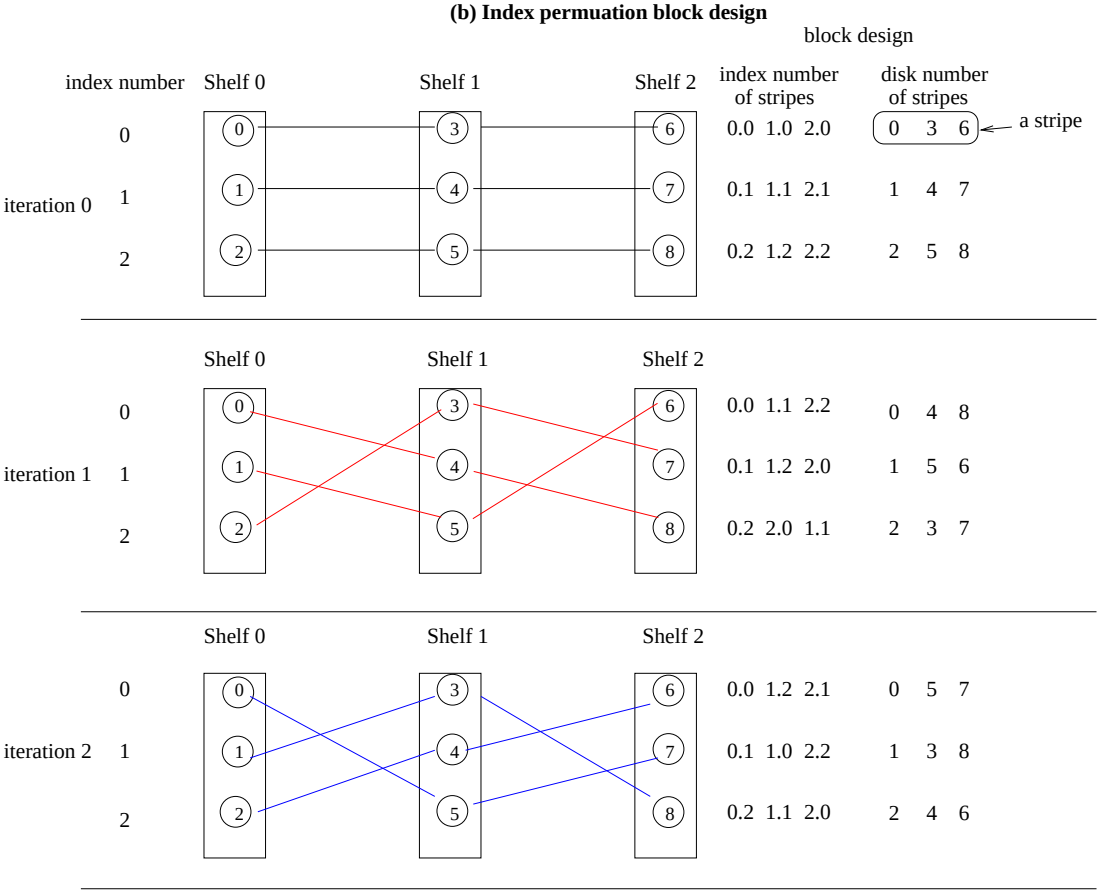
The simple PCDSDF has the following properties:

Theorem 1.

(a) Architecture of disk array



(b) Index permuation block design



(c) Data Layout on Disk

Disk	Shelf 0			Shelf 1			Shelf 2		
	0	1	2	3	4	5	6	7	8
	S0	S1	S2	S0	S1	S2	S0	S1	S2
	S3	S4	S5	S5	S3	S4	S4	S5	S3
	S6	S7	S8	S7	S8	S6	S8	S6	S7

Figure 6.1: **Simple PCDSDF block design process and data layout** $L = k = 3, D = 3$

Algorithm 3 Simple_PCDSDF (D, L, $index\_vector[D*D][L]$)

```
1: for i = 0 to D − 1 do
2:   for j = 0 to D − 1 do
3:     for m = 0 to L − 1 do
4:       index_vector[j + D ∗ i][m] ⇐ ( j + i ∗ m)%D
5:     end for
6:   end for
7: end for
```

(i) The number of stripes in each rotation is $D*D$ and each disk contains D stripe units.

(ii) The number of stripes in each rotation is the minimum number of stripes required to achieve a balanced rebuilding workload.

(iii) When D is a prime number and $k \leq D$, each disk shares an equal amount of the rebuilding workload with all disks that are from different shelves.

(iv) When D is a prime number and $k \leq D$, in each rotation, any two disks from different shelves appear in the same stripe once and only once.

(v) The declustering ratio is $\frac{1}{D}$.

Proof.

(i) According to the Simple PCDSDF algorithm, there are D iterations and each iteration builds D stripes. Therefore, the number of stripes in each rotation is $D*D$. Since each stripe contains k stripe units and there are $D*k$ disks in total, each disk contains $\frac{D*D*k}{(D*k)} = D$ stripe units in each rotation.

(ii) Assume that the number of stripes per rotation is S, then the number of stripe units per rotation on each disk is $\frac{S*k}{n}$. When a disk fails, it needs to read $k-1$ stripe units to reconstruct each stripe unit on the failed disk. Hence, the total number of stripe units to be read per rotation for the disk reconstruction is $\frac{S*k*(k-1)}{n}$. This reconstruction load should be evenly distributed over the $(L-1)*D$ surviving disks that do not reside in the same shelf as the failed disk. Thus, each of these surviving disks needs to read $\frac{S*k*(k-1)}{n*(L-1)*D}$ stripe units. According to the assumption of the simple PCDSDF scheme, the number of shelves

L is equal to the stripe width k. Also, n is equal to $L*D$, which is $k*D$. Therefore, the number of stripe units that each surviving disk needs to read is $\frac{S}{D*D}$. Since the minimum number of stripe units that each surviving disk needs to read is 1, the minimum value of S is $D*D$. This proves that the number of stripes in each rotation of simple PCDSDF is the minimum number of stripes required to achieve a balanced rebuilding workload.

(iii) This proof is outlined in Appendix A.

(iv) According to property (iii), any two disks from different shelves share the rebuilding workload, namely they appear in the same stripe. According to property (ii), to rebuild a failed disk, 1 stripe unit must be read from each surviving disk for each rotation. Thus, in each rotation, any two disks from different shelves appear in the same stripe once and only once.

(v) Because each disk shares an equal amount of the rebuilding workload with any disk that is from a different loop, each disk shares its rebuilding workload with $(k-1)*D$ disks. Therefore, the parity declustering ratio is $\frac{(k-1)}{(k-1)*D} = 1/D$.

□

6.1.2 Complete PCDSDF

Simple PCDSDF deals with the situation when D is a prime number, $L = k$ (the number of shelves is equal to the stripe width), and $k \leq D$. However, real systems are more complicated. This sub-section considers a data layout which relaxes the constraints of Simple PCDSDF. The following three cases are considered.

- $L > k$

  When L is larger than k, namely the number of shelves is larger than the number of disks required for each stripe, a two-step block design can be performed. The first step is to perform a shelf-level block design, which selects k shelves from L shelves such that each shelf appears the same number of times and each shelf shares the same rebuilding workload with the other shelves. This can be done by using the BCBD, BIBD or PRIME/RELPR algorithm. The second step is to perform a disk-level block design for the disks in each shelf combination by using simple PCDSDF. Figure 6.2 shows an example of this two-step block design. In this example, there are four shelves, with each shelf containing three

disks, and the stripe width k equal to 3. The first step of the block design is to conduct the shelf-level block design. BCBD is used to select three out of four shelves in this case. Step 1 of Figure 6.2 (b) lists all these shelf combinations. The second step of the block design is to conduct a simple PCDSDF design for disks in each shelf combination, as shown in Figure 6.2 (b) step 2. Figure 6.2 (c) shows the final data layout on the disks.

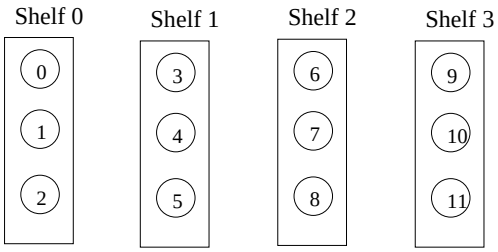
- D **is not a prime number.**

  The property of sharing the rebuilding workload with all disks from different shelves only stands for prime number D. If D is not a prime number, each disk only shares the rebuilding workload with certain disks, resulting in bottlenecks during disk reconstruction. Figure 6.3 (a) gives such an example. In this example, there are four shelves and each shelf contains four disks. It can be seen that using a simple index permutation for the block design will result in disk 0 and disk 2 in shelf 0 only sharing their rebuilding workload with disk 8 and disk 10 in shelf 2, not with disk 9 and disk 11. Similarly, disk 1 and disk 3 in shelf 0 only share their rebuilding workload with disk 9 and disk 11 in shelf 2, not with disk 8 and disk 10. To avoid this, the number of shelves connected by index permutation must be not larger than the smallest prime factor of D, denoted as PD. If k is larger than PD, k shelves needs to be divided into multiple sub-groups, with each sub-group containing PD or less than PD shelves, as shown in Figure 6.3 (b). Disks in each sub-group are first combined together using index permutation. Following that, the sub-groups are combined together. If there are more than PD sub-groups, those sub-groups must also be divided into sub-groups again so that performed on PD or less than PD shelves.

- $k > D$

  When k is larger than D, simply applying the index permutation technique over k shelves will result in a scenario where some disks always appear in the same parity groups, resulting in an uneven reconstruction load distribution. To deal with this situation, the k shelves are evenly divided into a number of sub-groups such that each sub-group contains D or less than D shelves, as shown in Figure 6.4. This is implemented by first connecting the disks within each sub-group and then connecting those sub-groups together using the index permutation technique. To avoid a rebuilding bottleneck, it is essential that the number of sets to

(a) System architecture



(b) Two−step block design

step 1: Combine shelves using BCBD or BIBD

Shelf comb 0	Shelf 0	Shelf 1	Shelf 2
Shelf comb 1	Shelf 0	Shelf 1	Shelf 3
Shelf comb 2	Shelf 0	Shelf 2	Shelf 3
Shelf comb 3	Shelf 1	Shelf 2	Shelf 3

step 2: For each shelf combination arrange data using simple PCDSDF

Shelf comb 0			Shelf comb 1			Shelf comb 2			Shelf comb 3		
0	3	6	0	3	9	0	6	9	3	6	9
1	4	7	1	4	10	1	7	10	4	7	10
2	5	8	2	5	11	2	8	11	5	8	11
0	4	8	0	4	11	0	7	11	3	7	11
1	5	6	1	5	9	1	8	9	4	8	9
2	3	7	2	3	10	2	6	10	5	6	10
0	5	7	0	5	10	0	8	10	3	8	10
1	3	8	1	3	11	1	6	11	4	6	11
2	4	6	2	4	9	2	7	9	5	7	9

(c) Final Data Layout

Shelf	Shelf 0			Shelf 1			Shelf 2			Shelf 3		
disk	0	1	2	3	4	5	6	7	8	9	10	11
	S0	S1	S2	S0	S1	S2	S0	S1	S2	S9	S10	S11
	S3	S4	S5	S5	S3	S4	S4	S5	S3	S13	S14	S12
	S6	S7	S8	S7	S8	S6	S8	S6	S7	S17	S15	S16
	S9	S10	S11	S9	S10	S11	S18	S19	S20	S18	S19	S20
	S12	S13	S14	S14	S12	S13	S23	S21	S22	S22	S23	S21
	S15	S16	S17	S16	S17	S15	S25	S27	S24	S27	S24	S25
	S18	S19	S20	S27	S28	S29	S27	S28	S29	S27	S28	S29
	S21	S22	S23	S30	S31	S32	S32	S30	S31	S31	S32	S30
	S24	S25	Σ26	S33	S34	S35	S33	S35	S34	S35	S34	S33

Figure 6.2: **PCDSDF data layout for $L > k$. L=4, k=3, D=3**

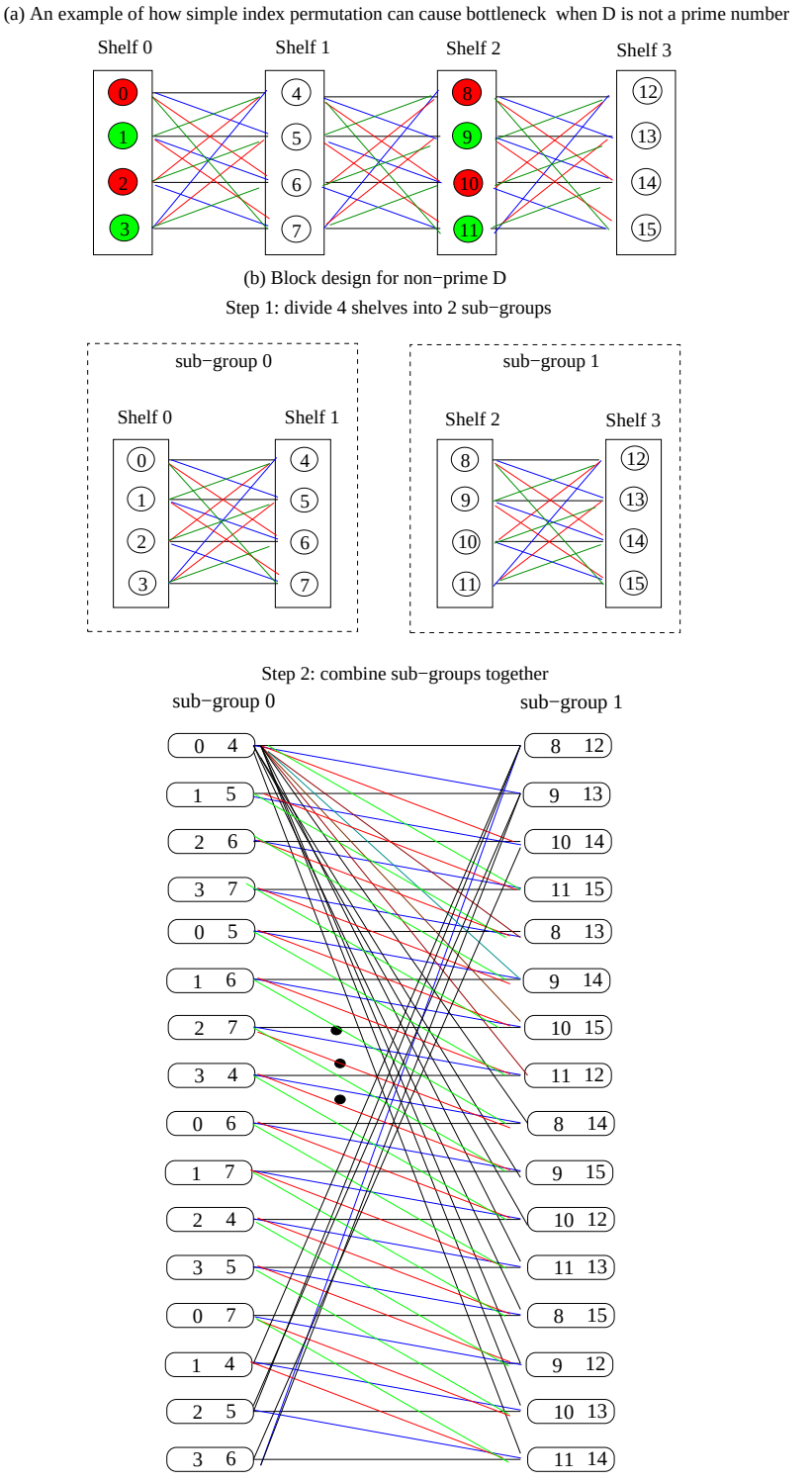


Figure 6.3: **PCDSDF data layout for non-prime D:** $L = k$=4, D=4

be connected is not larger than D. Therefore, if in the second step the number of sub-groups is still larger than D, those sub-groups are further aggregated into sub-groups. This process is repeated until the number of sub-groups to be connected is equal to or less than D.

Combining the mechanisms discussed above leads to the proposed complete PCDSDF algorithm (Algorithm 4) which works for any disk array configuration. The first step of the complete PCDSDF is to organise the overall L shelves into a number of k-element shelf sub-sets using BCBD, BIBD or PRIME/RELPR[3]. Then, for each shelf combination, the disk-level block design algorithm is invoked.

Assume that PD is the smallest prime factor of D ($PD = D$ if D is a prime number). If k is larger than PD, then k shelves are first divided into S subsets with each containing PD or less than PD shelves. For each of these subsets, the index permutation technique (namely the simple PCDSDF algorithm) is called to combine disks together for each stripe. If S is larger than 1, this step is repeated until one sub-set remains.

Table 6.2 lists the number of stripes per rotation generated by the complete PCDSDF algorithm for a conventional stripe width 8, and compares it to a complete block design (requiring $\binom{L}{k} \times D^k$ stripes). The number of shelves is also equal to 8. It can be seen that using the PCDSDF algorithm, the number of stripes per rotation is significantly reduced.

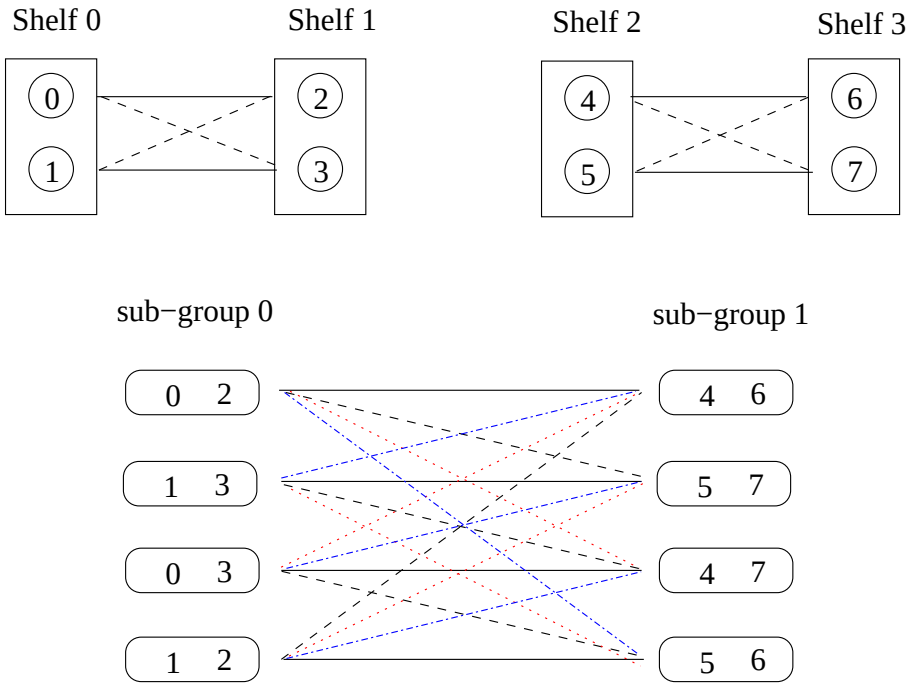
D	PCDSDF	$\binom{L}{k} \times D^k$
1	8	8
2	256	256
3	6561	6561
5	625	390625
7	2401	5764801
11	121	214358881

Table 6.2: Number of Stripe Per Rotation

6.1.3 Reliability Analysis of PCDSDF

This subsection analyses system reliability by considering disk failures and shelf failures. Assuming that the failure rates of disks and shelves are exponentially distributed

[3]Depending on L and k, to reduce the number of combinations



Block Design

0	2	4	6
1	3	5	7
0	3	4	7
1	2	5	6

0	2	4	7
1	3	5	6
0	3	4	6
1	2	5	7

0	2	5	7
1	3	4	7
0	3	5	6
1	2	4	6

0	2	5	6
1	3	4	6
0	3	5	7
1	2	4	7

Data Layout on Disks

	Shelf 0		Shelf 1		Shelf 2		Shelf 3	
Disk	0	1	2	3	4	5	6	7
	S0	S1	S0	S1	S0	S1	S0	S1
	S2	S3	S3	S2	S2	S3	S3	S2
	S4	S5	S4	S5	S5	S4	S6	S4
	S6	S7	S7	S6	S7	S6	S7	S5
	S8	S9	S8	S9	S8	S9	S9	S8
	S10	S11	S11	S10	S10	S11	S10	S11
	S12	S13	S12	S13	S13	S12	S12	S14
	S14	S15	S15	S14	S15	S14	S13	S15

Figure 6.4: **PCDSDF data layout when $k > D$. k = 4, D=2**

Algorithm 4 Complete PCDSDF

```
1: Organise L shelves into k-element shelf sub-sets (also called groups).
2: PD ⇐ the smallest prime divisor of D.
3: for each k-element shelf sub-set do
4:    G ⇐ k
5:    while G > 1 do
6:       Evenly divide G groups into S sub-sets such that each sub-set contains PD or
         less than PD groups.
7:       for sgIndex = 0 to S − 1 do
8:          K1 ⇐ number of groups in the sub-set sgIndex.
9:          Simple_PCDSDF(itemsPerGroup, K1, index_vector).
10:         for i = 0 to index_vector.size() − 1 do
11:            for j = 0 to K1 − 1 do
12:               connect_index ⇐ index_vector[i][j];
13:               for l = 0 to itemsPerGroup − 1 do
14:                  disk ⇐ disks_of_groups[j + sgIndex ∗ K1][connect_index][l];
15:                  disks_on_sub_group[sgIndex][i].push(disk);
16:               end for
17:            end for
18:         end for
19:      end for
20:      disks_of_groups ⇐ disks_on_sub_group;
21:      G ⇐ S.
22:   end while
23: end for
```

and the failures are independent, the failure rate of a RAID system that deploys only a simple RAID 5 data layout is computed in Equation 6.1. In such a system, two kinds of failure can cause the system to lose data service: having another disk failure from the same protection group after a disk fails and having a shelf failure[4]. Thus, the total system failure rate is the sum of the failure rates of these two failures.

$$\textbf{Failure rate}_{\textbf{RAID5}} = \frac{n}{MTTF_{disk}} * \frac{(k-1)MTTR_{disk_{RAID5}}}{MTTF_{disk}} + \frac{L}{MTTF_{shelf}} \tag{6.1}$$

The MTTF of the system is then the inverse of the system failure rate, as shown in Equation 6.2.

$$\textbf{MTTF}_{\textbf{RAID5}} = \frac{1}{\frac{n}{MTTF_{disk}} * \frac{(k-1)MTTR_{disk_{RAID5}}}{MTTF_{disk}} + \frac{L}{MTTF_{shelf}}} \tag{6.2}$$

If the RAID system deploy the PCDSDF data layout, it can survive both a single disk failure and a single shelf failure. The failures that can lead to a system failure include:

- having another disk failure from other shelves in the same protection group after a disk fails [5];
- having another shelf failure after a shelf fails;
- having a disk failure and then a shelf failure by a shelf that does not contain the failed disk;
- having a shelf failure, and then a disk failure by a disk that is not mounted in the failed shelf.

The total system failure rate is the sum of the failure rates of the above failures, as

[4] the probability of having just one shelf failure is $\frac{\binom{L}{1}}{MTTF_{shelf}} * (1 - \frac{1}{MTTF_{shelf}})^{L-1}$. As $\frac{1}{MTTF_{shelf}}$ is very small, the probability of having just one shelf failure is approximately $\frac{L}{MTTF_{shelf}}$.

[5] Because the PCDSDF data layout can survive a single shelf failure, another disk failure from the same shelf will not lead to system failure.

shown in Equation 6.3.

$$\begin{aligned}\textbf{Failure rate}_{\textbf{PCDSDF}} &= \frac{n}{MTTF_{disk}} * \frac{(n-D)MTTR_{disk_{PCDSDF}}}{MTTF_{disk}} \\ &+ \frac{L}{MTTF_{shelf}} * \frac{(L-1)MTTR_{shelf}}{MTTF_{shelf}} \\ &+ \frac{n}{MTTF_{disk}} * \frac{(L-1)MTTR_{disk_{PCDSDF}}}{MTTF_{shelf}} \\ &+ \frac{L}{MTTF_{shelf}} * \frac{(n-D)MTTR_{shelf}}{MTTF_{disk}}\end{aligned} \tag{6.3}$$

where $MTTR_{disk_{PCDSDF}} = \frac{MTTR_{disk_{RAID5}} * (k-1)}{n-D}$, because PCDSDF is a parity declustering data layout and theoretically the disk rebuilding speed can improve by $\frac{(n-D)}{(k-1)}$ times since all disks except those in the same shelf participate in the disk reconstruction.

The MTTF of the system that deploys PCDSDF data layouts is then the inverse of Equation 6.3. Figure 6.5 compares the MTTF of PCDSDF with that of RAID 5 with respect to the number of shelves. It can be seen that the MTTF of the system is significantly improved by deploying a PCDSDF data layout.

6.2 System Architecture for Fast Disk Reconstruction

As described in Chapter 1, although there has been a lot of research on fast disk reconstruction algorithms, the focus has been on the algorithm itself and not on ways to improve the disk reconstruction speed from the aspect of system architecture design. This section describes a system architecture and rebuilding mechanism for fast disk reconstruction based on parity declustering data layouts.

6.2.1 System Architecture

A large scale RAID system usually consists of multiple loops with each loop containing a number of disk shelf enclosures. Figure 6.6 shows a typical architecture of a modern large-scale RAID system.

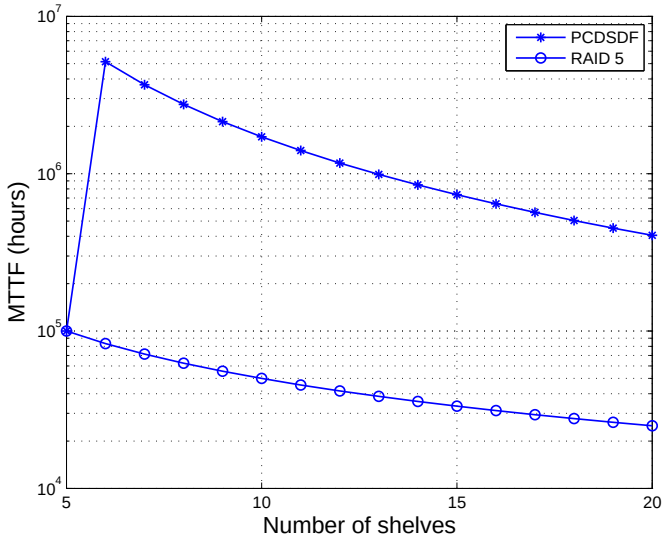


Figure 6.5: **MTTF of PCDSDF VS MTTF of RAID 5 with respect to the number of shelves** In this figure, the number of disks per shelf (D) is set to the standard industry number 14. Assume that the MTTF of the disks and shelves is 50,000 hours. The MTTR of disks under RAID 5 data layouts is 6 hours and the MTTR of a shelf is assumed to be 24 hours. The stripe width is is set to 5.

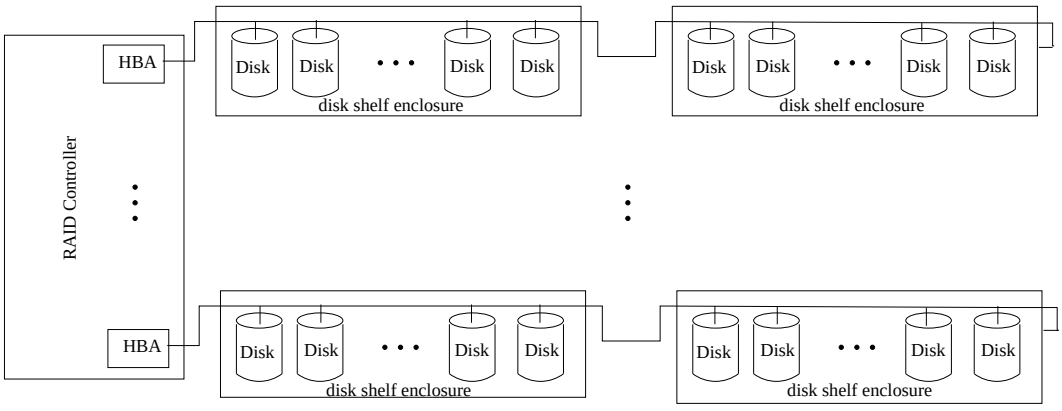


Figure 6.6: **Typical RAID System Architecture**

6.2.2 Minimising the Effect of Limited Loop Bandwidth during Disk Reconstruction

In OLTP work environments, the disk workload in normal operation mode is mainly random, but the disk rebuilding workload is sequential. Therefore, more loop bandwidth is required to support the reconstruction workload. If all disks are organised into a single parity declustered layout and they all participate in a reconstruction, then loop bandwidth which is otherwise sufficient in normal mode might become a bottleneck. In order to eliminate the loop bandwidth bottleneck, the disk array is divided into a number of *Parity deClustering (PC) groups*, as shown in Figure 6.7 . Each PC group selects the same number of disks from each loop. The block layout for one PC group is independent from any other PC group. When a disk (or two) fails in a PC group, only disks in that PC group participate in the reconstruction. The size of a PC group (the number of disks in each PC group) is a function of the number of loops, the loop bandwidth and the average user workload during reconstruction. By choosing an appropriate PC group size, it should be possible to minimise the effect of the loop bandwidth bottleneck while simultaneously realizing the benefits of parity declustering.

Algorithm 5 describes how to build P PC groups in a disk array such that each PC group (containing N_{PC} disks) is distributed across all loops to achieve the maximum loop bandwidth during disk reconstruction.

6.2.3 Data Structure of A PC Group

As described in Section 2.1.1.1, the basic storage unit in a typical RAID system is a stripe unit. The minimum collection of stripe units (including both data units and parity units) over which the parity unit is computed is called a *stripe*. The number of disks in a stripe is called the *stripe width*. Most prior work on parity declustering has used a stripe as the basic unit to be distributed over disks. However, this approach suffers from the disadvantage that a disk seek might be required for each stripe unit during the rebuild process. As a result, reconstruction might not be able to exploit the sequential disk bandwidth to its maximum. To address this issue, a *stripe group* is defined in this dissertation as the basic distribution unit. A stripe group is a number of contiguous stripes. Contiguous stripe units on each disk within a particular stripe group are called *chunks*. Stripe groups are distributed across all disks in a PC group. The minimum number of stripe groups that form an evenly balanced parity declustering data layout

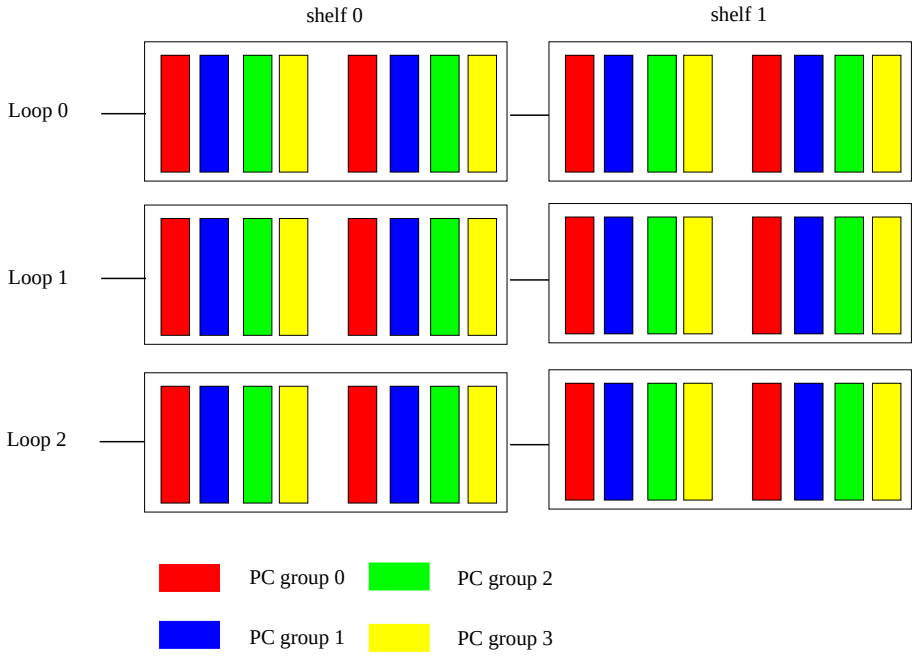


Figure 6.7: **Division of a Disk Array into PC groups** In this example, there are three loops and the disks in these loops are divided into four PC groups, with each PC group selecting the same number disks from each loop.

Algorithm 5 Build Parity Declustering Groups cross All Loops

for Each loop j **do**
  $D[j] \leftarrow 0$ $\{D[j]$ is the smallest disk number that has not been assigned to any PC group$\}$
end for
$loop \leftarrow 0$
for $i = 0$ to $P-1$ **do**
  $k \leftarrow 0$
  while $k \leq N_{PC}$ **do**
    $PCG[i][k] \leftarrow D[loop]$
    $k \leftarrow k+1$
    $D[loop] \leftarrow D[loop]+1$
    $loop \leftarrow (loop+1)\%L$
  end while
end for

is called a *rotation*. Figure 6.8 shows the structure of a stripe group and a chunk while Figure 6.9 shows the data layout of a PC group which contains a number of rotations where each rotation consists of a set of stripe groups. In this example, stripe groups with stripe width equal to four are distributed over seven disks, with each stripe group containing six stripes. Once the stripe groups achieve a balanced rebuilding workload, namely a rotation, the same data layout is repeated to form another rotation.

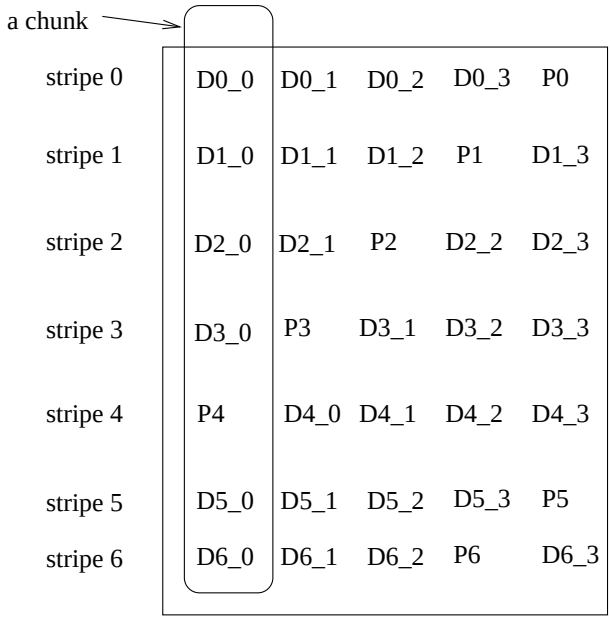


Figure 6.8: **Structure of a stripe group and a chunk** This stripe group consists of 6 stripes. The placement of parity units is right asymmetric. This placement can be changed to any other type of placement according to the requirement.

6.2.4 Disk Reconstruction Algorithm Design

According to Holland, a good disk reconstruction algorithm should have the following properties [42]:

- The algorithm should absorb as much as possible of the array's bandwidth that is not absorbed by the user I/Os.
- The algorithm should preserve the sequentiality of the reconstruction process.
- The algorithm should work on relatively small sets of parity stripes at any one time. (The stripes under reconstruction have to be locked to prevent simultane-

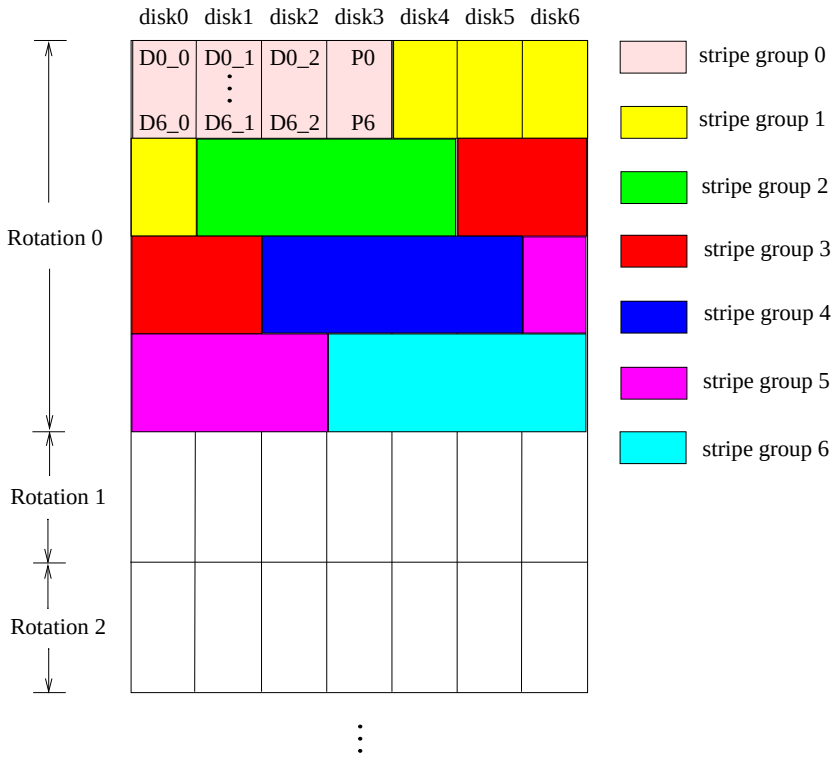


Figure 6.9: **Data structure of a PC group** k=4, n=7. Each stripe group contains six stripes. In this example the parity units for each stripe group are placed on one disk. According to the requirement, they can rotate within each stripe group using the parity placement schemes described in Section 2.1.3.

ous overlapping user writes. Working on a small sets of parity stripes reduces the frequency of user access conflicts with reconstruction operations.)

As described in Section 2.1.4, there are two basic disk reconstruction algorithms: a stripe-oriented algorithm and a disk-oriented algorithm. The stripe-oriented disk reconstruction algorithm involves one (or multiple) process(es) that sequentially rebuild the units on the failed disks. The rebuilding process first identifies the stripe units belonging to the stripe being rebuilt and then sends *read* commands to the surviving disks. After reads to all surviving stripe units have completed, this process XORs the units and then dispatches the write command to the replacement disks. The drawback of the stripe-oriented reconstruction algorithm is that it cannot fully utilise the disk bandwidth that is not used by user requests. In contrast to the stripe-oriented algorithm, the disk-oriented reconstruction algorithm (proposed by Holland and Gibson [42]), instead of having a number of parallel reconstruction processes associated with stripes, creates *n* (the number of disks in a disk array) processes. Each of these processes is associated with a disk. The process associated with each surviving disk reads data units on that disk in order and then submits them to the RAID controller. Once all the data belonging to a particular stripe is ready, the RAID controller executes the XOR operation. The process associated with the replacement disk continually issues write commands to send the reconstructed data to the replacement disk. With this disk-oriented approach, the recovery process can fully utilise the array bandwidth, leading to a significant reduction in disk rebuilding time, with only a small increase in the user response time.

Because of its advantages, a disk-oriented reconstruction algorithm was chosen for further development, modified to suit the system architecture. The reconstruction algorithm includes two types of process. The first type manages reading the necessary stripe units from the surviving disks. The second type manages writing reconstructed stripe units to the disks (to a replacement disk or distributed hot spares). The reading process works as follows:

It is worth noting that for a parity declustered data layout, not all units on each disk are needed for reconstruction. The stripe units to be read depend on which disk failed. Before starting the reconstruction process, it is necessary to build a table that records which chunks/stripe units are needed. A centralized buffer is used to store the partial XOR results for the stripe units read from surviving disks (*i.e.* before the unit to be written has completed). This centralized buffer is called a *free buffer*. Besides these free buffers, each surviving disk has its own private buffers to store units that

Algorithm 6 Disk-oriented Reconstruction Reading Process

1: For each surviving disk
2: **repeat**
3: Find the next lowest-numbered unit on this disk that is needed for reconstruction.
4: Issue a low-priority request to read the indicated unit into a buffer.
5: Wait for the read to complete.
6: Submit the unit's data to a centralized buffer manager for XOR, or Block the process if the buffer manager has no memory to accept the unit
7: **until** all necessary units have been read

have been read but not yet submitted to the free buffer. In previous research work, the reconstruction algorithm submits a stripe unit to the free buffer as long as there are free buffer units available. However, in the system defined in Section 6.2.3, each chunk consists of multiple stripe units. Therefore, using the same strategy to manage the free buffer might result in a deadlock for two reasons:

- One disk may be faster than the others (due to imbalanced workload or different disk characteristic) and these disks will keep reading stripe units for different stripes. These partially-completed stripes might end up consuming all the free buffers. On the one hand, the reconstruction process will end up waiting for the remaining units for these stripes, while on the other, disks might be blocked because of unavailability of free buffers.

- Because each stripe group contains more than one stripe, if free buffer capacity is small, the stripes of a certain stripe group may use all the free buffers. While these stripes are waiting for the rest of their stripe units, other disks might be blocked due to the unavailability of free buffers. An example is shown in Figure 6.10. In this example, there are five disks and stripe groups with stripe width equal to three are distributed over these five disks (each colour represents a stripe group). The free buffer contains only one buffer unit and each disk has one private buffer unit. Assume that disk *c* has failed. Each disk reads the lowest-numbered unit that is needed for reconstruction. After the units are read, they will be submitted to the free buffer. Disk *a* and disk *b* submit stripe units for stripe 0 which are XORed. The write unit is then sent to the replacement disk. As there are no more free buffer units available, disk *d* is blocked from submit-

ting unit $d3$. After the reconstruction of stripe 0 is completed, the free buffer unit is released and disk d submits unit $d3$ to it. Disk a and b keep reading the next unit $a1$ and $b1$. As the free buffer is full, disk a and disk b are now blocked from submitting. Because unit $d3$ belongs to stripe 6 and the other unit of stripe 6 is $b3$, the free buffer is waiting for the submission of unit $b3$. However, as disk b is blocked, it cannot read $b3$ from the disk, which leads to a deadlock.

The deadlock problem is solved by restricting the number of free buffers that each disk can use. Since each stripe only needs one buffer unit (or two buffer units if RAID 6 is used), the disk that submits the first stripe unit for a stripe is counted as the disk that is using that free buffer unit. To prevent deadlock, the number of free buffer units that each surviving disk can use is restricted to $\frac{FREE\_BUFFER}{NUM\_OF\_SURVIVING\_DISKS}$. The number of free buffer units must be larger than the number of surviving disks. If the number of free buffers used by a disk reaches this threshold, this disk will be blocked until one of the free buffers that it is using gets released. The stripe unit submission algorithm works as follows:

Algorithm 7 Submission of ready units to free buffers for disk i

Require: FB[i]=0;

$MAX\_FB = FREE\_BUFFER/NUM\_OF\_SURVIVING\_DISKS$;

1: **if** $\exists$ stripe within the free buffer that the unit read from the disk belongs to **then**
2: Submit the unit to that stripe for XOR operation
3: **else if** $FB[i] \leq MAX\_FB$ **then**
4: Allocate a stripe to unit and queue reads for all other units for that stripe.
5: FB[i]++;
6: **else**
7: Block this reading process
8: **end if**

Once the write units of a stripe have been sent to the replacement disk and the controller receives the response, the free buffer units will be released. The disk using those free buffer units will be able to unblock the reading process.

Holland's dissertation concluded that, in general, using about **three times** as many reconstruction free buffers as there are disks is sufficient to achieve the full benefits of a disk-oriented reconstruction algorithm. With the stripe group based data layout, this means that reconstruction needs $3 * n * size_{chunk}$, where n is the number of surviving

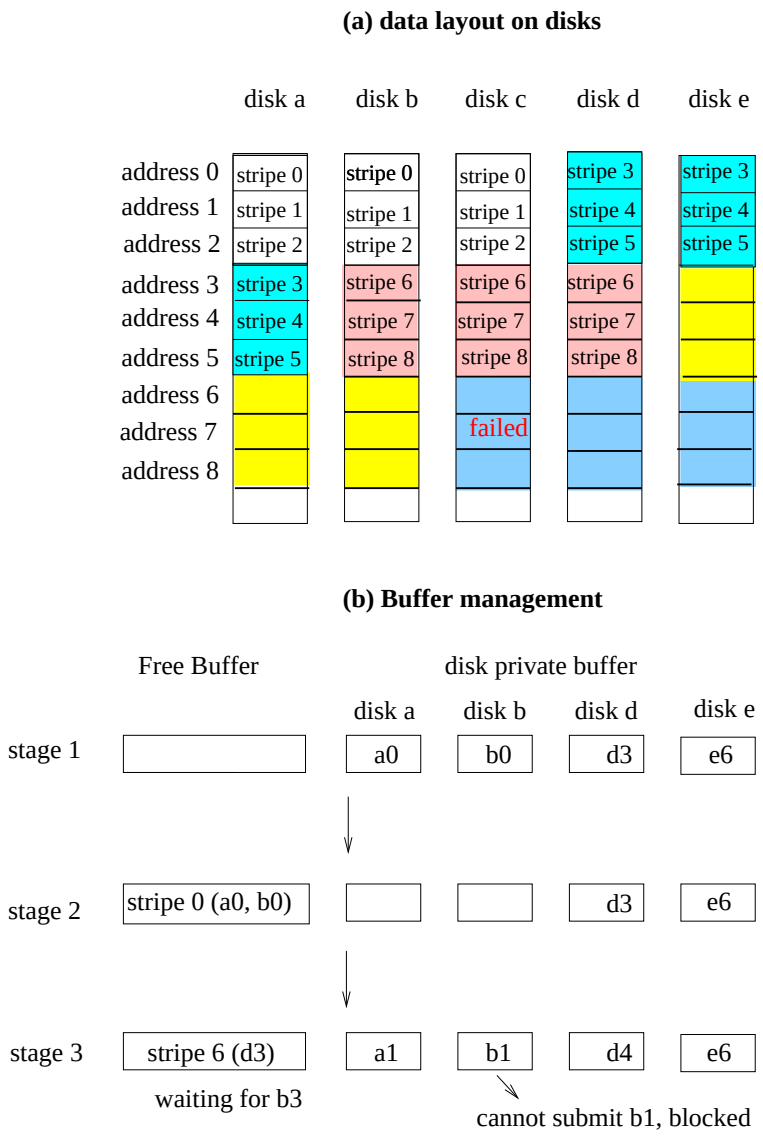


Figure 6.10: **Example of rebuilding process deadlock** (a) shows the data layout on the disks; and (b) shows the contents of free buffer and disk private buffer at three stages. Assuming that disk c fails, at stage 1, each disk read the first stripe units it needs to read to its own private buffer. At stage 2, disk a and disk b submit the unit in their private buffer to the free buffer, but disk d and disk e are blocked from submitting because the free buffer is full now. At stage 3, disk d submits the unit in its private to the free buffer which belongs to stripe 6, and disk a and disk b read the next necessary unit into their private buffer. As stripe 6 consists of unit $b3$, $c3$ and $d3$, $c3$ cannot be rebuilt unless unit $b3$ is also submitted. However, at this stage disk b has just read unit $b1$ from disk into its own private buffer and is now blocked because of the full free buffer. Therefore, the rebuilding process enters deadlock.

disks. When the chunk size is large, a lot of memory can be consumed by the reconstruction process. The effect of the free buffer size is investigated in the section 6.5.2.4.

6.3 Distributed Hot Sparing

This section presents an efficient distributed hot spare allocation and assignment algorithm for a general parity declustering data layout. Distributed hot sparing was proposed to improve performance, reduce disk reconstruction time and prevent the writing disk from becoming a bottleneck during reconstruction [68]. In a data layout that deploys distributed hot sparing, the following two properties are desired.

1. Distributed sparing: all disks contain the same number of spare stripe units.

2. Efficient sparing: after reconstruction of the failed disk(s) there will be no spare space left.

Deriving a distributed-sparing layout for a declustered organisation involves two steps. The first step is to statically reserve disk space as spare space. The term "statically" means that the disk space reserved does not depend on which disk has failed. To simplify the mapping and data management, a contiguous storage area with fixed disk offset is chosen as the spare space. Assume that the system can tolerate failure of f disks. Assuming that each disk contains r chunks in each rotation, if f disks fail, $r * f$ chunks are lost from each rotation, where r is the number of chunks of a rotation on each disk. These rf chunks should be evenly distributed across the surviving $n-f$ disks. If rf is not an integral multiple of $n-f$, to avoid storage space wastage, a number of data rotations are allocated in contiguous storage space and the corresponding spare space is allocated as a contiguous area right after the data area. The number of data rotations combined is $N_f = \frac{LCM(rf, n-f)}{rf}$ and the number of rows of free chunks is $N_s = \frac{LCM(rf, n-f)}{n-f}$. Figure 6.11 shows a data layout where N_f data rotations are combined and N_s rows of spare chunks are associated with them.

The second step is to assign the reserved spare chunks to the lost chunks on the failed disks as replacements. This step is dynamic and depends on which disk has failed. Logically, this assignment occurs only after a disk has failed. Holland claimed that in practice the assignment can be pre-computed for every possible disk failure to ensure that the assignment will not delay the reconstruction. However, if the system deploys a RAID 6 protection mechanism, there are $n * (n-1)$ possible assignments.

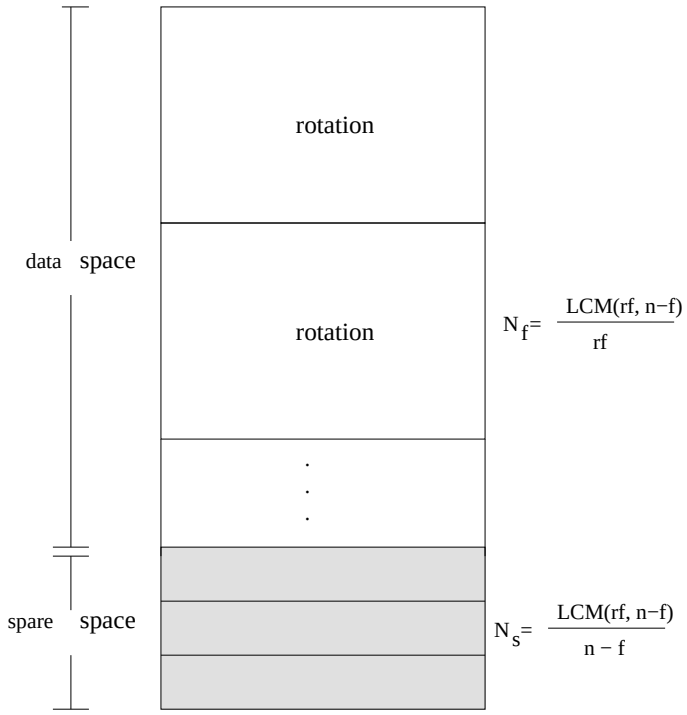


Figure 6.11: **Distributed Hot Spare Structure** r is the number of chunks of a rotation. N_f is the number of data rotations to be combined together and N_s is the number of rows of chunks for the spare unit space.

Thus, it is more practical to compute the assignment after a disk failure. In this dissertation an efficient spare units assignment algorithm is presented which works for any evenly balanced declustered data layout, as shown in Algorithm 8. This routine is invoked to build the map between spare chunks and lost chunks before the reconstruction process starts. Unlike Holland's assignment algorithm, which restricts f to 1, this algorithm can also work on a data layout that can survive 2 disk failures. When there is more than one disk failure, it will assign the spare chunks to each failed disk in order. During the assignment for the first failed disk, it will assume that all the other disks work well, including the failed disk that has not yet been considered (namely the failed disk that has not got its spare chunks). After the assignment for the first failed disk, the algorithm will assign spare chunks for chunks on the second failed disk, including those in the spare region.

The first step in assigning spare chunks for a failed disk is to collect all the stripe groups affected by the failure, as in Line 2 of the Algorithm. Following this, all the *complement disks* (namely the disks that do not participate in a stripe group) are listed for each affected stripe group, as shown in Line 3. To survive another disk failure after

reconstruction, the spare chunk for a particular failed chunk must reside on one of the complement disks of the stripe group to which it belongs. If a certain disk is included in the complement disk set of a failed chunk, this chunk is called *being related to* that disk. The following steps assign spare chunks for each failed chunk. The assignment process is based on two heuristics: (1) first assign spare chunks from the disks that have minimum failed chunks *related* to it and (2) among the chunks that are *related* to a particular disk, first assign spare chunks for those that are *related* to the minimum number of disks. Line 8 to Line 11 of Algorithm 6 implement this process. First, it lists the chunks *related* to each surviving disk. Second, it selects the disk that has the minimum number of *related* chunks, denoted as d_{min}, and sorts those chunks in ascending order by the number of complement disks to which they are *related*. Third, to ensure that replacement chunks are evenly distributed over all surviving disks, the number of spare chunks to be assigned from d_{min} is restricted to be the minimum (U/N, spare chunks on d_{min}), where U is the unassigned failed chunks and N is the number of remaining disks from which spare chunks can be assigned. The number of spare chunks to be assigned from d_{min} is denoted as Max. Fourth, the algorithm assigns Max spare chunks to its first Max related chunks. Lastly, disk d_{min} is removed from the surviving disk set and the complement disk set of all failed chunks. Moreover, chunks that have been assigned a spare chunk are removed from the related chunk list of each surviving disk. This assignment process is repeated until all failed chunks have been assigned.

Figure 6.12 gives a simple example of the distributed hot spare assignment process for a parity declustering data layout. Figure 6.12 (a) lists the original block design of the data layout and Figure 6.12 (b) depicts the data layout. As mentioned before, to avoid wasting storage space, two data rotations are combined together to share three rows of contiguous spare space. Figure 6.12 (c) lists the affected stripe groups and their corresponding block design (assuming that disk d2 has failed). To distinguish stripe groups of different data rotations, a chunk is represented by “rotation number.stripe group number”. Figure 6.12 (d) shows the complement disks of the affected stripe groups and the chunk lists that are related to each surviving disk. As each disk has the same number of related chunks, disk d0 is simply selected as the first disk from which spare chunks will be assigned. Based on the rule in Algorithm 8, spare chunks from disk d0 are assigned to chunks 0.6, 0.7 and 0.9[6] respectively. Figure 6.12 (e) shows the updated complement disks and the chunks related to each surviving disk

[6]Since each stripe group only loses one chunk, stripe group number is used to represent that chunk

Algorithm 8 Assign spare chunks to rebuilding chunks

1: **for** each failed disk fd_i **do**
2: Group all affected stripe groups (chunks that reside on disk fd_i) into a set $\mathbb{C}$. $\mathbb{C} = \{C_0, C_1, C_i..., C_{r-1} | \exists d_m \in C_i\}$; these chunks include both chunks that originally reside on fd_i and those allocated to it as hot spare chunks;
3: Group the complement disks of the affected groups in to a set B. $\mathbb{B} = \{B | \exists C \in \mathbb{C}, B = \Psi - C\}$;
4: List the chunks that are "related" to each surviving disk;
5: N=number of surviving disks;
6: U=total number of affected chunks;
7: **while** $N > 0$ **do**
8: Select the disk that has the minimum number of chunks related, denoted as d_{min};
9: Sort the chunks related to d_{min} by the number of complement disks to which they are related;
10: Calculate the number of chunks to be allocated on disk d_{min} as Max=minimum(U/N, spare chunks on d_{min});
11: Assign spare chunks to the first Max sorted chunks related to d_{min} from disk d_{min};
12: Remove disk d_{min} from complement disk set of all chunks.
13: Remove chunks that has been assigned a spare chunk from the related chunk list of each surviving disk;
14: $U \Leftarrow U - Max$;
15: $N \Leftarrow N - -$;
16: **end while**
17: **end for**

after the assignment from disk d0. It can be seen that disk d0 and chunks that have been assigned the spare chunks have been removed from this figure. As each disk still has the same number of chunks related, disk d1 is selected for spare assignment. Among the chunks related to it, chunk 1.9 is related to only one complement disk, so chunk 1.9 is first assigned a spare chunk from disk d1. As the other chunks related to disk d1 are related to the same number of disks, the spare chunks left on disk d1 are assigned to chunks 0.3 and 0.4. To save space, the rest of the assignment process is not drawn in Figure 6.12. Figure 6.12 (f) shows the final data layout that includes both data rotations and spare space filled with lost chunks.

6.4 Experimental Set Up

SIMRAID was used to evaluate the performance of the three approaches proposed in this chapter. This section describes how experiments are set up. The experiments described in this chapter include the three parts:

- to evaluate the performance of PCDSDF;
- to explore the design space of the system architecture for fast disk reconstruction;
- to evaluate the effect of the distributed hot spare allocation and assignment algorithm.

As the size of a PC group (the number of disks in each PC group) is affected by multiple specific factors such as of the number of loops, the loop bandwidth and the user workload during reconstruction, it is trivial to investigate it through simulation. Therefore, this dissertation does not carry out experiment to investigate the size of a PC group or the number of PC groups in a system.

Figure 6.13 shows the system architecture configured for the experiments. For the purpose of simplicity, it contains only one controller and one SBOD switch. As the intent of the experiment is to compare different declustering data layout and adding cache does not change the performance trend, the experiment does not include cache. In so doing, it also saves some simulation time as the system can achieve a larger throughput if cache is used , which implies a longer simulation time. In the experiments carried out to evaluate the performance of PCDSDF, the disks are logically divided into different shelves. All the experiments deploy distributed hot sparing. For PCDSDF, the hot spare is located in the disk slot that is located in the same shelf as the failed disk.

(a) Original block design

stripe group number	block design
0	<0, 1, 2>
1	<0, 1, 3>
2	<0, 1, 4>
3	<0, 2, 3>
4	<0, 2, 4>
5	<0, 3, 4>
6	<1, 2, 3>
7	<1, 2, 4>
8	<1, 3, 4>
9	<2, 3, 4>

(b) Data Layout (N_{ft} = 2 and N_s = 3)

	d0	d1	d2	d3	d4
rotation 0	0	0	0	1	2
	1	1	3	3	4
	2	2	4	5	5
	3	6	6	6	7
	4	7	7	8	8
	5	8	9	9	9
rotation 1	0	0	0	1	2
	1	1	3	3	4
	2	2	4	5	5
	3	6	6	6	7
	4	7	7	8	8
	5	8	9	9	9
spare space					

(c) Affected stripe groups

strip group number	block design
0.0	<0, 1, 2>
0.3	<0, 2, 3>
0.4	<0, 2, 4>
0.6	<1, 2, 3>
0.7	<1, 2, 4>
0.9	<2, 3, 4>
1.0	<0, 1, 2>
1.3	<0, 2, 3>
1.4	<0, 2, 4>
1.6	<1, 2, 3>
1.7	<1, 2, 4>
1.9	<2, 3, 4>

(d) Complement disks and chunks related to each surviving disks

strip group number	Complement disks
0.0	<3, 4>
0.3	<1, 4>
0.4	<1, 3>
0.6	<0, 4>
0.7	<0, 3>
0.9	<0, 1>
1.0	<3, 4>
1.3	<1, 4>
1.4	<1, 3>
1.6	<0, 4>
1.7	<0, 3>
1.9	<0, 1>

d0	0.6	0.7	0.9	1.6	1.7	1.9
d1	0.3	0.4	0.9	1.3	1.4	1.9
d3	0.0	0.4	0.7	1.0	1.4	1.7
d4	0.0	0.3	0.6	1.0	1.3	1.6

(e) Complement disks and chunks related to each surviving disks after spare chunks on d0 been assigned

strip group number	Complement disks
0.0	<3, 4>
0.3	<1, 4>
0.4	<1, 3>
1.0	<3, 4>
1.3	<1, 4>
1.4	<1, 3>
1.6	<4>
1.7	<3>
1.9	<1>

d1	1.9	0.3	0.4	1.3	1.4
d3	1.7	0.0	0.4	1.0	1.4
d4	1.6	0.0	0.3	1.0	1.3

(f) Data layout and assignment of failed chunks to spare chunks

	d0	d1	d2	d3	d4
rotation 0	0	0	0	1	2
	1	1	3	3	4
	2	2	4	5	5
	3	6	6	6	7
	4	7	7	8	8
	5	8	9	9	9
rotation 1	0	0	0	1	2
	1	1	3	3	4
	2	2	4	5	5
	3	6	6	6	7
	4	7	7	8	8
	5	8	9	9	9
spare space	0.6	1.9		1.7	1.6
	0.7	0.3		1.4	1.0
	0.9	0.4		0.0	1.3

Figure 6.12: **Distributed Hot Spare Assignment** n=5, k=3, f=1; Assume that disk d2 is failed. The number in each disk represents the stripe group number. As two rotations are combined here, *rotation number.stripe group number* is used to identify a stripe group. 0.6 means stripe group 6 of rotation 0.

For other parity declustering data layouts, the proposed distributed hot spare allocation and assignment algorithm is used. Table 6.3 lists the setting of the parameters through the experiments.

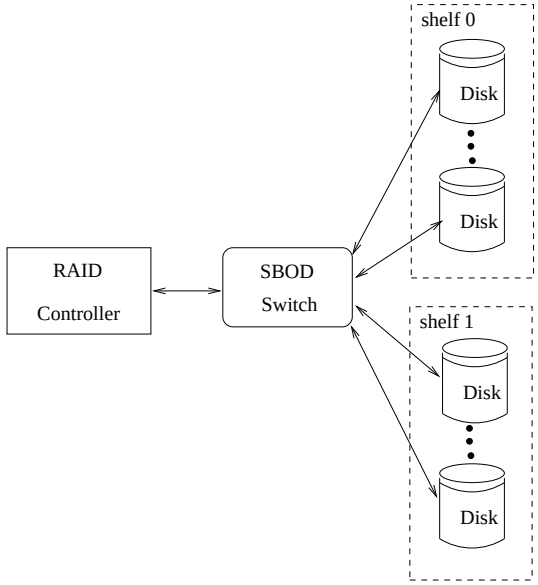


Figure 6.13: **System architecture for the experiments**

Parameter	Value
disk capacity	10 GB
cache policy	no cache
maximum disk queue length	20
RAID level	RAID 5
size of stripe unit	16 KB if not specified in the experiment
stripe units per chunk	16 if not specified in the experiment
number of PC groups	1

Table 6.3: Parameter Settings in the Experiments

6.5 Performance Evaluation Results

This section presents performance evaluation results. Firstly, the degraded and rebuild performance of PCDSDF is presented, with comparison with those of RAID 5 and RELPR. Secondly, the effects of the design parameters of the system architecture on

fast disk recovery are examined. These design parameters include the parity declustering ratio, the chunk size, the private buffer size of the surviving disks and the free buffer size. Lastly, the advantage of using distributed hot sparing is shown.

6.5.1 PCDSDF Performance Evaluation

This subsection presents the performance of the PCDSDF layout, including system reliability, and degraded-mode and rebuild performance.

6.5.1.1 Performance In Degraded Operation Mode

Figure 6.14 shows the system throughput and maximum response time in normal operation mode and degraded operation mode compared to a RAID 5 and RELPR (section 3.3.2) data layout. PCDSDF is able to maintain a good degraded-mode performance for both single disk failure as well as shelf failures. With one disk failure, the degradation of throughput for PCDSDF is less than 6%, which is even better than RELPR. With one shelf failure, the system throughput is still at 83% of that of normal operation mode. Moreover, for PCDSDF there is only a slight increase in the maximum response time in degraded operation mode.

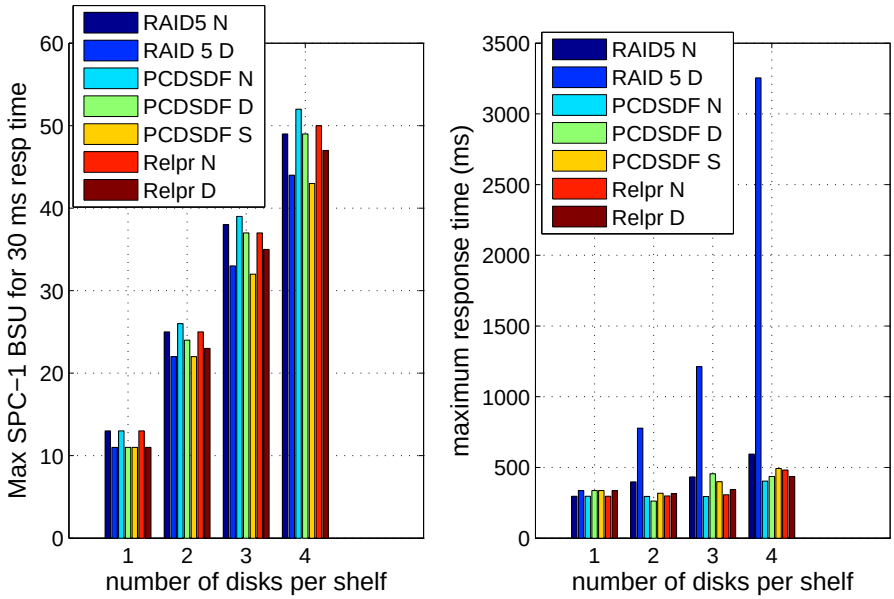


Figure 6.14: **Degraded Performance** $L = k$=8, f=1. 'N' represents normal operation mode. 'D' represents degraded operation mode with one disk failure. 'S' represents degraded operation mode with one shelf failure

6.5.1.2 Disk Rebuilding Performance

Figure 6.15 shows the disk rebuilding speed and the average response time for user I/Os during reconstruction with respect to the number of disks per shelf. It can be seen that as the number of disks per shelf increases, the rebuilding speed for PCDSDF increases almost linearly, which is comparable with RELPR. However, the rebuilding speed for RAID 5 remains the same. Increasing the user I/O workload from 50% of the throughput to 75% decreases the rebuilding speed by about 40%, but increases the average response time of user I/Os by 40%. As the number of disks increases, the disk rebuilding speed increases by 4 times, whereas the average response time of user I/Os increases by only 30%. Using 4 stripe units worth of private disk buffers instead of 2 units increases the rebuilding speed by about 30%, but the increase in the average response time is less than 20%.

6.5.2 Design Space Exploration of Fast Disk Reconstruction Architecture

This sub-section investigates the effect on disk rebuilding speed as a result of varying various system parameters: the parity declustering ratio, the chunk size, the surviving disk private buffer size and the free buffer size. To discount any impact due to user I/Os, the experiments discussed in this sub-section have no user workload during reconstruction.

6.5.2.1 Effect of Parity Declustering Ratio

Figure 6.16 shows the rebuilding speed and loop bandwidth requirement for the PRIME and RELPR data layouts for different declustered group sizes while keeping the stripe width k fixed at 8. As expected, increasing the number of disks in the system results in a linear increase in the rebuilding speed as well as the loop bandwidth required. Compared with PRIME, RELPR gains less reconstruction speedup because its rebuilding workload is not evenly distributed across all disks.

6.5.2.2 Effect of Chunk Size

Figure 6.17 shows the effect of chunk size (in terms of the number of stripe units). The experiment is carried out without any user I/O input. It can be seen that the chunk size should neither be too small nor too large. Due to the nature of the parity declustering

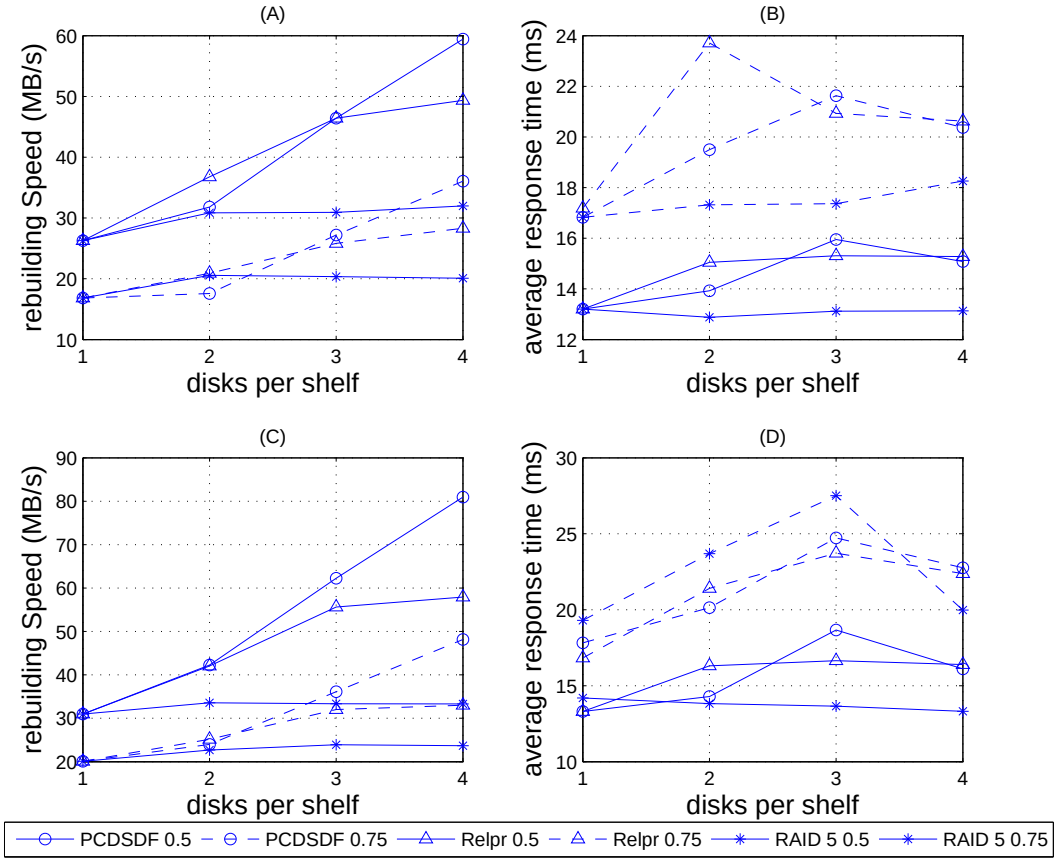


Figure 6.15: **Rebuilding Performance** $L = k$=8, f=1. '0.5' means that the input workload is 50% of the throughput in degraded mode and '0.75' means that the input workload is 75% of the throughput in degraded mode. 'RU' means the number of stripe units of each disk private buffer.

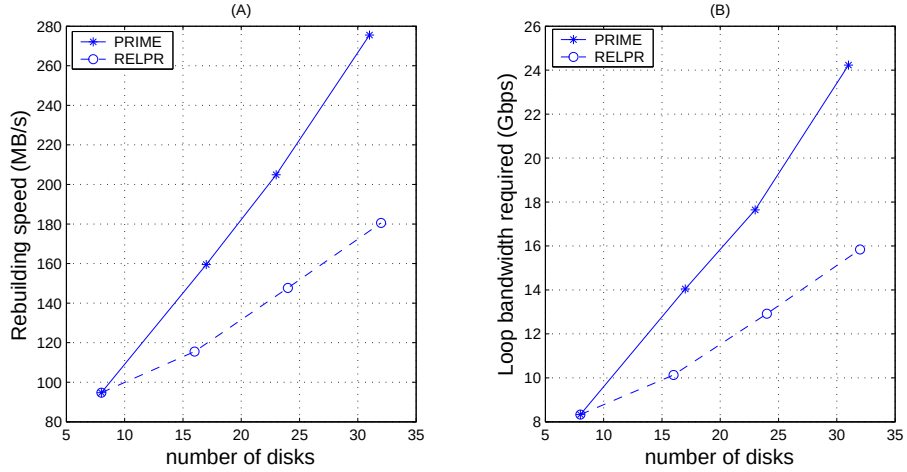


Figure 6.16: **Rebuilding Speed and Loop bandwidth Requirement** $k = 8$

layout, reconstruction I/Os might have to access non-contiguous blocks on disk. If the chunk size is small a lot of time is spent in seeking, which lowers the rebuilding speed, whereas if the chunk is too large, the reconstruction parallelism is reduced. The simulation experiments showed that 16 stripe units per chunk is the optimal value for the 16kB and 32 kB stripe unit size, whereas 8 stripe units per chunk gives the best performance for 64 kB stripe unit size.

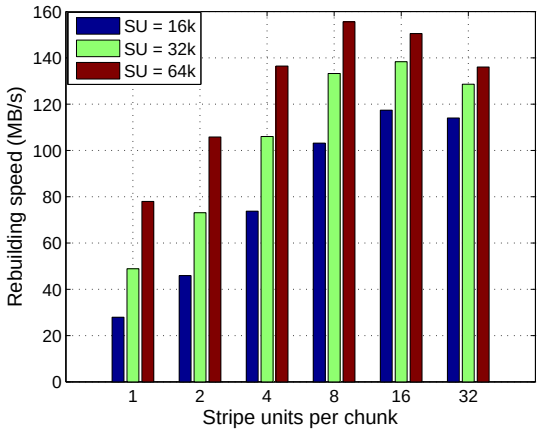


Figure 6.17: **Effect of chunk size** n=17, k=8, $ru = 8$. PRIME and distributed hot spare data layout. 'SU' denotes the size of the stripe unit.

6.5.2.3 Effect of Surviving Disks' Private Buffer Size

Figure 6.18 shows the effect of the private buffer size of the surviving disks (in terms of the number of stripe units) on the disk rebuilding speed. It can be seen that increasing the private buffer size from one stripe unit to two increases the disk rebuilding speed significantly. This is because using two stripe units buffer size results in more efficient sequential reads, whereas a one stripe unit buffer size results in missed revolutions. Further increasing the surviving disk private buffer size only increases the disk rebuilding speed slightly.

6.5.2.4 Effect of free buffer size

Figure 6.19 shows the effect of free buffer size on the rebuilding speed. In this experiment, the rebuilding speed under three configurations is tested: free buffer size equal to $2 * (n-1) * \textit{Chunk Size}$, $(n-1) * \textit{Chunk Size}$ and $\frac{1}{2} * (n-1) * \textit{Chunk Size}$. It can be seen that increasing the free buffer size increases the disk rebuilding speed. As the

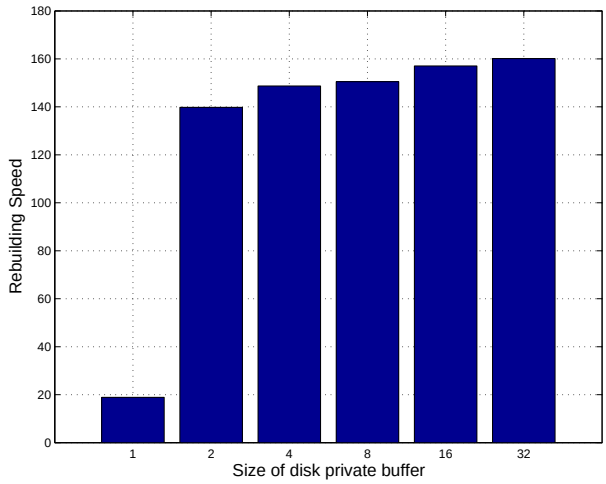


Figure 6.18: **Effect of surviving disk private buffer size (number of stripe units)** PRIMEdata layout. n=17, k=8 and f=1. Free buffer size is $(n-1)*$*chunk size*.

number of disks increases, the incremental reconstruction speedup achieved increases with a larger free buffer size.

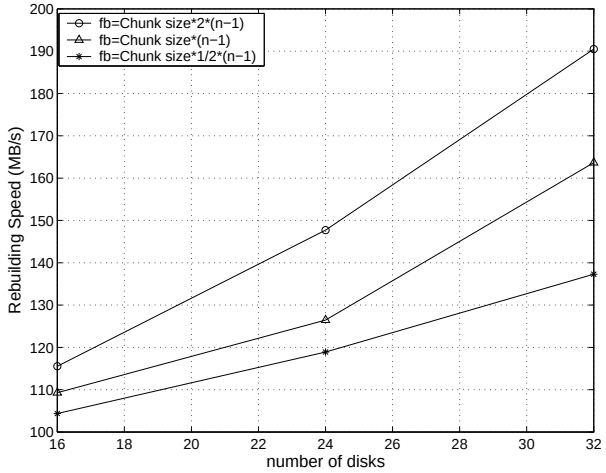


Figure 6.19: **Effect of Free Buffer size** RELPR data layout. k=8 and f=1.

6.5.3 Evaluation of Distributed Hot Sparing

Figure 6.20 compares the rebuilding speed and average response time of user requests under distributed hot sparing using the allocation and assignment mechanisms proposed in this chapter with traditional dedicated hot sparing. It can be seen that as the number of disks increase the rebuilding speed using a distributed data layout (solid

line) increases linearly, whereas the rebuilding speed using a dedicated hot spare (dotted line) remains almost constant. Thus, distributed hot sparing effectively solves the write bottleneck. Furthermore, the average response time of user requests under distributed hot sparing increase by less than 1 ms.

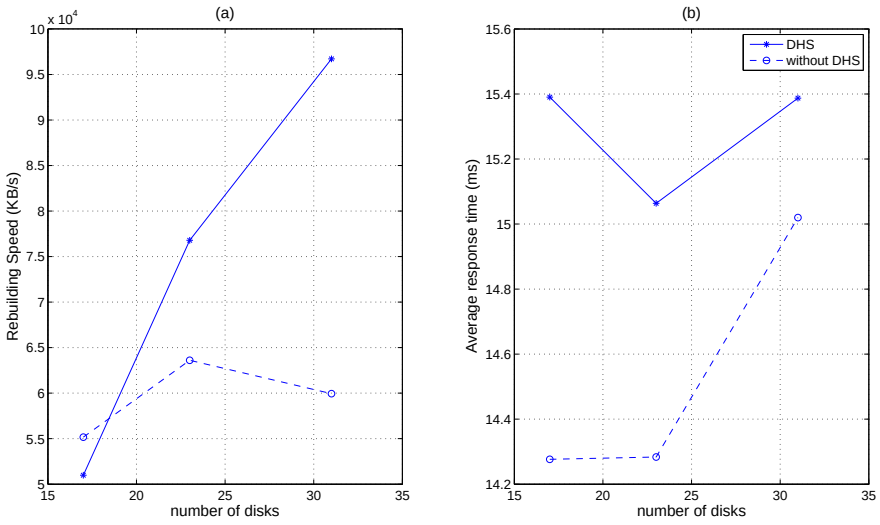


Figure 6.20: **Distributed hot sparing vs. dedicated hot sparing** k=8, ru=4, $f = 1$. Workload is 50% of the maximum workload in degraded mode. The parity declustering data layout is generated by PRIME.

6.6 Summary

This chapter has proposed several approaches to improve system reliability and scalability.

Firstly, a novel data layouts called PCDSDF that can survive disk failures caused by physical interconnect failures or correlated disk failures is proposed. PCDSDF combines the advantages of both parity declustering and orthogonal data layouts - it can not only survive dependent disk failures resulting from physical interconnect failures or correlated disk failures, but also has good degraded-mode and rebuilding performance. The generating process of PCDSDF is simple, deterministic and efficient in terms of both storage and time. The size of a rotation is small. Analysis shows that deploying PCDSDF data layouts can significantly improve system reliability. Moreover, the simulation results presented in this chapter show that the performance of PCDSDF

is comparable to other parity declustering data layouts such as PRIME. It is worth noting that NetApp is planing apply US patent for PCDSDF.

Secondly, a system architecture and rebuilding mechanism aimed at fast disk reconstruction is designed. This architecture is based on parity declustering data layouts and a disk-oriented reconstruction algorithm. To get a fast rebuilding speed, it uses stripe groups instead of stripes as the basic distribution unit, so that it can make use of the sequential nature of the rebuilding workload. Correspondingly, the original disk-oriented reconstruction algorithm is amended to suit this architecture. Moreover, it divides the whole disk array into a number of parity declustering groups to solve the loop bandwidth bottleneck, a problem during the disk reconstruction. The design space of system factors such as parity declustering ratio, chunk size, private buffer size of surviving disks and free buffer size have been explored in the performance evaluation section, which will provide a guideline for storage system design.

Lastly, an efficient distributed hot spare allocation and assignment algorithm for general parity declustering data layouts has been developed. This algorithm avoids conflict problems in the process of assigning distributed spare space for the units on the failed disk. Simulation results show that it effectively solves the write bottleneck problem. At the same time, there is only a small increase in the average response time of user requests. This algorithm has been applied for US patent with application number 12247877 [60].

Chapter 7

Conclusion

This chapter summarises the contents of this dissertation and identifies its notable contributions. Furthermore, this chapter outlines the future prospects and direction of this research.

7.1 Thesis Summary

RAID systems face unprecedented challenges from data intensive applications such as image processing, transaction processing and data warehousing. To provide the required capacity, the scale and complexity of RAID systems are growing at an unprecedented rate. As the scale of RAID systems increases, designers face both performance and reliability challenges. These challenges include limited back-end network bandwidth, physical interconnect failures, correlated disk failures and long disk reconstruction time.

Because simulation is more flexible than empirical measurement and is more accurate than analytical modelling, it was chosen as the main research method to be used for the work described in this dissertation. Based on the simulation model inherited from the Storlite project [45], a discrete event driven simulator for RAID systems has been developed, called SIMRAID. SIMRAID is based on HASE [25], which provides the underlying DES engine. HASE also provides a GUI for the rapid development of the model. SIMRAID uses benchmark generated workloads; both the SPC-1 benchmark and the Iometer benchmark are implemented in SIMRAID. A modular implementation allows models to be easily scaled. Each component of SIMRAID is highly parameterised, which enables it to explore a wide range of design space parameters. To improve the simulation speed, a set of abstraction techniques was developed to extract the

behaviour of the interconnection protocol without losing accuracy. Finally, to meet the technology trend toward heterogeneous storage architectures, a framework has been developed to easily model different types of device and interconnection technique.

SIMRAID was used first to study the scalability of the back-end network. In particular, FC SBOD [30] was chosen as the research subject, because it represents the current state of the art in scalable back-end storage sub-systems. The simulation model was initially configured with a network bandwidth that is higher than any currently available to study the factors that affect the bandwidth requirement of disks. These factors include workload feature, cache size, stripe unit size and RAID algorithm. The bandwidth was then reduced to study network saturation characteristics and to provide an answer to the question: given an interconnection network, how many disks can be connected to the system?

Following the network scalability study, system scalability was studied from the aspect of reliability. In particular, three approaches have been proposed to improve system reliability, and these will eventually improve system scalability. Firstly, a novel data layout is proposed called PCDSDF. PCDSDF combines the advantages of orthogonal data layouts and parity declustering data layouts so that the system can survive multiple disk failures caused by physical interconnect failures or correlated disk failures with a good degraded and rebuilding performance. Secondly, a system architecture and a rebuilding mechanism for fast disk reconstruction were designed and evaluated. This architecture is based on parity declustering data layouts and disk-oriented disk reconstruction algorithm. To get a fast rebuilding speed, it uses stripe groups instead of stripes as the basic distribution unit so that the system can make use of the sequential nature of the rebuilding workload. Correspondingly, the original disk-oriented reconstruction algorithm is amended to suit this architecture. Moreover, it divides the whole disk array into a number of parity declustering groups to solve the loop bandwidth bottleneck problem during disk reconstruction. The design space of system parameters such as parity declustering ratio, chunk size, private buffer size of surviving disks and free buffer size have been explored in the performance evaluation section, to provide a guideline for storage system design. Lastly, an efficient distributed hot spare allocation and assignment algorithm for general parity declustering data layouts has been developed. SIMRAID has been used to evaluate system performance when using these approaches.

7.2 Key Contributions

The key contributions of this dissertation to the field of storage system design are as follows:

Firstly, SIMRAID is an efficient performance evaluation tool for modern RAID systems. Although other simulation tools have been developed for RAID system study, such as Disksim [16] and RAIDFrame [28], they just focus on the performance of RAID algorithms and disk performance and lack of the capability of studying other components of RAID systems. SIMRAID contains all the essential components of a modern RAID system, including RAID controllers, caches, interconnection interfaces and networks, and disks. SIMRAID is highly configurable and parameterised, which enables it to easily explore a wide range of design space parameters. The abstraction techniques developed for the network communication protocol improve the simulation speed by 6 orders of magnitude compared with other FC network simulators without losing accuracy. Its framework, designed for modelling heterogeneous systems, allows models to be easily and rapidly extended and scaled. The accuracy of SIMRAID has been validated against a real system at application level, showing that it is accurate to within 5%. Finally, running SIMRAID only needs a very small amount of real memory and virtual memory. All these features makes SIMRAID an efficient performance evaluation tool for modern RAID systems. The research presented in this dissertation has confirmed its capability as a tool for studying RAID systems.

Secondly, the scalability of the back-end network of RAID systems has been studied using FC SBOD as the research subject. This research answers two questions: (1) given a number of disks, which factors affect the back-end network bandwidth requirements of disks? and (2) given an interconnection network, how many disks can be connected to the system? Through simulation, it was found that the bandwidth requirement per disk is primarily determined by workload features and stripe unit size, with cache size and RAID algorithms having very little effect on this value. Moreover, the number of disks that a system can scale to is limited by the back-end network bandwidth. A smaller stripe unit size has better scalability than a larger one. The maximum number of disks that a 2G FC port can support for a number of conventional system configurations was provided. Such accurate simulations provide practical guidance for storage sub-system design.

Lastly, three approaches to improving the reliability and scalability of RAID systems have been proposed. The first is a novel data layout called PCDSDF. PCDSDF

combines the advantages of orthogonal data layouts and parity declustering data layouts, so that it can not only survive multiple disk failures caused by physical interconnect failures or correlated disk failures, but also has a good degraded and rebuild performance. The generating process of PCDSDF is deterministic and time-efficient. The number of stripes per rotation (namely the number of stripes to achieve rebuild workload balance) is small. Simulations performed on SIMRAID confirm that the performance of PCDSDF is comparable to other parity declustering data layouts, like RELPR. Reliability analysis shows that deploying PCDSDF data layout can significantly improve system reliability, which will eventually improve system scalability.

The second is a system architecture and rebuilding mechanism for fast disk reconstruction. The design of this architecture is based on parity declustering data layouts and a disk oriented rebuilding algorithm. Unlike previous research which has focused on system performance during disk reconstruction, the design of this architecture focuses on improving the disk rebuilding speed. The design space explored using SIMRAID provides a guideline on how the system design parameters affect the system performance and rebuilding speed, which is useful for future RAID system design.

The third is an efficient distributed hot spare allocation and assignment algorithm for general parity declustering data layouts. This algorithm works for any evenly distributed parity declustering data layout. The assignment of distributed spare space avoids conflict problems in the process of assigning distributed spare space for the units on the failed disk. Simulation results shows that it effectively solves the write bottleneck problem. At the same time, there is only a small increase in the average response time of user requests.

7.3 Future Directions

This section outlines the future prospects and directions for the work described in this dissertation. These future research directions include extending SIMRAID to operate on the Serial Attached SCSI (SAS) network protocol, research on how to perform disk array resizing under parity declustering data layouts, designing an efficient copy-back scheduling mechanism and using Solid State Disks to improve system performance and reliability.

7.3.1 SAS Network Model Implemention

Besides FC, the main current alternative storage systems back-end network protocol is SAS [94]. It is therefore important to have an SAS model in SIMRAID in order to study future RAID systems. The framework of SIMRAID will allow an SAS model to be easily added without changing other components of the model.

SAS is very similar in nature to FC-AL, not particularly surprising as both are mechanisms to move SCSI commands from a command source to a target and to return the corresponding responses. In SAS there is no arbitration process to manage access to the shared communication resource. This comes from the original intent of SAS being used for point-to-point communication in which there is no contention for communication resource. However current SAS technology includes the use of expanders to allow switching of SAS signals between multiple SAS devices. Rather than using an arbitrated access protocol a SAS device simply transmits an OPN request. If the expander is able to connect the necessary devices, this is done and the OPN is forwarded to its intended target. If it is not possible to connect the devices, the OPN is rejected and the rejection is sent to the source of the OPN. From then on communication is remarkably similar to FC-AL, with a ready transmission followed by data transfer and a close. The main differences are in initialisation and in timing. Initially it will be necessary to set the arbitration delay to zero, thus leaving the arbitration process of the abstraction having a similar effect to the OPN transmit followed by connection or rejection of SAS. The initialisation time must also be adjusted following protocol analysis so that the time required for this matches a real SAS system. The main reason that it is possible to amend the abstraction developed in this dissertation to work with SAS is that both FCAL and SAS are methods of moving SCSI commands from a controller to a storage device. Therefore, it is possible to amend it for SAS protocol.

7.3.2 Disk Array Resizing under Parity Declustering Data Layout

It is very common to change the capacity of storage systems due to the changing of user requirements, especially to increase the capacity. This process of changing system capacity is called *disk array resizing*. It is easy to change the system capacity when non-parity declustering data layouts like RAID 5/RAID 6 are deployed. Depending on the requirement, a new disk can be added to an existing protection group or a new protection group can be added. For the former approach, the system needs to recalculate the parity information. For the latter approach, the system can directly use

the new storage space. However, it is very challenging to perform this resizing under parity declustering data layouts. The system is usually configured with a large parity declustering group to achieve a good degraded and rebuilding performance. Thus, it is not very common to add a complete new parity declustering group to the system. The only choice is then to add new disks to an existing parity declustering group. When a small number of disks such as a shelf of disks are added to the system, the system has to re-arrange the data layout and move some data to the new disks so that the workload and the rebuilding workload can be evenly distributed over all disks. This data layout re-arrangement needs a long time and a large amount of system resources as the parity declustering data layouts over different number of disks are very different. Research on how to minimise or reduce the amount of data to be moved and further reduce the system resource used for this data layout re-arrangement will be very useful for designing a flexible system.

7.3.3 Efficient Copy-back Scheduling Algorithm

This dissertation proposes an efficient distributed hot sparing allocation and assignment algorithm to eliminate the write bottleneck during disk reconstruction. After the failed disk are replaced with a new disk, the data reconstructed on the distributed spare space should be eventually copied back to the replacement disk. It is important to have a scheduling algorithm which can efficiently copy these data back to the replacement disk and minimise its effect of the system performance.

7.3.4 Using Solid State Disk to Improve the System Performance and Reliability

As described in Section 2.2.1.2, Solid State Disks (SSDs) use DRAM memory backed by battery or flash memory to store persistent data and have an average access time of less than 0.15 ms, which is 250 times faster than that of hard disk drives. As the price of SSDs keeps decreasing, there is increasing interest in using SSDs in RAID systems. They could be used in several ways: as a new layer of large cache, to store hot data or simply as substitutes for hard disks. However, which approaches is the most effective way to improve system performance and reliability is as yet unknown. Thus, it would be interesting to build SSD models in SIMRAID and study how to use them effectively in RAID systems.

7.4 The Future of RAID Systems

As observed at the beginning of this dissertation, the scale and complexity of RAID systems are growing at an unprecedented rate. Future RAID systems are going to be larger and larger. A good example of such a large system is an archive system currently being planned for a generic broadcasting company. The purpose of this system is to store all the video and audio files that the company has produced, which is 500 PBytes of data. About 20% of this data is going to be stored on disks and the rest will be stored on tape. That means that 100 PBytes of data needs to be stored on disk. Assuming that disks with capacity of 1 TBytes are used, more than 100,000 disks will be needed to store this data and redundancy information, and more than 7000 shelves will be needed to house these disks. Assuming that failed disks can be rebuilt using all the disk bandwidth, it will take six hours to rebuild a 1 TBytes disk. Assuming the MTTF of the disks and shelves is 500,000 hours, the MTTF of this system will only be about 70 hours if RAID 5 (assuming conventional 7+1 RAID 5) is used to protect the system. However, using the techniques described in this dissertation can reduce the rebuilding time and improve the system reliability. In particular, the system can be divided into a number of PC groups, and each PC group implement the PCDSDF data layout and the architecture for fast rebuilding. By so doing, the system reliability can be significantly improved. For example, if 56 disks are selected from 8 shelves, with each shelf contributing 7 disks to form a PC group, then using PCDSDF, not only will the system be able to survive a shelf failure, but the disk rebuilding speed will be improved by a factor of 7. In terms of MTTF, then based on equation 7.1:

$$\begin{aligned}\textbf{Failure rate}_{\textbf{PCDSDF}} &= \frac{n}{MTTF_{disk}} * \frac{(n_{PC}-D)MTTR_{disk_{PCDSDF}}}{MTTF_{disk}} \\ &+ \frac{L}{MTTF_{shelf}} * \frac{(k-1)MTTR_{shelf}}{MTTF_{shelf}} \\ &+ \frac{n}{MTTF_{disk}} * \frac{(k-1)MTTR_{disk_{PCDSDF}}}{MTTF_{shelf}} \\ &+ \frac{L}{MTTF_{shelf}} * \frac{(n_{PC}-D)MTTR_{shelf}}{MTTF_{disk}}\end{aligned} \tag{7.1}$$

where n=100,000, n_{PC}=56, D=7, k=8, $MTTF_{disk}$=500,000, $MTTF_{shelf}$=500,000, $MTTR_{disk_{PCDSDF}}$=6/7 and $MTTR_{shelf}$=24,
the MTTF will be more than 17,000 hours. Clearly, techniques such as those described

here will be essential in future very large scale RAID systems.

Appendix A

Proof of Simple PCDSDF Property II

We first introduce the following four lemmas:

Lemma 1. $(x-y)\ mod\ D = ((x\ mod\ D) - (y\ mod\ D))\ mod\ D$

Lemma 2. $(x+y)\ mod\ D = ((x\ mod\ D) + (y\ mod\ D))\ mod\ D$

Lemma 3. $\forall i$ and $j \in Z$, $0 \leq i < D$, $0 \leq j < D$, and $i \neq j$
s.t. $(i+\Delta)\ mod\ D \neq (j+\Delta)\ mod\ D, \Delta$ is a integer.

Proof. Assume that $(i+\Delta)\ mod\ D = (j+\Delta)\ mod\ D$, then
$((i+\Delta)\ mod\ D - (j+\Delta)\ mod\ D)\ mod\ D = ((i+\Delta)-(j+\Delta))\ mod\ D = (i-j)\ mod\ D = 0$.

Because $0 \leq i < D$, $0 \leq j < D$, we get $i - j = 0$, which is not true. Therefore, $(i+\Delta)\ mod\ D \neq (j+\Delta)\ mod\ D$.

Lemma 4. $\forall i$ and $j \in Z$, $0 \leq i < D$, $0 \leq j < D$, $i \neq j$, $\Delta \in Z$, and $0 < \Delta < D$.
s.t. $(i*\Delta)\ mod\ D \neq (j*\Delta)\ mod\ D$,

Proof. Assume $(i*\Delta)\ mod\ D = (j*\Delta)\ mod\ D$, then
$(i*\Delta - j*\Delta)\ mod\ D = ((i-j)\Delta)\ mod\ D = 0$.

As D is a prime number and $\Delta < D$, we get $i = j$, which is not true. Therefore, $(i*\Delta)\ mod\ D \neq (j*\Delta)\ mod\ D$.

Proof of Simple PCDSDF Property II: The proof translates to proving that $\forall$ pairs of disks $m_1.x$ and $m_2.y$, where m_1 and m_2 refer to the loop id/number and x and y are the disk indexes, $\exists i$ and j, such that $(j + m_1 i)\ mod\ D = x$ and $(j +$

$m_2 i)\ mod\ D = y$, which implies that disks $m_1.x$ and $m_2.y$ appear in the same combination.

Assume that $(y-x)\ mod\ D = \Delta$, where $\Delta \in [0, D-1]$. From Lemma A we know that $\exists i$ such that $(i(m_2 - m_1))\ mod\ D = \Delta$. This i is what we are looking for.

From Lemma A we know that $\exists j$ such that $(j + m_1 i)\ mod\ D = x$. Then $(j + m_2 i)\ mod\ D = (j + (m_1 + m_2 - m_1)i)\ mod\ D = (j + m_1 i + i(m_2 - m_1))\ mod\ D = ((j + m_1 i)\ mod\ D + (i(m_2 - m_1))\ mod\ D)\ mod\ D = (x + \Delta)\ mod\ D = y$.

Bibliography

[1] Fibre channel arbitrated loop (FC-AL) ANSI(NCITS) T11 working draft proposal, Rev7.0, amendment.

[2] What is a solid state disk? Available at http://www.superssd.com/whatisassd.htm, [online; accessed 12-March-2009].

[3] SPC-1 benchmark executive summary of IBM Enterprise Storage Server F20, Dec 2001. Available at http://www.storageperformance.org/results/a00001_IBM_SPC1_executive_summary.pdf, [online; accessed 12-March-2009].

[4] Historical notes about the cost of hard drive storage space, March 2003. Available at http://www.littletechshoppe.com/ns1625/winchest.html, [online; accessed 12-March-2009].

[5] SPC benchmark 1 (SPC-1) official specification version 1.7, July 2003. Available at http://www.storageperformance.org/Specifications/SPC-1_v170.pdf, [online; accessed 12-March-2009].

[6] SPC benchmark 2 (SPC-2) draft version 0.9.0, 2004. Available at http://www.storageperformance.org/Specifications/SPC-2_v090.pdf, [online; accessed 12-March-2009].

[7] SPC benchmark 3br (SPC-3BR) draft specification public review version, 2007. Available at http://www.storageperformance.org/specs/

SPC-3BR_Public-Review-Draft_2007-12-03.pdf, [online; accessed 12-March-2009].

[8] Adaptec. Software RAID vs. hardware RAID, 2002. Available at http:://www.adaptec.com/pdfs/raid_soft_v_hard.pdf, [accessed 5-March-2008].

[9] Adaptec. Hardware RAID vs. software RAID: Which implementation is best for my application, 2006. Available at http:://www.adaptec.com/NR/rdonlyres/14B2FD84-F7A0-4AC5-A07A-214123EA3DD6/0/4423_SW_HWRAID_10.pdf, [accessed 12-March-2009] .

[10] S. Alam, R. Ibbett, and F. Mallet. Simulation of a computer architecture for quantum chromodynamics calculations. *Crossroads*, 9(3):16–23, 2003.

[11] G. A. Alvarez, W. A. Burkhard, L. J. Stockmeyer, and F. Cristian. Declustered disk array architectures with optimal and near-optimal parallelism. In *ISCA'98: Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 109–120, New York, NY, USA, 1998. ACM Press.

[12] D. Anderson, J. Dykes, and E. Riedel. More than an interface—SCSI vs. ATA. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 245–257, Berkeley, CA, USA, 2003. USENIX Association.

[13] E. Bachmat and J. Schindler. Analysis of methods for scheduling low priority disk drive tasks. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 55–65, New York, NY, USA, 2002. ACM.

[14] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler. An analysis of latent sector errors in disk drives. *ACM SIGMETRICS Performance Evaluation Review4*, 35(1):289–300, 2007.

[15] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on Computers*, 44(2):192–202, Feb 1995.

[16] J. S. Bucy, G. R. Ganger, and Contributors. The disksim simulation environment version 3.0 reference manual, 2003. Available at

`http://www.pdl.cmu.edu/PDL-FTP/DriveChar/CMU-CS-03-102.pdf, [accessed Mar-12-2009]`.

[17] P. Cao, S. B. Lin, S. Venkataraman, and J. Wilkes. The TickerTAIP parallel RAID architecture. *ACM Transactions on Computer Systems*, 12(3):236–269, 1994.

[18] L. R. Carley, G. R. Ganger, and D. F. Nagle. MEMS-based integrated-circuit mass-storage systems. *Communications of the ACM*, 43(11):72–80, 2000.

[19] G. A. Castets, D. Leplaideur, J. A. Bras, and J. Galang. IBM enterprise storage server, Dec 2001. Available at `http://www.redbooks.ibm.com/redbooks/pdfs/sg245465.pdf, [online; accessed 12-March-2009]`.

[20] P. M. Chen, G. A. Gibson, R. H. Katz, and D. A. Patterson. An evaluation of redundant arrays of disks using an amdahl 5890. In *SIGMETRICS '90: Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 74–85, New York, NY, USA, 1990. ACM Press.

[21] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, 1994.

[22] S. Chen and D. Towsley. The design and evaluation of RAID 5 and parity striping disk array architectures. *Journal of Parallel and Distributed Computing*, 17(1-2):58–74, 1993.

[23] S. Chen and D. Towsley. A performance evaluation of RAID architectures. *IEEE Transactions on Computers*, 45(10):1116–1130, 1996.

[24] F. Chevalier, R. Ibbett, and T. Courtney. Iometer performance comparison of SBOD and MBOD. In *AIC'04: Proceedings of the 4th WSEAS International Conference on Applied Informatics and Communications*, pages 1–6, Stevens Point, Wisconsin, USA, 2004. World Scientific and Engineering Academy and Society (WSEAS).

[25] P. Coe, F. Howell, R. Ibbett, and L. Williams. A hierarchical computer architecture design and simulation environment. *ACM Transactions on Modeling and Computer Simulation*, 8(4):431–446, Oct 1998.

[26] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-diagonal parity for double disk failure correction. In *Proceedings of the USENIX FAST '04 Conference on File and Storage Technologies*, pages 1–14, San Francisco, CA, Mar 2004.

[27] T. Courtney, F. Chevalier, and Y. Li. Novel techniquefor accelerated simulation of storage systems. In *IASTED PDCN'06: Proceedings of Parallel and Distributed Computing and Networks*, pages 266 – 272, Feb 2006.

[28] W. V. Courtright II, G. A. Gibson, M. Holland, and J. Zelenka. RAIDframe: rapid prototyping for disk arrays. volume 24, pages 268–269, New York, NY, USA, 1996. ACM.

[29] EMC. EMC symmetrix DMX-4 specification sheet, March 2005. Available at `http://www.emc.com/collateral/hardware/specification-sheet/c1166-dmx4-ss.pdf, [online; accessed 12-March-2009]`.

[30] Emulex. InSpeed^{TM}SOC 320 embedded storage switch, March 2003. Available at `http://www.emulex.com/products/embeddedswitch/320/ds.pdf, [online; accessed 12-March-2009]` .

[31] W. Gang, L. Xiao-Guang, and L. Jing. Parity declustering data layout for tolerating dependent disk failures in network RAID systems. In *Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing*, Feb 2002.

[32] G. R. Ganger, B. L. Worthington, and Y. N. Patt. The DiskSim simulation environment version 2.0 reference manual. Technical report, Carnegie Mellon University / University of Michigan, Dec 1999.

[33] G. A. Gibson. *Redundant disk arrays: reliable, parallel secondary storage*. PhD thesis, Berkeley, CA, USA, 1992.

[34] G. Gole. Estimating drive reliability in desktop computers and consumer electronics systems. Technical report, Seagate, 2000.

[35] M. Hall. *Combinatorial Theory*. Wiley, 1986.

[36] J. Handy. Flash memory vs. hard disk drives - which will win, 2006. Available at
`http://www.storagesearch.com/semico-art1.html, [online; accessed 12-March-2009]`.

[37] J. R. Heath and P. J. Yakutis. High speed storage area networks using a fibre channel arbitrated loop interconnect. *IEEE Network*, 14(2):51–56, 2000.

[38] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, the third edition*. Morgan Kaufmann, 2002.

[39] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, the fourth edition*. Morgan Kaufmann, 2006.

[40] M. Hofri. Disk scheduling: FCFS vs. SSTF revisited. *Communications of the ACM*, 23(11):645–653, 1980.

[41] M. Holland and G. A. Gibson. Parity declustering for continuous operation in redundant disk arrays. In *ASPLOS-V: Proceedings of the fifth international conference on Architectural support for programming languages and operating systems*, pages 23–35, 1992.

[42] M. Holland, G. A. Gibson, and D. P. Siewiorek. Fast, on-line failure recovery in redundant disk arrays. In *Proceedings of the 23rd Annual International Symposium on Fault-Tolerant Computing*, pages 421–433, 1993.

[43] K. Hopkin and R. Pitwon. Pluggable optical backplane technology, 2005. Available at
`http://www.andataco.com/pdfs/whitepapers/Xyratex_white_paper_Optical_Backplane.pdf, [online; accessed 12-March-2009]`.

[44] R. Y. Hou and Y. N. Patt. Trading disk capacity for performance. In *Proceedings the 2nd International Symposium on High Performance Distributed Computing*, pages 263–270, 1993.

[45] R. Ibbett. Storlite project, 2006. Available at
`http://www.icsa.inf.ed.ac.uk/research/groups/hase/projects/storlite/index.html,[online; accessed 12-March-2009]`.

[46] G. Ingargiola. Shugart's law. Available at `http://knight.cis.temple.edu/ ingargio/cis307/readings /firstlecturef06.html,[online; accessed 12-March-2009]`.

[47] Intel. Iometer user's guid, 2001. Available at `http://www.iometer.org/doc/documents.html, [online; accessed 12-March-2009]`.

[48] Intel. Intel 80331 I/O processor developer's manual, 2003. Available at `http://www.intel.com/design/iio/manuals/273942.htm, [online; accessed 12-March-2009]`.

[49] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky. Are disks the dominant contributor for storage failures? a comprehensive study of storage subsystem failure characteristics. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 111–125, Berkeley, CA, USA, 2008. USENIX Association.

[50] R. W. Kembel and H. L. Truestedt. *In-Depth Fibre channel arbitrated loop*. Northwest Learning Associates, 1996.

[51] M. Y. Kim. Synchronized disk interleaving. *IEEE Transactions on Computers*, 35(11):978–988, 1986.

[52] M. Y. Kim and A. N. Tantawi. Asynchronous disk interleaving: Approximating access delays. *IEEE Transactions on Computers*, 40(7):801–810, 1991.

[53] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[54] A. Kuratti and W. H. Sanders. Performance analysis of the RAID 5 disk array. In *IPDS '95: Proceedings of the International Computer Performance and Dependability Symposium on Computer Performance and Dependability Symposium*, page 236, Washington, DC, USA, 1995. IEEE Computer Society.

[55] E. K. Lee and R. H. Katz. Performance consequences of parity placement in disk arrays. In *ASPLOS-IV: Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, pages 190–199, New York, NY, USA, 1991. ACM Press.

[56] E. K. Lee and R. H. Katz. An analytic performance model of disk arrays. In *SIGMETRICS '93: Proceedings of the 1993 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 98–109, New York, NY, USA, 1993. ACM Press.

[57] Y. Li, T. Courtney, F. Chevalier, and R. N. Ibbett. SIMRAID: An efficient performance evaluation tool for RAID systems. In *Summer Computer Simulation Conference*, pages 431–438, 2006.

[58] Y. Li, T. Courtney, R. N. Ibbett, and N. Topham. On the scalability of storage sub-system back-end network. In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies*, pages 7–7, Berkeley, CA, USA, 2007. USENIX Association.

[59] Y. Li, T. Courtney, R. N. Ibbett, and N. Topham. On the scalability of the back-end network of storage sub-systems. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, 2008.

[60] Y. Li and A. Goel. An efficient distributed hot sparing scheme in a parity declustered raid organization. *US Patent No. 12247877*, Oct 2008.

[61] M. Livny and et al. Multi-disk management algorithms. In *SIGMETRICS '87: Proceedings of the 1987 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 69–77, New York, NY, USA, 1987. ACM Press.

[62] F. MacWilliams and J. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

[63] F. Mallet, S. Alam, and R. Ibbett. An extensible clock mechanism for computer architecture simulations. In *In Proceedings of the IASTED International Conference on Modelling and Simulation*, pages 291–305. IASTED, May 2002.

[64] W. Marurngsith. *Simulation Modelling of Distributed-Shared Memory Multiprocessors*. PhD thesis, University of Edinburgh. College of Science and Engineering. School of Informatics, 2006.

[65] W. Marurngsith and R. N. Ibbett. DSIMCLUSTER: A simulation model for efficient memory analysis experiments of DSM clusters. In *In Proceedings of*

the 2004 Summer Computer Simulation Conference (SCSC 04), pages 191–197. SCS, July 2004.

[66] J. Menon. Performance of RAID5 disk arrays with read and write caching. *Distributed and Parallel Databases*, 2(3):261–293, 1994.

[67] J. Menon and J. Cortney. The architecture of a fault-tolerant cached RAID controller. In *ISCA '93: Proceedings of the 20th annual international symposium on Computer architecture*, pages 76–87, New York, NY, USA, 1993. ACM.

[68] J. Menon and D. Mattson. Comparison of sparing alternatives for disk arrays. In *ISCA '92: Proceedings of the 19th annual international symposium on Computer architecture*, pages 318–329, New York, NY, USA, 1992. ACM Press.

[69] J. Menon, J. Roche, and J. Kasson. Floating parity and data disk arrays. *Journal of Parallel and Distributed Computing*, 17(1-2):129–139, 1993.

[70] A. Merchant and P. S. Yu. An analytical model of reconstruction time in mirrored disks. In *Performance '93: Proceedings of the 16th IFIP Working Group 7.3 international symposium on Computer performance modeling measurement and evaluation*, pages 115–129, Amsterdam, The Netherlands, The Netherlands, 1994. Elsevier Science Publishers B. V.

[71] A. Merchant and P. S. Yu. Analytic modeling of clustered RAID with mapping based on nearly random permutation. *IEEE Transactions on Computers*, 45(3):367–373, 1996.

[72] M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski. Scale-up x scale-out: A case study using nutch/lucene. In *IPDPS'07: Proceeding of IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, Washington, DC, USA, 2007. IEEE Computer Society.

[73] R. R. Muntz and J. C. S. Lui. Performance analysis of disk arrays under failure. In *Proceedings of the sixteenth international conference on Very large databases*, pages 162–173, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[74] NetAPP. NetApp FAS6000 series technical specifications. Available at `http://www.netapp.com/us/products/storage-systems/fas6000/fas6000-tech-specs.html`, `[online; accessed 12-March-2009]`.

[75] OMNeT++. OMNeT++. Available at `http://www.omnetpp.org/, [online; accessed 12-March-2009]`.

[76] C. I. Park. Efficient placement of parity and data to tolerate two disk failures in disk array systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(11):1177–1184, 1995.

[77] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). pages 109–116, 1988.

[78] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies*, pages 2–2, Berkeley, CA, USA, 2007. USENIX Association.

[79] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *SoftwarełPractice & Experience*, 27(9):995–1012, 1997.

[80] J. S. Plank. The RAID-6 liberation codes. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.

[81] O. Pretzel. *Error-Correction Codes and Finite Fields*. Oxford University press, 1992.

[82] T. M. Ruwart. Performance characterization of large and long fibre channel arbitrated loops. In *16th IEEE Symposium on Mass Storage Systems*, pages 11–21. IEEE Computer Society, March 1999.

[83] SANDirect.com. Sanblade qla2342-e-sp 2gb pci-x [emc firmware]. Available at `http://www.sandirect.com/product_info.php?cPath=257&products_id=145, [online; accessed 27-December-2009]`.

[84] SANDirect.com. Sanblade qle2564-ck 8gb pcie hba. Available at `http://www.sandirect.com/product_info.php?cPath=213_257_261_270% products_id=1312, [online; accessed 27-December-2009]`.

[85] S. Savage and J. Wilkes. AFRAID: a frequently redundant array of independent disks. In *ATEC '96: Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, pages 3–3, Berkeley, CA, USA, 1996. USENIX Association.

[86] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you. In *FAST'07: Proceeding of 5th USENIX Conference on File and Storage Technologies*, pages 62–72, Feb 2007.

[87] M. Schulze, G. Gibson, R. Katz, and D. Patterson. How reliable is a RAID. In *COMPCON Thirty-Fourth IEEE Computer Society International Conference*, pages 118–123, Feb 1989.

[88] E. J. Schwabe and I. M. Sutherland. Improved parity-declustered layouts for disk arrays. In *SPAA '94: Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, pages 76–84, New York, NY, USA, 1994. ACM.

[89] T. J. E. Schwarz, J. Steinberg, and W. A. Burkhard. Permutation development data layout (PDDL). In *HPCA '99: Proceedings of the 5th International Symposium on High Performance Computer Architecture*, page 214, Washington, DC, USA, 1999. IEEE Computer Society.

[90] Seagate. Cheetah 15k.3 FC product manual, March 2003. Available at `http://www.seagate.com/support/disc/manuals/fc/100148129b.pdf, [online; accessed 12-March-2009]`.

[91] simSANs. Simulating storage area networks. Available at `http://simsan.storwav.com/,[online; accessed 12-March-2009]`.

[92] J. H. Son and M. H. Kim. An analysis of the optimal number of servers in distributed client/server environments. *Decis. Support Syst.*, 36(3):297–312, 2004.

[93] D. Stodolsky, G. A. Gibson, and M. Holland. Parity logging overcoming the small write problem in redundant disk arrays. In *ISCA '93: Proceedings of the 20th annual international symposium on Computer architecture*, May 1993.

[94] T10. Serial attached SCSI. Available at `http://www.t10.org/drafts.htm#sas1, [online; accessed 12-March-2009]`.

[95] A. Thomasian, C. Han, G. Fu, and C. Liu. A performance evaluation tool for RAID disk arrays. In *Proceedings of the First International Conference on the Quantitative Evaluation of Systems (QEST)*, pages 8–17, Washington, DC, USA, 2004. IEEE Computer Society.

[96] A. Thomasian and J. Menon. Performance analysis of RAID5 disk arrays with a vacationing server model for rebuild mode operation. In *Proceedings of the Tenth International Conference on Data Engineering*, pages 111–119, Washington, DC, USA, 1994. IEEE Computer Society.

[97] A. Thomasian and J. Menon. RAID5 performance with distributed sparing. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):640–657, 1997.

[98] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song. PRO: a popularity-based multi-threaded reconstruction optimization for RAID-structured storage systems. In *FAST'07: Proceedings of the 5th conference on USENIX Conference on File and Storage Technologies*, pages 32–32, Berkeley, CA, USA, 2007. USENIX Association.

[99] M. Uysal, G. A. Alvarez, and A. Merchant. A modular, analytical throughput model for modern disk arrays. In *MASCOTS '01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*, page 183, Washington, DC, USA, 2001. IEEE Computer Society.

[100] E. Varki and et al. Issues and challenges in the performance analysis of real disk arrays. *IEEE Trans. Parallel Distrib. Syst.*, 15(6):559–574, 2004.

[101] C.-Y. Wang, F. Zhou, Y.-L. Zhu, C. T. Chong, B. Hou, and W.-Y. Xi. Simulation of fibre channel storage area network using SANSim. In *Proceedings of the 11th IEEE International Conference on Networks*. IEEE Compute Society, Sep 2003.

[102] J. Wilkes. The Pantheon storage-system simulator. Technical Report HPL-SSP-95-14, Storage Systems Program, Computer Systems Laboratory, Hewlett-Packard Laboratories, Palo Alto, CA, May 1996.

[103] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The hp AutoRAID hierarchical storage system. *ACM Trans. Comput. Syst.*, 14(1):108–136, 1996.

[104] Q. Xin, E. Miller, T. Schwarz, D. Long, S. Brandt, and W. Litwin. Reliability mechanisms for very large storage sytems. In *Proceedings of the 20th IEEE/11th NASA Boddard Conference on Mass Storage System and Technologies*, pages 146–156, Apr 2003.

[105] L. Xu and J. Bruck. X-Code: Mds array codes with optimal encoding. *IEEE Transc. on Information Theory*, 45(1):272–276, 1999.

[106] J. Yang and F.-B. Sun. A comprehensive review of hard-disk drive reliability. In *Proceedings of Reliability and Maintainability Symposium*, pages 403–409, Jan 1999.

[107] P. S. Yu and A. Merchant. Analytic modeling and comparisons of striping strategies for replicated disk arrays. *IEEE Transaction on Computers*, 44(3):419–433, 1995.