



THESIS  
2  
7002

This is to certify that the

dissertation entitled

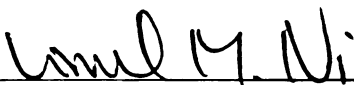
CcStore: A STORAGE CLUSTER ARCHITECTURE  
USING NETWORK ATTACHED STORAGE DEVICES

presented by

Yong Chen

has been accepted towards fulfillment  
of the requirements for

Doctoral degree in Computer Science  
& Engineering

  
Major professor

Date May 7, 2002

**LIBRARY**  
**Michigan State**  
**University**

**PLACE IN RETURN BOX** to remove this checkout from your record.  
**TO AVOID FINES** return on or before date due.  
**MAY BE RECALLED** with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE

CONJUGATE  
CROSS REACT

CONJUGATE  
CROSS REACT

in blood

in blood

**CoStore: A STORAGE CLUSTER ARCHITECTURE  
USING NETWORK ATTACHED STORAGE DEVICES**

**By**

**Yong Chen**

**A DISSERTATION**

**Submitted to  
Michigan State University  
in partial fulfillment of the requirements  
for the degree of**

**DOCTOR OF PHILOSOPHY**

**Department of Computer Science and Engineering**

**2002**

## **ABSTRACT**

### **CoStore: A STORAGE CLUSTER ARCHITECTURE USING NETWORK ATTACHED STORAGE DEVICES**

**By**

**Yong Chen**

The exponential growth of the Internet has presented numerous challenges to computer scientists and engineers. One of the major challenges is to design advanced storage systems to meet the demanding requirements for high performance, high capacity, and strong reliability.

This dissertation proposes the CoStore cluster architecture to construct reliable storage systems using network attached storage devices. The performance of a CoStore prototype has been measured and the preliminary results show that CoStore's performance is comparable to that of the efficient NFS and that the CoStore cluster architecture is highly scalable in terms of the cluster size.

This dissertation study investigates remote replication using CoStore clusters to construct highly reliable and highly available storage systems. It has been demonstrated that if a cluster is mirrored over the network with sufficient bandwidth and low latency, the remote replica cluster can considerably reinforce preparedness for disaster recovery without sacrificing performance.

This dissertation study also confirms the feasibility of deploying CoStore architecture to construct reliable storage service utilizing idle disk space on workstation clusters on existing desktop computing infrastructure.

**Copyright © 2002 Yong Chen**

**All rights reserved.**

## ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my advisor Prof. Lionel M. Ni for his years of advising, guiding and continuous strong support. I also would like to thank other guidance committee members, in particular, Prof. Abdol H. Esfahanian, Prof. Matt W. Mutka, and Prof. Zhengfang Zhou for having spent time reviewing this dissertation and providing feedbacks.

I am grateful for the unconditional love and support from my wife, my parents and parents-in-law, and all other members in my family in my dear home country. Without their spiritual support, this work would not have been possible. I especially thank my wife Danhua for her continuous support, encouragement and patience, in addition to her love. I also would like to thank my younger sister Jun Chen for her encouragement.

I am very glad that I have had the opportunity working as a graduate system administrator in the Department of Computer Science and Engineering at Michigan State University. This experience has been proved invaluable in the dissertation study and in my career.

# TABLE OF CONTENTS

LIST OF TABLES .....	viii
LIST OF FIGURES .....	ix
Chapter 1 Introduction and Motivations.....	1
1.1 Introduction .....	1
1.2 Motivations .....	2
1.2.1 Evolving storage architectures .....	2
1.2.2 Data is the most valuable asset .....	4
1.2.3 Storage space utilization irony .....	4
1.3 Problem Abstraction.....	5
1.4 Dissertation Overview .....	6
Chapter 2 Background.....	7
2.1 Storage System Components.....	7
2.1.1 Storage devices.....	7
2.1.1.1 Disk drives .....	7
2.1.1.2 Disk arrays.....	9
2.1.1.2.1 RAID level 4 and level 5 .....	11
2.1.1.2.2 RAID status.....	13
2.1.2 I/O interfaces .....	14
2.1.2.1 Parallel interfaces .....	15
2.1.2.2 Serial interfaces .....	17
2.1.2.3 Merger of I/O Channels and Ethernet .....	20
2.2 Local File Systems .....	22
2.2.1 File system layers in operating systems .....	22
2.2.2 Virtual File System (VFS) .....	24
2.2.3 File system structures .....	24
2.2.3.1 Ext2 file system.....	26
2.2.4 File system management.....	30
2.2.4.1 Journaling .....	31
2.2.4.2 Soft Updates.....	33
2.2.4.3 Ext3: journaling Ext2.....	33
2.2.4.3.1 Backwards compatibility .....	33
2.2.4.3.2 The Ext3 approach .....	34
2.2.4.3.2 Journaling for both meta-data and data.....	36
2.3 Distributed File Systems .....	36
2.3.1 Network File System (NFS).....	38
2.3.2 Andrew and Coda File Systems.....	39

2.3.3 Zebra and xFS.....	40
2.3.4 Petal and Frangipani .....	41
2.3.5 Network Attached Secure Disks (NASD) .....	42
<b>Chapter 3 The CoStore Storage Cluster Architecture and Performance</b>	
<b>Evaluation .....</b>	<b>46</b>
3.1 Introduction .....	46
3.2 Technology Trends in Networked Storage .....	47
3.2.1 Storage interfaces: block vs. file .....	47
3.2.2 Architecture taxonomy.....	48
3.2.3 Storage Area Network (SAN).....	50
3.2.4 Network Attached Storage (NAS) .....	51
3.2.5 Upcoming storage architectures .....	52
3.3 A Storage Cluster Architecture Using Network Attached Storage Devices..	54
3.3.1 Architecture overview .....	54
3.3.2 Distributed file subsystem .....	56
3.3.3 Local file subsystem .....	57
3.3.4 RAID subsystem.....	60
3.3.4.1 Block buffer cache .....	62
3.3.5 System operations.....	64
3.3.5.1 Cluster initialization.....	65
3.3.5.2 Client initialization .....	65
3.3.5.3 File and directory creation .....	66
3.3.5.4 Intra-cluster communications and deadlock.....	67
3.3.5.5 Recovery from node failures .....	68
3.4 Prototype Implementation .....	69
3.5 Performance Evaluation.....	70
3.5.1 Experimental setup.....	70
3.5.2 Write performance: CoStore vs. NFS vs. CIFS .....	74
3.5.3 Read performance: CoStore vs. NFS vs. CIFS .....	74
3.5.4 Impact of distributed RAID and commit policy .....	77
3.5.5 Scalability of CoStore .....	79
3.5.6 Parity daemon's bottleneck effect in RAID-4 .....	81
3.6 Summary .....	82
<b>Chapter 4 CoStore Clusters Utilizing Idle Disk Space on Workstations .....</b>	<b>83</b>
4.1 Introduction .....	83
4.2 Assumptions and Environment Description.....	85
4.3 Alternative Solutions .....	87
4.3.1 Ad hoc solutions .....	87
4.3.2 NBD based solutions.....	87
4.3.3 Peer-to-peer solutions .....	89
4.3 Feasibility Assessment of Deployment on Existing Desktop Computing	
Infrastructure .....	90
4.3.1 Reliability theory in RAID.....	90

4.3.2 Feasibility assessment .....	94
4.4 Summary .....	96
 Chapter 5     Reliable and Highly Available Storage Systems Using CoStore Clusters .....	97
5.1 Introduction .....	97
5.2 Backup and Archiving Techniques for Recovery .....	97
5.2.1 Tape backups .....	99
5.2.2 Snapshots .....	100
5.2.2 Redundancy .....	102
5.3 Reliable and Highly Available Storage Systems Using CoStore Clusters..	103
5.3.1 Benefits and implications .....	104
5.4 Experimental Setup.....	107
5.4.1 Network emulator EMIP .....	109
5.5 Performance Evaluation.....	112
5.5.1 Local mirroring.....	112
5.5.2 Remote mirroring.....	116
5.5.3 Performance enhancing techniques .....	118
5.5 Summary .....	119
 Chapter 6     Conclusions and Future Work.....	121
6.1 Related Work .....	121
6.1.1 Storage cluster architecture .....	121
6.1.2 Remote storage replication .....	123
6.1.3 Storage service utilizing idle space .....	124
6.2 Conclusions.....	124
6.2.1 Storage cluster architecture .....	125
6.2.2 Remote storage replication .....	125
6.2.3 Storage cluster utilizing idle space .....	126
6.3 Contributions .....	126
6.4 Future Work .....	127
6.4.1 Implementations .....	127
6.4.2 Future research areas .....	127
 Bibliography .....	129

## LIST OF TABLES

2-1	SCSI parameters. DT means double transition clocking.....	16
4-1	$MTTF_{ws}$ and $MTTR_{ws}$ based on system uptime data .....	95
4-2	The reliability ( $MTTF_{cluster}$ ) of clusters .....	96
5-1	Test sets and the characteristics of network environments.....	108

## LIST OF FIGURES

2-1	(L) Structures inside disk; (R) mechanisms in a magnetic disk drive.....	7
2-2	The layout of data and parity blocks in a RAID level 5 .....	9
2-3	RAID level 5 host writes and reads in normal condition .....	11
2-4	RAID level 5 host writes and reads in the presence of a fault .....	12
2-5	File system layers in UNIX .....	22
2-6	(a) The schematic structure of a file system; (b) the structure of Ext2 file system .	26
2-7	The superblock in Ext2 file system.....	28
2-8	(a) The structure of a Unix inode; (b) the inode in Ext2 file system.....	29
2-9	A directory entry in the Ext2 file system .....	30
2-10	Network-attached secure disks (NASD).....	43
2-11	NASD/Cheops architecture .....	45
3-1	Server-attached disks (SAD) .....	49
3-2	Network Attached Storage (NAS) vs. Storage Area Network (SAN).....	52
3-3	The structure of a CoStore cluster .....	54
3-4	Functional modules in a CoStore daemon .....	56
3-5	Block addressing and <i>inode</i> numbering scheme.....	58
3-6	File system layout in a CoStore cluster with RAID level 4 .....	59
3-7	Architectural similarities between CoStore and DASH .....	60
3-8	Linklists in block buffer cache .....	63
3-9	The write performance of CoStore.....	73
3-10	The read performance of CoStore .....	76

3-11	The impact of distributed RAID overhead and commit policy .....	78
3-12	The scalability of CoStore clusters.....	80
3-13	Parity daemon's bottleneck effect in CoStore with RAID level 4 .....	81
4-1	Queuing theory in the reliability of disk arrays. ....	90
4-2	The relationship between $MTTF_{cluster}$ and $MTTF_{ws}$ .....	93
5-1	A CoStore cluster with RAID level 4+1 .....	104
5-2	RAID buffers and block buffer cache in CoStore.....	107
5-3	The network diagram of testing environments .....	109
5-4	The internals of EMIP .....	111
5-5	A CoStore cluster with RAID level 0/0+1 in a switched LAN.....	113
5-6	A CoStore cluster with RAID level 4/4+1 in a switched LAN.....	115
5-7	A CoStore cluster mirrored in various network environments.....	117

# **Chapter 1**

## **Introduction and Motivations**

### **1.1 Introduction**

The Internet has been a phenomenon that has cultivated a worldwide information revolution. At unprecedented paces and in unparalleled scales, the information digitalization processes have presented numerous new challenges to IT scientists and engineers. One of the major challenges is to design advanced storage systems to meet the demanding requirements for high performance, high capacity, and strong reliability.

Traditional storage systems rely on file servers to copy data between clients and storage devices. File servers, actually generic computers themselves, are not efficient in moving data between clients and storage nodes. The reason is that the data path involved includes client network interface, system memory, disk controllers to storage devices, and then along the same path back to the client. Inevitably memory copying and protocol processing have introduced many delays and overhead.

As a result, file servers in storage systems have emerged as a major barrier to achieve a better scalability. The performance of storage systems has been actively studied, including research areas on disk arrays [Anderson 1995b; Gibson 1996; Hitz 1994; Lee 1996] and distributed file systems [Anderson 1995b; Gibson 1996; Hartman 1993; Hitz 1994; Sandberg 1985; Satyanarayanan 1989; Thekkath 1997].

Over the years the market has witnessed many technological innovations ranging from faster peripheral channel, to dedicated storage area networks (SAN), and finally to aggressively specialized storage systems with custom hardware, operating systems and file

systems. Unavoidably these highly specialized storage systems come with high prices, which, more often than not, makes many organizations fumble for more budget on storage systems.

According to [Alvarez 2000], storage takes up 30-50% of total system cost before even paying for the recurring storage management. It is estimated that recurring personnel costs for storage management now dominate one-time capital costs over the equipment's useful lifetime [Gibson 2000]. The storage cost of per MB (megabytes) decreases constantly; however, the management cost keeps growing because of the increasing shortage of skilled storage system administrators in current tight IT labor market [Gibson 2000].

## **1.2 Motivations**

### ***1.2.1 Evolving storage architectures***

The architecture of storage systems has evolved into three generations. The first generation is called Direct-Attached Storage (DAS) [Duplessie 2001]. In DAS, storage servers are just generic workstations running a distributed file system like NFS, connecting to a SCSI RAID controller. SCSI's limited bandwidth and scalability quickly become a bottleneck when there are a large amount of clients.

The second generation Storage Area Networks (SAN) [Gibson 2000] has emerged to help solve the limitations imposed mainly by SCSI channels. With a dedicated storage network, Fibre Channel [Mearian 2001] can provide higher bandwidth (1Gbps) and higher capacity (256 devices per Fibre Channel – Arbitrated Loop, or more if using switch fabrics). There is a new technology called IP Storage (iSCSI) [Chudnow 2002; Satran 2000], which adapts SCSI protocol onto TCP/IP suite so that storage devices can run on

commodity Ethernet networks. At the same time, similar technologies have been proposed to expand Fibre Channel onto TCP/IP, namely iFCP and FCIP [Mearian 2001].

The third generation architecture is called Network Attached Storage (NAS) [Gibson 2000]. With SAN, a file server is still needed to manage the block space on an SAN rack like in the DAS. NAS devices are simply plugged into the network and are extremely easy to manage, therefore often referred to as storage appliances. NAS should not be considered as a competing technology to SAN, because internally very likely SAN technology is still used. Usually a dedicated file server is embedded inside an NAS storage rack.

The DAS structure is primarily used in small sever environment. The SAN structure is mostly used in medium to large enterprise products and the NAS structure is mainly used for department-level servers. With the evolution from DAS, to SAN and NAS, the fundamental architecture in storage systems has not changed. Essentially it is still a single server for both distributed file system and local file system, probably for redundancy management as well.

The author believes that this single-server architecture is preventing us from realizing the full potential from storage devices connected by the dedicated high-speed storage networks. The solution is to cluster multiple servers, each connected with a large amount of storage space, so that much higher aggregated bandwidth, performance, and capacity can potentially be achieved. The research in this dissertation study proposes the CoStore cluster architecture to construct storage systems using network attached storage devices and to achieve high performance, high reliability, high scalability and high capacity.

### ***1.2.2 Data is the most valuable asset***

The value of data has risen dramatically over the last few years to a point where, for many enterprises, it is the most valuable corporate asset [Webster 2001]. To maintain the availability, integrity and disaster recoverability of data, storage is now the most important resource in information infrastructure. This dissertation study also investigates remote replication using CoStore clusters to achieve high reliability and high availability storage systems and to reinforce preparedness for disaster recovery.

### ***1.2.3 Storage space utilization irony***

Besides the need for higher performance and better reliability, the other problem facing most infrastructure management is the demand for more storage capacity. With the Internet growing in size and reaching into every aspect of the society, information explosion has generated billions of bits every millisecond. Examples include databases of astronomical, geographical and medical images, data centers for e-Commerce, and multimedia content for news or entertainment. Even personal email has become attachment rich with pictures, audio or video clips. Given the rapid growth rate, many organizations are under continuous pressure to expand their storage systems as demands for capacities grow when the data sets swell relentlessly.

This dissertation study is in part motivated by the ironical fact that in some organizations there are growing demands for more storage space while enormous amount of disk space is idling and unused in their infrastructure. The disparity of storage space utilization ratios is expected to deteriorate further over time. This dissertation study assesses the feasibility of deploying CoStore architecture to construct reliable storage

service utilizing idle disk space on workstation clusters on existing computing infrastructure.

### 1.3 Problem Abstraction

The problem in this dissertation study can be generalized as follows. Given a group of network attached storage (NAS) devices, we are to construct a storage cluster without external file system managers. Such a storage cluster will provide a unified file namespace with a unique root in the file system tree. In CoStore both the distributed file system responsibilities and local file system management are evenly distributed among all cluster members.

Such NAS devices have many different forms. However, they all share the following attributes: i) they all contain block devices; ii) they all have Ethernet connections (Fast Ethernet or Gigabit Ethernet); iii) they are intelligent enough to manage local block devices themselves and provide a *file* interface on the network, in addition to the *block* interface available from traditionally dumb disk drives.

With the exponential growth in ASIC chip technology, disk drive controllers are potentially very powerful and intelligent. Ideally CoStore can use the imminent network attached smart disks [Gibson 1996] or Active Disks [Riedel 2001]. Though technically possible, Active Disks have not appeared on the market as of this writing because there is little economic incentive for disk drive manufacturers.

For practical reasons, in this study we will use generic PCs, running modern operating systems and connected with one or more disk drives. If the CoStore architecture turns out to be very successful, we can use advanced servers each connected with SAN rack devices.

## **1.4 Dissertation Overview**

This dissertation consists of six chapters. Chapter 1 presents an introduction and motivations to this study and provides an abstraction to the problem being solved.

Chapter 2 presents a background review on storage systems. Chapter 3 first introduces technology trends; then presents the CoStore cluster architecture using contemporary approaches and describes the construction of a prototype CoStore cluster using commercial-off-the-shelf (COTS) components. The performance of the prototype has been measured and compared with other commonly available distributed file systems.

Chapter 4 assesses the feasibility of deploying the CoStore architecture using workstation clusters on existing desktop environment. Chapter 5 investigates the construction of highly reliable and highly available storage systems using CoStore clusters.

Chapter 6 first discusses related work and then concludes this dissertation study with conclusions and contributions made by this study. Chapter 6 provides areas for future work.

## Chapter 2

### Background

#### 2.1 Storage System Components

##### 2.1.1 Storage devices

##### 2.1.1.1 Disk drives

Disk drives are nonvolatile storage devices that record data on magnetic media [Ruemmler 1994]. Each disk device consists of one or more stacked platters attached to a rotating spindle (Figure 2-1). Disks magnetically store data on recording surfaces located on both sides of each platter. Concentric circles called tracks divide each platter surface. Sectors divide tracks into the smallest unit that can be read or written, typically a size of 512 bytes.

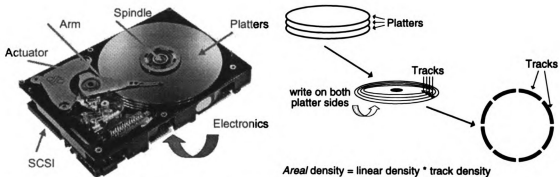


Figure 2-1 (L) Structures inside disk; Image courtesy of Seagate Technology, Inc. (R) mechanisms in a magnetic disk drive

Each recording surface has one head that reads and writes sector data. Heads attach to the ends of actuator arms. These arms pivot the heads, in unison, between tracks. Seek time is the duration required to position a head to the desired track. Rotational latency is the time

spent after a head seek but before the target sector rotates under the head. Together, seek time and rotational latency comprise total positioning time of a request.

Modern devices possess speed-matching buffers. To coordinate between channel availability and media latencies, devices temporarily store data in speed-matching buffers. Some advanced devices use buffer memory for caching. Caches may act as read-ahead buffers by pre-fetching likely to be read data. Caches may also write buffer requests by transferring data into buffers and then releasing the channel. Devices write cached data to media at a later time. Advanced configurations function as both read and write caches. Read and write caches typically employ variants of least recently used (LRU) replacement policies.

The latency of a disk access can therefore be broken down into three main elements: seek, rotational and transfer latencies. Seek latency refers to the time it takes to position the read/write head over the proper track. This involves a mechanical transitional movement that may require acceleration in the beginning and deceleration and repositioning in the end. As a result, although seek times have been improving, they have not kept up with the rates of improvement of silicon processors. While processing rates have improved by more than an order of magnitude, average seek times have shrunk to only half of their values of a decade ago [Gibson 1992].

The second element, rotational latency, refers to the time it takes to wait for the sector to rotate under the read/write head. This is determined by the rotational speed of the disk. Rotational speeds have improved slowly over the past decade, improving at an average annualized rate of 13%. Higher rotational speeds reduce rotational latencies and improve transfer rates. Unfortunately, they are hard to improve because of electrical and

manufacturing constraints. The third element is transfer time, which is the time for the target sectors to pass under the read/write head. Disk transfer times are determined by the rotational speed and storage density (in bytes/square inch). Disk areal densities continue to increase at 50 to 55% per year, leading to dramatic increases in sustained transfer rates, averaged at 40% per year [Grochowski 1996].

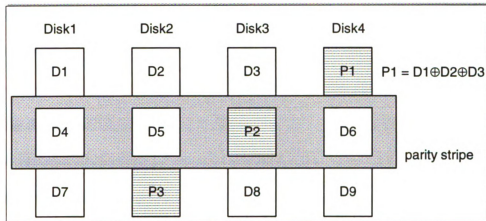


Figure 2-2 The layout of data and parity blocks in a RAID level 5

### 2.1.1.2 Disk arrays

In the late 1980s, in order to bridge the growing access gap between the storage subsystem and the processors, redundant arrays of inexpensive disks (RAID) were proposed [Chen 1994; Patterson 1988] to replace expensive large disk systems. (Inexpensive was later replaced with Independent.) RAID arrays provide the illusion of a single logical device with high small-request parallelism and large-request bandwidth. By storing a partially redundant copy of the data as parity on one of the disks, RAIDs improved reliability in arrays with a large number of disks.

A piece of data is spread over  $N-1$  data blocks and one parity block, which is the XOR of the corresponding bits of the data blocks. In other words the data and the parity are striped over different disks in the array. High performance is achieved by the parallelism of multiple disks and fault tolerance is provided by the extra check data parity – the contents of a failed data disk can be reconstructed by taking the exclusive-OR of the remaining data blocks and the parity block.

Several different schemes to organize disk arrays were defined by Paterson et al., known as RAID level 0 through RAID level 5. RAID 0 is data striping only without any check information; therefore it has the highest parallelism but no redundancy. The RAID 1 duplicates all disks or mirroring; hence fault tolerance is excellent and read performance is doubled. However, mirroring is the most expensive of all levels and the write performance is not improved. RAID 2 applies Hamming code error-correction like memory-style ECC and RAID 3 uses bit-interleaved parity. Both RAID 2 and 3 are rarely used in common disk arrays.

RAID 4 and 5 are both block-interleaved parity. In RAID 5, parity is uniformly distributed across all disks (Figure 2-2) whilst in RAID 4 parity is stored on one dedicated disk. Because the parity disk is a potential bottleneck, normally RAID 4 is seldom used.

There are also a few other levels proposed. RAID 6 is essentially an extension to RAID 5, which utilizes double correcting redundancy (Reed-Solomon) codes to provide additional fault tolerance. RAID 6 has a much more complex controller design and the write performance is very poor.

#### 2.1.1.2.1 RAID level 4 and level 5

The parity block is rotated around the devices in an array. Each write to any of the disks needs to update the parity block. Rotating the parity block balances the parity write traffic across the devices.

RAID level 5 employs a combination of striping and parity checking. The use of parity checking provides redundancy without the 100% capacity overhead of mirroring. In RAID level 5, a redundancy code is computed across a set of data blocks and stored on another device in the group. This allows the system to tolerate any single self-identifying device failure by recovering data from the failed device using the other data blocks in the group and the redundant code. The block of parity that protects a set of data units is called a parity unit. A set of data units and their corresponding parity unit is called a parity stripe.

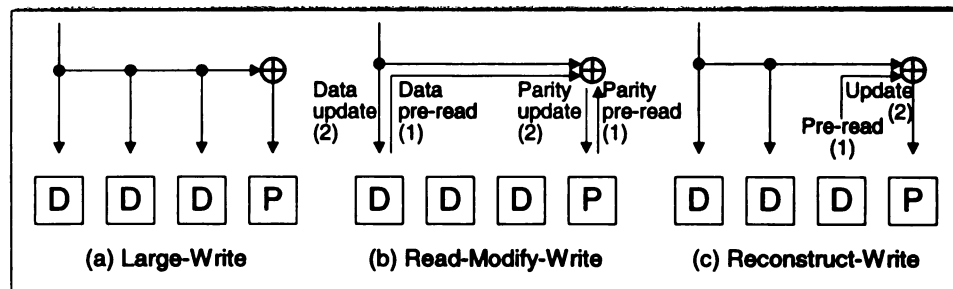
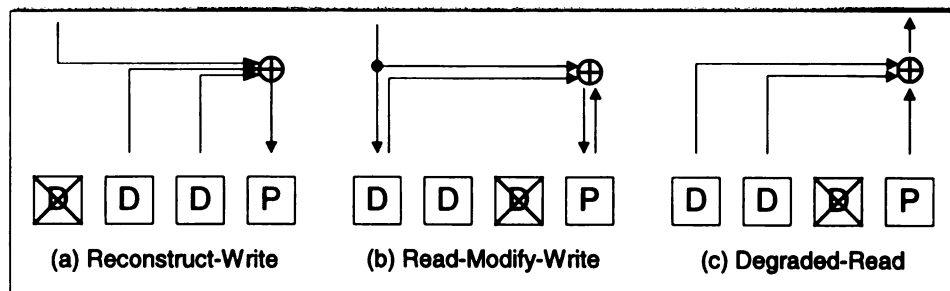


Figure 2-3 RAID level 5 host writes and reads in normal condition

Figure 2-3 shows RAID level 5 host writes and reads in normal condition. Write operations in fault-free mode are handled in one of three ways, depending on the number of units being updated. In all cases, the update mechanisms are designed to guarantee the property that after the write completes, the parity unit holds the cumulative XOR over the corresponding data units. In the case of a large write (Figure 2-3(a)), since all the data units

in the stripe are being updated, parity can be computed by the host as the XOR of the data units and the data and parity blocks can be written in parallel.

If less than half of the data units in a stripe are being updated, the read-modify-write protocol is used (Figure 2-3(b)). In this case, the prior contents of the data units being updated are read and XORed with the new data about to be written. This produces a map of the bit positions that need to be toggled in the parity unit. These changes are applied to the parity unit by reading its old contents, XORing it with the previously generated map, and writing the result back to the parity unit. Reconstruct-writes (Figure 2-3(c)) are invoked when the number of data units is more than half of the number of data units in a parity stripe. In this case, the data units not being updated are read, and XORed with the new data to compute the new parity. Then, the new data units and the parity unit are written.



**Figure 2-4 RAID level 5 host writes and reads in the presence of a fault**

Figure 2-4 shows RAID level 5 host writes and reads in the presence of a fault. If a device has failed, the degraded-mode write protocols shown in Figure 2-4(a) and Figure 2-4(b) are used. Data on the failed device is recomputed by reading the entire stripe and XORing the blocks together as shown in Figure 2-4(c). In degraded mode, all operational devices are accessed whenever any device is read or written.

The more balanced RAID 5 is by far the most popular RAID organization of all levels. However, Network Appliance has adopted RAID 4 because their novel WAFL (Write Anywhere File Layout) design can overcome most of the bottleneck effect [Hitz 1994]. Compared with RAID 5, RAID 4 is much easier to implement and disks can be added incrementally into the array with little reorganizing work. In this proposal only RAID level 4 and 0+1 are considered.

One major drawback of RAID 4 and 5 is that small writes have very high overhead, called small write penalty. Techniques have been proposed to improve the performance of small updates, including parity-logging [Stodolsky 1993], floating-parity scheme [Menon 1989] and log-structured file system design (LFS) [Rosenblum 1992.], among others. AFRAID (A Frequently Redundant Array of Independent Disks) was proposed [Savage 1996] to eliminate the small-update penalty to RAID 5. The concept is that considerably better performance can be achieved by consciously sacrificing a small amount of data redundancy.

#### *2.1.1.2.2 RAID status*

RAID 1 is increasingly popular because the controllers are simple to implement and many PC motherboards of recent models are providing them embedded, along with the combination of RAID 0 and 1. RAID 0+1 stands for striping across mirrored disks and RAID 1+0 stands for the mirroring of two virtual disks, each is a set of striping disks. The performance characteristics of these two are equivalent. However, whenever possible RAID 0+1 is preferred because a single disk loss will not affect the whole array, while in RAID 1+0 a single disk failure will ruin the whole striping set in one virtual disk

[VERITAS 2000]. The striped-mirrors RAID 0+1 or 1+0 (sometimes also called RAID 10) has the reliability of mirroring RAID 1 and the performance of striping-only RAID 0. Because the cost per MB continues to drop rapidly, mirroring RAID 1 or its variation 0+1 has witnessed accelerating market acceptance as inexpensive RAID solution. Another possibility is to combine level 4 and 1 into RAID 4+1, similar to RAID 0+1.

Full-blown hardware RAID is an expensive solution, compared to software RAID, which is implemented by operating systems kernel without hardware support, mostly in a form of device driver. Software RAID is available from many popular PC operating systems: Linux, Solaris and Windows 2000 Server.

A volume manager is a subsystem for online disk storage management [Teigland 2001]. It adds an additional layer between physical peripherals and the I/O interface in the kernel to present a logical view of disks. A volume manager will often implement one or more levels of software RAID to improve performance or reliability.

### ***2.1.2 I/O interfaces***

I/O interfaces transport data between host processors and peripheral devices. Interfaces often become open industry standard. I/O interfaces can be categorized into two groups: parallel and serial interface. It is an industrial trend that these parallel interfaces will soon be serialized. Parallel interfaces include ATA/IDE and SCSI. Serial interface include USB, IEEE1394, SSA, Fibre Channel, and forthcoming serial ATA and serial SCSI.

### ***2.1.2.1 Parallel interfaces***

The most popular I/O interface is the Integrated Drive Electronics (IDE) [Tanenbaum 1999], which appeared with the BIOS (Basic Input Output System) on IBM PCs since mid 1980s. The more official name for IDE is ATA (AT Attachment). For lack of foresight, the initial BIOS only supports a maximum drive of capacity 512MB because in the BIOS there are 4 bits for the head, 6 bits for the sectors and 10 bits for the cylinders. Eventually IDE drives evolved into EIDE (Extended IDE), which also supports a second addressing scheme called LBA (Logical Block Address), which just numbers the sectors starting at 0 up until a maximum of  $2^{24}-1$ .

IDE and EIDE disks were originally only for Intel-based systems since the interface is an exact copy of the IBM PC bus. However, now a few other computer architectures (including SUN SPARC workstations) also use them recently. The latest EIDE supports Ultra ATA 100/133 with a burst data rate up to 100MB/133MB per second. The disadvantage of (E)IDE is that only limited aggregated bandwidth can be supported, with limited expandability, up to 4 channels and 8 devices. Therefore, EIDE is still mostly limited to PCs and entry-level workstations.

In storage system, the other most important I/O interface is SCSI (Small Computer System Interface) interface, which was standardized in 1986 by the T10 committee of the National Committee on Information Technology Standards (NCITS). NCITS, formerly known as X3, operates under the auspices of the American National Standards Institute (ANSI). Since then, increasingly faster versions have been standardized with the names Fast SCSI (10MHz), Ultra SCSI (20MHz), and Ultra2 SCSI (40MHz). Each of these has a wide (16bit) version as well. The current (sixth) generation, Ultra160 SCSI data rate reaches to

160Mbps. Future versions include Ultra320 and Ultra640 each doubling the preceding generation's data rates [Mason 2000]. Unlike other I/O interfaces, SCSI product generations are always backward and forward compatible, preserving the owner's investment and making it possible to connect legacy devices to newer system [Mason 2000]. Table 2-1 shows the all of the different SCSI transfer modes and feature sets, along with their key characteristics [Kozierok 2001].

Because SCSI disk have high transfer rates, traditionally they are the standard disks in most UNIX workstations from Sun, HP, SGI and other vendors. They are also the standard disks on Macintoshes and high-end Intel PCs, especially network servers.

**Table 2-1 SCSI parameters. DT means double transition clocking.**

Transfer Mode	Defining Standard	Bus Width (bits)	Bus Speed (MHz)	Throughput (MB/s)	Cabling
SCSI-1	SCSI-1	8	5	5	50-pin
Wide SCSI	SCSI-2	16	5	10	68-pin
Fast SCSI	SCSI-2	8	10	10	50-pin
Fast Wide SCSI	SCSI-2	16	10	20	68-pin
Ultra SCSI	SCSI-3 / SPI	8	20	20	50-pin
Wide Ultra SCSI	SCSI-3 / SPI	16	20	40	68-pin
Ultra2 SCSI	SCSI-3 / SPI-2	8	40	40	50-pin
Wide Ultra2 SCSI	SCSI-3 / SPI-2	16	40	80	68-pin
Ultra3 SCSI	SCSI-3 / SPI-3	16	40 (DT)	160	68-pin
Ultra160 SCSI	SCSI-3 / SPI-3	16	40 (DT)	160	68-pin
Ultra320 SCSI	SCSI-3 / SPI-4	16	80 (DT)	320	68-pin

On the high-end storage servers, SCSI disks were once exclusively used. However, SCSI has its own limitations. The main disadvantages are the limited number of devices on each channel and limited distance of SCSI cabling. SCSI is a parallel interface, with a wide cable connector, which is very unpleasant in an environment of high-density devices. To accommodate the emerging requirements of massive capacity, high-bandwidth and continuous availability, recently on the high-end storage systems, SCSI interface disk drives have been gradually replaced by serial storage interfaces such as Fibre Channel – Arbitrated Loop (FC-AL).

The parallel ATA will give way, over the next two or three years, to a new connectivity standard proposed by the Serial ATA Working Group [Norman 2002]. The use of gigabit technology will make Serial ATA viable for at least 10 more years, enabling the interface to better keep up with the data-transfer demands of advanced processor. Serial ATA will also eliminated parallel ATA's large, unwieldy ribbon cables.

#### ***2.1.2.2 Serial interfaces***

USB (Universal Serial Bus) interface appeared in mid 1990s and is being widely implemented on personal computers [Anderson 1997]. USB is mainly designed for low-speed devices such as keyboards, mice, digital cameras, scanner, printers and so on. There are disk drives with USB interface on the market and most of them are external to take advantage of the convenience of mobility. The USB 1.1 bandwidth is 1.5MB/s (or 12Mbps). The latest USB 2.0 can support bandwidth as high as 480Mbps and is backward compatible with USB 1.1.

The IEEE 1394 multimedia connection enables simple, low-cost, high-bandwidth real-time data interfacing between computers, peripherals, and consumer electronics products such as camcorders, VCRs, printers, PCs, TVs, and digital cameras. The 1394 digital link standard was conceived in 1986 by technologists at Apple Computer, who chose the trademark 'FireWire', in reference to its speeds of operation. The first specification for this link was completed in 1987. It was adopted in 1995 as the IEEE 1394 standard. Some high-end disk drives are equipped with 1394 port for the purpose of faster transfer rate of data, which is a necessity in professional digital video processing, and other multimedia applications. Currently IEEE 1394 supports bandwidth of 400Mbps and future versions will go as high as 800Mbps and 1.6Gbps.

Serial Storage Architecture (SSA) SSA [SSA 1995], a serial-connection technology proposed by IBM, abandons the wide parallel cables used by SCSI in favor of a simple four-wire serial connection. Unlike SCSI, which uses a shared bus to connect the device, SSA uses a pair of point-to-point links to connect two devices together via a port on each device [Du 1996]. We investigated the impact of SSA's spatial reuse property on the performance of RAID storage system in [Jayaram 1998].

By year 2000, SSA has almost lost the war to the competing FC-AL technology, in part for the lack of wide industrial support to an IBM proprietary standard and for its technical limitations. However, some of its technology contributions will prevail as is attested by the fact that Seagate, Adaptec and IBM have started development of a new technology, called FC-EL (Fibre Channel – Enhanced Loop), which will combine both the advantages of SSA and FC-AL.

**Fibre Channel – Arbitrated Loop (FC-AL)** Fibre Channel (FC) is an emerging ANSI serial interface that supports channel and network operations [ANSI 1993]. Fibre Channel consists of five functional levels. Modular design allows independent implementation of each level.

Fibre Channel defines all computers, switching elements, and storage devices as nodes. Each node has one or more FC ports called network ports (N Ports). Each port possesses a transmitter and a receiver to interface media. Fibre Channel supports three topologies: point-to-point, arbitrated loop (FC-AL), and fabric. Point-to-point configurations are the simplest. This topology connects node pairs or acts as backbones for other network types. Arbitrated loops connect several ports via shared media. Fabrics connect ports to switched environments.

Like shared buses, only one arbitrated loop node transmits at any given time. As many as 126 nodes connect together to form a ring; the transmitter of one node connects to the receiver of another. A transmitting node must first arbitrate for the loop. After acquiring the loop, the transmitting node either sends messages to other nodes or broadcasts to several nodes. After transmission, the node releases the loop. Loop ports (L Ports) are nodes that support the arbitrated loop topology. The bandwidth per Fibre Channel loop is 1Gbps or 100MBps.

Performance and cost requirements dictate network topology. Many nodes may attach to inexpensive, arbitrated loops in order to share loop bandwidth and connectivity. Nodes can also attach to fabric ports and exploit the network bandwidth. Loops connect to fabric ports for high connectivity but relatively low costs.

### ***2.1.2.3 Merger of I/O Channels and Ethernet***

Recent interface trends combine channel and network technologies into single interfaces capable of supporting multiple protocols [Sachs 1994]. Interface merging tends to produce slightly more complicated designs, but these interfaces generally inherit advantages of both channels and networks. Combining these traditionally independent subsystems enables vendors to produce single products with multiple uses. Vendors providing combined products benefit from larger markets.

A network attached peripheral (NAP), according to Van Meter, is “a network computer peripheral that communicates via a network rather than a traditional I/O bus, such as SCST” [Van Meter 1996]. Van Meter presents characteristics that distinguish NAPs from bus-based devices. Such characteristics include interconnect distance, ownership, and an ability to handle general network traffic. Printers and computer terminals often fit this definition. Other peripherals like scanners and storage devices may also be designed as NAPs. Merging channels and networks provides new functionality to devices. Network attached storage (NAS) devices connect directly to networks. Multiple computers can share NAS devices.

One particular example is the new technology called IP Storage or iSCSI (Internet SCSI), which maps SCSI protocol onto TCP/IP protocol [Satran 2000]. iSCSI will enable SCSI storage controllers, disk subsystems and tape libraries to attach directly to IP networks. According to Nick Allen, iSCSI promises to let users operate SAN, NAS, LAN and WAN as a single, integrated network [Chudnow 2002]. While iSCSI can run on standard Gigabit Ethernet switches, the overhead involved in processing TCP and the iSCSI protocol can quickly overwhelm the CPU in servers with significant storage network

traffic. Therefore standard Ethernet adaptors with special host bus adapters (HBA) called TCP off-load engines (TOE). There is significant investment in the imminent 10GB Ethernet technology, which adds more fuel into iSCSI's development.

The future of Fibre Channel based SAN is overcast with questions due to the limited bandwidth (2GB) in its next generation while iSCSI is soon to take advantage of the 10GB Ethernet. However, the Fibre Channel camp is not resting. Several projects similar to iSCSI are in the works, including iFCP (Internet Fibre Channel Protocol) and FCIP (Fibre Channel over IP) [Mearian 2001]. FCIP enables the transmission of Fibre Channel information by tunneling data between SANs over IP networks. A hybrid technology, iFCP is a version of FCIP that moves Fibre Channel data over networks using the iSCSI protocols. All the three new protocols iSCSI, iFCP and FCIP are interfaces to SAN based storage systems [Paulson 2002].

The demand to move data with greater speed has been restricted by a number of bottlenecks and arguably the greatest restriction to data flow is the nature of the I/O architecture [Williams 2001]. The InfiniBand technology addresses the shortfall of current I/O buses and will provide speed range from 500Mbps to 6Gbps per link. The implications for storage networking are enormous [Williams 2001]. A fabric of InfiniBand switches and links is used to provide connections between servers, remote storage and networking devices. The key to InfiniBand is switching technology that links high-channel adaptors (HCA) to target channel adaptors (TCA). The TCAs support the storage and peripheral device I/O. The switch operates between the HCA and TCA to manage and direct data packets.

## 2.2 Local File Systems

This section presents operating system background with UNIX as example. Reasons for choosing UNIX include: (1) a large install base and variations of UNIX run on most platforms architectures; (2) a large amount of UNIX design literature and research exist; (3) numerous non-UNIX operating system inherit many design principles from UNIX; (4) the prototype of this proposal is targeted at direct integration into existing UNIX dominant infrastructure.

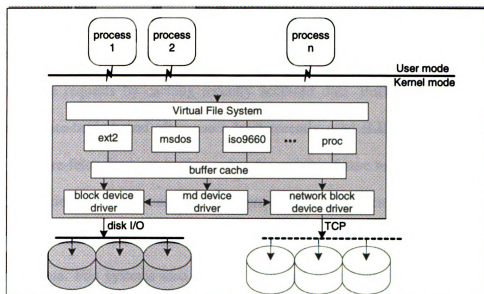


Figure 2-5 File system layers in UNIX

### 2.2.1 File system layers in operating systems

UNIX functionally organizes storage and network subsystems into layers [Bach 1986]. Figure 2-5 illustrates these layers. Each layer views storage and communication through different degrees of abstractions. The top layer of Figure 2-5 is user space. Programs operating at this level include command shells and system utilities; applications run on top

of these programs. All user level programs interact with the operating system, or kernel, through system calls.

The file system layer lies beneath the system call layer. Modern UNIX implementations include installable file system interfaces. Many UNIX implementations incorporate the Virtual File System (VFS) interface. VFS, developed by Sun Microsystems, provides a common interface to file systems. VFS divides file system functionality into file system and individual file operations [Kleiman 1986; Sandberg 1985].

File systems often use buffer cache services. The buffer cache layer consists of system memory buffers and routines that operate on these buffers. Buffer caches provide caching, pre-fetching, and temporary memory for non-aligned transfers. These caches reduce device read and write requests by caching recently accessed data. Buffer caches either write-through or write-behind, and often use a least recently used (LRU) replacement policy. File systems pre-fetch data into buffer caches for possible future reference. Pre-fetches are often continuations of current requests. Such pre-fetching only increases data transfer times of request durations, since devices already perform initial seeks and rotations for the non-pre-fetched data. File systems also use buffer caches to temporarily store data for non-aligned transfers.

The device driver layer is the lowest software level. Device drivers, or drivers, are interfaces between the kernel and hardware. Drivers hide hardware device specifics. Several levels of device drivers comprise the device driver layer. High-level device drivers are more abstract than low-level drivers. One good example is that software RAID is implemented as high-level device driver, which interacts with low-level block device drivers. Low-level drivers are most hardware specific [Rubini 2001]. I/O requests arrive at

device drivers from higher in the kernel. Drivers use input requests to construct requests suitable for lower drivers or hardware. Drivers place newly formed requests on the driver queue. These queues are usually first-in, first-out (FIFO), but can be priority-based to enable scheduling. Drivers pass queued requests down to lower device drivers or the hardware level.

### ***2.2.2 Virtual File System (VFS)***

The virtual file system (VFS) is an interface that supports various file system types within a kernel. Several UNIX implementations incorporate VFS; however, interfaces differ from one platform to another. There have been many file systems and they all share common structures. VFS is an object-oriented interface. This interface defines virtual VFS and vnode operations. Each installed file system provides the kernel with functions associated with VFS and vnode operations.

VFS operations include functions that operate on file systems by mounting, unmounting, and reading status, respectively. Vnode operations manipulate individual files. A vnode is the VFS virtual equivalent of an inode. VFS creates and passes vnodes to file system vnode operations. Vnode operations include opening, closing, creating, removing, reading, writing, and renaming files. VFS defines many other vnode operations, yet file system implementations need only support a subset of these routines.

### ***2.2.3 File system structures***

File systems manage user and system data on secondary storage. Applications often address data at the byte level, though storage devices are typically block addressable. File systems

perform translations between byte and block level addressing schemes. The terms metadata and real data classify file system structure data and user data, respectively. In other words, real data is data that users store in files. File systems create metadata to store layout information; metadata is not directly visible to users.

Files abstractly hide details concerning storage management from users. UNIX recognizes several file types. File systems and application programs handle each file type differently. Users store and retrieve data from regular files as contiguous, randomly accessible segments of bytes. Users are responsible for organizing data stored in regular files.

Directories are file abstractions that organize collections of files. In most modern file systems, directories nest within one another, thereby forming a tree structure with empty directories and regular files at the leaves. File names are unique to directories but not to file systems. Applications identify files by complete pathnames. A file name and the names of all encompassing directories comprise a pathname. The entire collection of file names composes the file system name-space.

Computers may simultaneously access files from multiple file systems, including local and network file systems. To provide a transparent name-space, file systems mount other file systems at mount points. Users transparently traverse from one file system, across a mount point, to another file system. File system root directories attach to mount points.

Internal fragmentation occurs when file systems do not fill entire blocks. Traditionally, file systems limited internal fragmentation by using small file system block sizes. Today, conservation of storage is less important, so optimizations focus on improving file transfer rates. External fragmentation is due to non-contiguous storage of data blocks. Since I/O

requests incur substantial device overheads, external fragmentation strongly influences transfer rates. Increases to file system block sizes tend to reduce external fragmentation, however large block sizes increase internal fragmentation.

File systems transfer blocks of data between system memory and storage devices. File system block sizes are multiples of device block sizes.

### 2.2.3.1 Ext2 file system

In this section the Second Extended File System (Ext2) [Card 1986], the *de facto* standard file system on Linux, is used to explain meta-data structures for generic files systems. In Ext2 file system block sizes are 1KB, 2KB and 4KB.

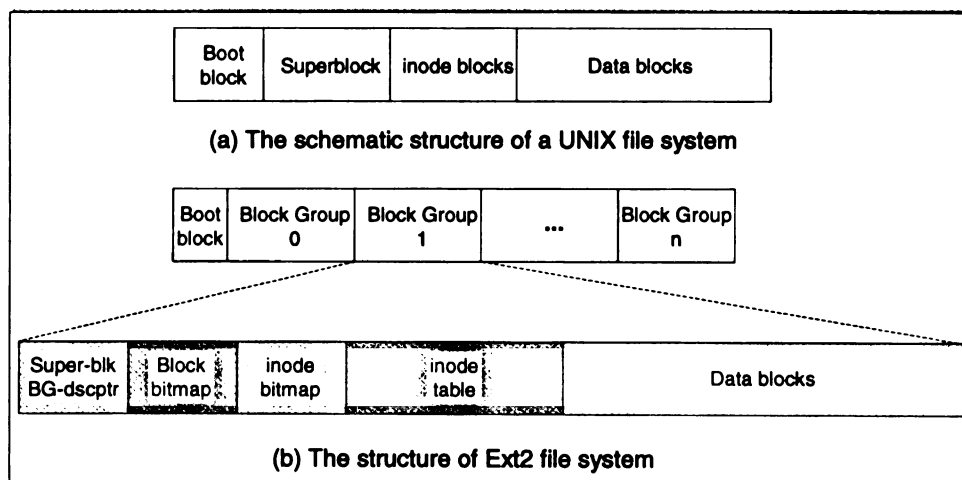


Figure 2-6 (a) The schematic structure of a file system; (b) the structure of Ext2 file system

The basic structure is the same for all the different UNIX file systems (Figure 2-6(a)). Each file system starts with a *boot block*. This block is reserved for the code required to boot the operating system. All the information essential for managing the file system is held in the *superblock*. *superblocks* maintain information concerning the amount of free

space left on the file system, the device on which the file system is mounted, and file system access privileges. *superblocks* also maintain pointers to locate file system root directories. *superblock* is followed by a number of *inode blocks* containing the inode structures for the file system. The remaining blocks provide the space for data. These *data blocks* thus contain ordinary files along with the directory entries and the indirect blocks.

In Unix file systems each file is represented by one *inode*, which stores all the information about that file: permissions, type, size, and data block locations etc. The block devices are divided into metadata and data blocks. The metadata blocks store *superblock*, *block-bitmap* and *inode-bitmap*, and *inode-table*. For efficiency reason, blocks are first split into block groups with each block group has its own metadata [McKusick 1984]. Figure 2-6(b) shows a typical Ext2 file system structure.

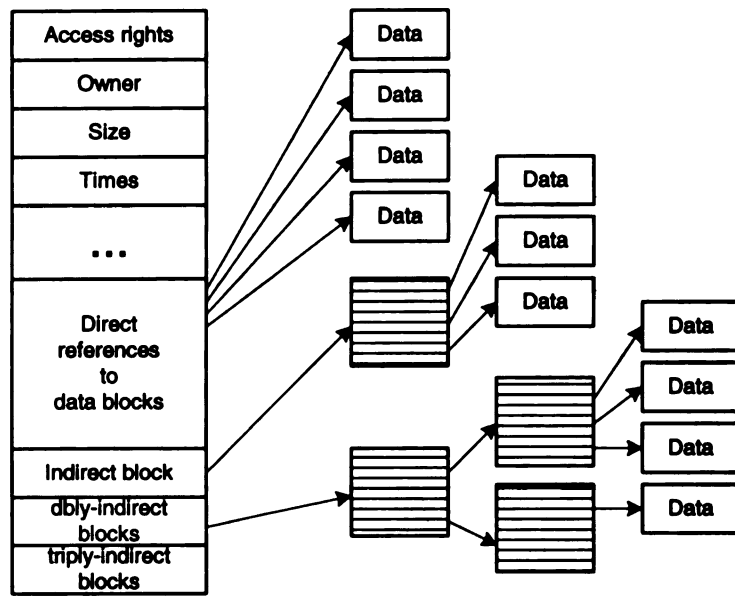
The design of the Ext2 file system is very much influenced by BSD's Fast File System [McKusick 1984]. A partition is divided into a number of block groups, with each block group holding a copy of the *superblock*, and inode and data blocks, as shown in Figure 2-6(b). The block groups are employed with aim of keeping (i). data blocks close to their inodes; and (ii) file inodes close to their directory inode. Block groups reduce disk seek time to a minimum. Every block contains the superblock along with information on all the block groups, allowing the file system to be restored in an emergency. Figure 2-7 show the *superblock* structure in Ext2 file system.

	0	1	2	3	4	5	6	7				
0	Number of inodes				Number of blocks							
8	Number of reserved blocks				Number of free blocks							
16	Number of free inodes				First data block							
24	Block size				Fragment size							
32	Blocks per group				Fragments per group							
40	Inodes per group				Time of mounting							
48	Time of last write				Status		Max. mount counter					
56	Ext2 signature		Status		Error behaviour							
64	Time of last test				Max. test interval							
72	Operating system				File system revision							
80	RESUID		RESGID									

Figure 2-7 The superblock in Ext2 file system

Additionally, in each block group Ext2 file systems maintain free lists of unallocated data blocks inode by means of bitmap tables: *block-bitmap* and *inode-bitmap* as shown in Figure 2-6(b). File systems set bits to signify blocks or inode that are allocated to files.

Figure 2-8(a) illustrates the structure of an inode and metadata pointer tree for a traditional UNIX file system. The inode contains header information and several pointers. The first ten pointers are direct pointers that address data blocks. The first indirect pointer addresses a data block filled with direct pointers. The second indirect pointer addresses a data block filled with indirect pointers. The third indirect pointer addresses a block of double indirect pointers, taking a total of three indirections to reach real data.



a) The structure of a UNIX inode

	0	1	2	3	4	5	6	7
0	Type/permissions		User (UID)		File size			
8	Access time				Time of creation			
16	Time of modification				Time of deletion			
24	Group (GID)		Link counter		No. of blocks			
32	File attributes				Reserved (OS-dependent)			
40	12 direct blocks							
88	One-stage indirect block				Two-stage indirect block			
96	Three-stage indirect block				File version			
104	File ACL				Directory ACL			
112	Fragment address				Reserved (OS-dependent)			
120								

b) The inode in Ext2 file system.

Figure 2-8 (a) The structure of a Unix inode; (b) the inode in Ext2 file system

Figure 2-8(b) shows the structure of an inode in Ext2. inodes also store file type (file or directory), ownership information, access permissions, access times, and file sizes. Directory files contain file names and unique inode index numbers (Figure 2-9), called inode numbers. File systems use inode numbers to locate inodes stored on disk. The separation of inode numbers from file names allows file systems to support link files. Link files transparently reference other files. Links provide multiple file names for single files.

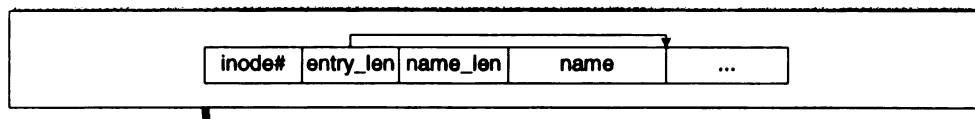


Figure 2-9 A directory entry in the Ext2 file system

Ext2 is adopted with minor modifications as local file system in the prototype implementation of this proposal. These modifications enable multiple Ext2 file systems to construct a *virtual* file system spanning across all storage nodes.

#### 2.2.4 File system management

File system operations include data operations and meta-data operations. Data operations are performed on actual user data, reading or writing data from/to files. Meta-data operations modify the file system structure, creating, deleting or renaming files or directories.

When unfortunate situations occur, such as an unexpected power failure or system lock-up, the system does not have the opportunity to cleanly unmount filesystems. When the system is rebooted and fsck starts its scan and detects that these filesystems were not cleanly unmounted. It is very likely that the meta-data is in inconsistency status. To fix this

situation, fsck will begin an exhaustive scan and sanity check on the meta-data, correcting any errors that it finds along the way. Once fsck is complete, the filesystem is ready for use. Although some recently modified data may have been lost due to the unexpected system crash, since the meta-data is now consistent, the filesystem is ready to be mounted and used.

The problem with fsck is that complete consistency check on all meta-data is a time-consuming task in itself and normally takes at least several minutes to finish. The bigger the filesystem, the longer this exhaustive scan will take.

Therefore during a meta-data operation, the system must ensure that data are written to disk in such a way that the file system can be recovered to a consistent state if a system crash occurs. Traditionally this requirement has been met by synchronously writing each block of meta-data [Seltzer 2000]. Unfortunately synchronous writes can significantly impair file system's performance on meta-data operations.

There are two commonly used approaches for improving the performance of meta-data operations and recovery: journaling and Soft Updates. Journaling (or logging) file systems use an auxiliary log to record meta-data operations and Soft Updates uses ordered writes to ensure meta-data consistency. Most operating systems have adopted journaling file systems. There have been quite a few journaling file systems: JFS, ReiserFS, XFS, Ext3 and GFS.

#### ***2.2.4.1 Journaling***

Journaling file systems attack the meta-data update problem by maintaining an auxiliary log that records all meta-data operations and ensuring that the log and data buffers are

synchronized in such a way to guarantee recoverability [Seltzer 2000]. The system enforces write-ahead logging [Gray 1993], which ensures that the log is written to disk before any pages containing data modified by the corresponding operations. If the system crashes, the log system replays the log to bring the file system to a consistent state. Journaling systems always perform additional I/O to maintain ordering information (i.e., they write the log). However, these additional I/Os can be efficient, because they are sequential compared with random writes with long track-seeking. When the same piece of meta-data is updated frequently, journaling systems consolidate these log writes in exchange for avoiding multiple meta-data writes.

When the filesystem is mounted, the file system checks to see whether the filesystem is consistent. If for some reason it is not, then the meta-data needs to be fixed, but instead of performing an exhaustive meta-data scan in fsck, it takes a look at the journal. Since the journal contains a chronological log of all recent meta-data changes, it simply inspects those portions of the meta-data that have been recently modified. Thus, it is able to bring the filesystem back to a consistent state in a matter of seconds. And unlike the more traditional approach that fsck takes, this journal replaying process does not take longer on larger filesystems.

The key design issues in a journaling file system are where to store the log; how to manage the log and when to reclaim space and when to do checkpointing; interface between the log and the main file system; and how to recover from the log.

#### ***2.2.4.2 Soft Updates***

Soft Updates tries to solve the meta-data update problem by guaranteeing that blocks are written to disk in their required order without using synchronous disk I/Os [Seltzer 2000]. In general, a Soft Updates system must maintain dependency information, or detailed information about the relationship between cached pieces of data. For example, when a file is created, the system must ensure that the new inode reaches disk before the directory that references it does. In order to delay writes, Soft Updates must maintain information that indicates that the directory data block is dependent upon the new inode and therefore, the directory data block cannot be written to disk until after the inode has been written to disk. In practice, this dependency information is maintained on a per-pointer basis instead of a per-block basis in order to reduce the number of cyclic dependencies.

#### ***2.2.4.3 Ext3: journaling Ext2***

Ext3 has been designed by Stephen Tweedie to be extremely easy to deploy [Robbins 2001; Tweedie 2000]. Ext3 is built on the solid Ext2 filesystem code and it inherits a great fsck tool. And Ext3's journaling capabilities have been specially designed to ensure the integrity of both metadata and data [Robbins 2001].

##### ***2.2.4.3.1 Backwards compatibility***

Ext2 and Ext3's on-disk format is identical, which means that a cleanly unmounted Ext3 filesystem can be remounted as an Ext2 filesystem- It is possible to perform in-place Ext2 to Ext3 filesystem upgrades. By upgrading a few key system utilities, installing a modern 2.4 kernel and typing in a single tune2fs command per filesystem, users can convert

existing Ext2 file systems into journaling Ext3 systems. The transition is safe, reversible, and incredibly easy.

In addition to being Ext2-compatible, Ext3 inherits other benefits by sharing Ext2's metadata format. Ext3 users gain access to a rock-solid fsck tool. If you do end up getting corrupted metadata, either from a flaky kernel, bad hard drive, or something else, you can still use fsck to fix inconsistency in your Ext3 file systems.

Ext3's journal is stored in an inode, or basically a file. By storing the journal in an inode, Ext3 is able to add the needed journal to the filesystem without requiring incompatible extensions to the Ext2 metadata. This is one of the key ways that an Ext3 filesystem maintains backwards compatibility with Ext2 metadata.

#### ***2.2.4.3.2 The Ext3 approach***

Ext3 handles journaling very differently than other journaling filesystems. With ReiserFS, XFS, and JFS, the filesystem take extra special care to journal metadata, but makes no provisions for journaling data. However, unexpected reboots and system lock-ups can result in significant corruption of recently modified data. Ext3 uses a couple of innovative solutions to avoid these problems, which we'll look at in a bit.

In Ext3, the journaling code uses a special API called the Journaling Block Device (JBD) layer. The JBD has been designed for the express purpose of implementing a journal on any kind of block device. Ext3 implements its journaling by hooking in to the JBD API. For example, the Ext3 filesystem code will inform the JBD of modifications it is performing, and will also request permission from the JBD before modifying certain data on disk. By doing so, the JBD is given the appropriate opportunities to manage the journal

on behalf of the Ext3 filesystem. The JBD is being developed as a separate, generic entity and it could be used to add journaling capabilities to other filesystems in the future.

There are a number of ways to implement a journal. For example, a filesystem developer could design a journal that stores spans of bytes that need to be modified on the host filesystem. The advantage of this approach is that your journal would be able to store lots of tiny little modifications to the filesystem in a very efficient way, since it would only record the individual bytes that need to be modified and nothing more.

JBD takes a different approach. Rather than recording spans of bytes that must be changed, JBD stores the complete modified filesystem blocks themselves. The Ext3 filesystem also uses this approach and stores complete replicas of the modified blocks (1KB, 2KB or 4KB) in memory to track pending IO operations.

At first, this may seem a bit wasteful. After all, complete blocks contain modified data but may also contain unmodified (already on disk) data as well. The approach that the JBD uses is called physical journaling, which means that the JBD uses complete physical blocks as the underlying currency for implementing the journal. In contrast, the approach of only storing modified spans of bytes rather than complete blocks is called logical journaling.

Because Ext3 uses physical journaling, an Ext3 journal will have a larger relative on-disk footprint than those using logical journaling. But because Ext3 uses complete blocks internally and in the journal, Ext3 does not deal with as much complexity as it would if it were to implement logical journaling. In addition, the use of full blocks allows Ext3 to perform some additional optimizations, such as "squishing" multiple pending IO operations within a single block into the same in-memory data structure. This, in turn, allows Ext3 to write these multiple changes to disk in a single write operation.

#### ***2.2.4.3.2 Journaling for both meta-data and data***

Ext3 filesystem provides both metadata and data journaling to ensure both data and metadata integrity. Originally, Ext3 was designed to perform full data and metadata journaling. In this mode (data=journal), the JBD journals all changes to the filesystem, whether they are made to data or metadata. Because both data and metadata are journaled, JBD can use the journal to bring both metadata and data back to a consistent state. The drawback of full data journaling is that it can be slow, although you can reduce the performance penalty by setting up a relatively large journal.

A new journaling mode has been added to Ext3 that provides the benefits of full journaling but without introducing a severe performance penalty. This new mode works by journaling metadata only. However, the Ext3 filesystem keeps track of the particular data blocks that correspond with each metadata update, grouping them into a single entity called a transaction. When a transaction is applied to the filesystem proper, the data blocks are written to disk first. Once they are written, the metadata changes are then written to the journal. By using this technique (data=ordered mode), Ext3 can provide data and metadata consistency, even though only metadata changes are recorded in the journal. Ext3 uses this mode by default.

### **2.3 Distributed File Systems**

Distributed file system consists of file system servers, file system clients, and the specification of the file system service interface to the client. The performance of distributed file systems is affected by several key design decisions. Among them are the statelessness or statefulness of servers, caching location, and semantics of file sharing.

Servers are either stateless or stateful. Stateless servers maintain no information regarding the history of client requests. Since servers maintain no state information, requests must contain all necessary information to describe server tasks. Stateless behavior simplifies recovery; clients only need to resend requests that do not successfully complete. Stateful servers maintain information about previous client requests. With this knowledge, clients and servers effectively pre-fetch and cache data. State information is difficult to rebuild if lost by server failures.

Distributed file systems cache data at a variety of locations. These locations include main memories and storage devices of clients and servers. Data location largely determines client access times. Access times are shortest for data cached in client memories. Access times are longest when servers cache data on local storage devices.

File systems support various degrees of file sharing. There are two important sharing models. The first model, UNIX semantics, states that when a READ operation follows a WRITE operation, the READ returns the value just written. This model is easy to understand and straightforward to implement. A more relaxed model, session semantics, states that changes to an open file are initially visible only to the process that modified that file and only when the file is closed are the changes made visible to other processes.

Cache inconsistencies occur when clients modify one or more copies of the same data. To prevent or manage inconsistencies, distributed file systems have mechanisms to ensure coherence. These consistency mechanisms either invalidate or update stale copies of data. Stateless servers typically rely on clients to maintain consistency. Before accessing locally cached data, clients must verify consistency with servers. With stateless servers, clients perform write-through caching so that other clients receive modified data. Stateful servers

take an active approach to consistency management. Servers perform callbacks to notify clients and other servers that cached data is inconsistent and must be written back or invalidated.

### ***2.3.1 Network File System (NFS)***

The Network File System (NFS) was designed by Sun Microsystems in 1985 [Sandberg 1985]. The goals of NFS include system independence, name transparency, and preservation of UNIX file system semantics. The server design is stateless, so clients make requests with all information necessary to complete operations. Clients and servers communicate over a network using the remote procedure call (RPC) protocol. RPC is a high-level protocol built upon User Datagram Protocol (UDP) and Internet Protocol (IP). Transmission Control Protocol (TCP) may replace UDP to provide connection-oriented communication with guaranteed, in-order delivery.

NFS servers are stateless and write modified data to stable storage before completing requests. NFS maintains only a weak form of consistency, since single read and write requests may span several RPC operations. Multiple clients may issue overlapping requests.

Both NFSv2 and NFSv3 [Callaghan 2000] clients and servers cache data in system memory in order to improve read performance. Clients also cache file attributes, but periodically invalidate these attributes to limit the use of stale data. Clients maintain file data consistency by verifying file modification times with servers.

The stateless server design is the crux of NFS simplicity. Servers use local file systems to store data; NFS does not manage storage. Primary server functions manage client

requests and transport data. Server statelessness simplifies crash recovery. Failed clients do not affect the operations of servers or other clients. Servers that fail need only reboot; clients resend requests not completed within a given duration. Clients perceive failed servers as slow servers.

NFS offers portability and high connectivity but lacks fundamentals necessary for good performance. Single servers become bottlenecks as the number and size of client requests increase. NFS is distributed in the sense that multiple computers share files; however, the design is not distributed in a manner capable of providing scalable performance. Single servers also make NFS vulnerable to failures.

### ***2.3.2 Andrew and Coda File Systems***

The Andrew File System (AFS) was developed from a joint research project between IBM and Carnegie Mellon University [Satyanarayanan 1990]. Coda descended from AFS research [Satyanarayanan 1989]. Both AFS and Coda are designed to operate on distributed networks of workstations scaling up to 5000 machines. AFS and Coda use locally attached storage devices on both servers and clients.

AFS distributes the file system across multiple server computers. AFS servers maintain state. Servers perform call-backs when client cached data is modified by other clients. AFS only guarantees consistency at the granularity of the entire file. When multiple copies of a file exist, servers save the last file written.

Transarc Corporation took AFS technology and developed the Distributed File System (DFS) [Bever 1993]. DFS is the basis of the Open Software Foundation (OSF) *Distributed*

*Computing Environment* (DCE) [Kazar 1990]. DFS provides stronger UNIX consistency semantics than AFS.

Coda, which stands for “Constant Data Availability”, improves the availability of AFS. Clients cache entire files locally in memory and on disk. Furthermore, multiple file copies may exist on different servers. Single server failures have little impact on availability. Clients may also run in *disconnected operation mode*, thereby using only locally cached files. Disconnected clients later reconnect to the network and synchronize modified files with the distributed system.

Like AFS, Coda distributes file manager responsibilities to server computers, although Coda clients also perform server-like functions during disconnected operation. For this reason, the Coda client file manager organization is a merged client/server system with private file managers. However, Coda server design is that of a distributed organization.

### 2.3.3 Zebra and xFS

Zebra network file system [Hartman 1993] is a log-structured file system striped over network storage nodes directly by clients. Zebra stripes client logs of recent file system modifications across network storage servers and uses RAID level 4 to ensure fault-tolerance of the each log. By logging many recent modifications before initiating a parallel write to all storage servers, Zebra avoids the small write problem of RAID level 4.

As in the Log-structured File System (LFS) [Rosenblum 1992.], Zebra uses stripe cleaners to reclaim free space. Zebra assumes clients are trusted; each time a client flushes a log to the storage servers, it notifies the file manager of the new location of the file blocks just written through a message, called a “delta” which is post-processed by the manager to

resolve conflicts with the cleaner. Zebra lets each clients write to the storage servers without going through the server and coordinates the clients and the cleaners optimistically with file server post-processing. By making clients responsible for allocating storage for new files across the storage servers, Zebra effectively delegates to the clients the responsibility of low-level storage management.

The Serverless Network File System (xFS) is part of the Network of Workstations (NOW) project at the University of California at Berkeley [Anderson 1995a]. xFS uses a log structured organization like the Log-structured File System (LFS) [Rosenblum 1992.] and striping techniques from Zebra [Hartman 1993] to simplify failure recovery and provide high throughput transfers. Fast, switched networks connect xFS clients.

The xFS project recognizes that central servers are performance and reliability bottlenecks. Therefore, xFS distributes traditional server responsibilities to the clients. Hence, any system can manage control directives, metadata, and real data. The serverless design attempts to improve load balancing, scalability, and availability.

xFS differs from Coda, and AFS, since xFS distributes metadata management across multiple nodes. In contrast, the other systems divide directory trees into subtrees and assign each subtree to different servers. xFS is a merged client/server architecture.

#### ***2.3.4 Petal and Frangipani***

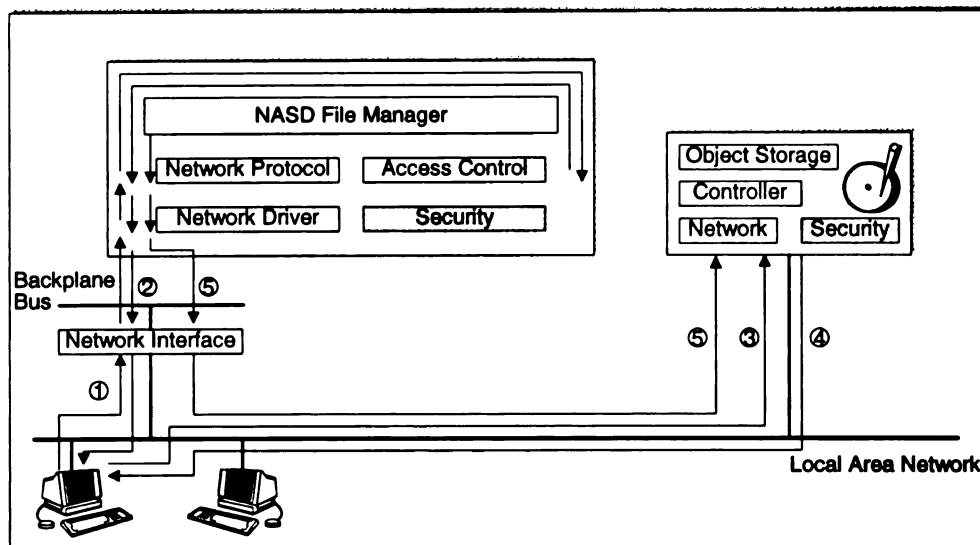
Petal was a research project at Compaq's Systems Research Center [Lee 1996] based on arrays of storage-appliance-like disk "bricks" but offering a block-oriented interface rather than a file interface. Petal scaled by splitting the controller function over a cluster of controllers, any one of which had access to consistent global state. As a Petal system's

capacity grew, so did the number of Petal servers in the cluster along with the performance they sustained. Logically, Petal could be viewed as a RAID system implemented on a symmetric multiprocessor, though it used distributed consensus algorithms instead of shared memory for global state.

Frangipani is a distributed file system based on the lower layer Patel, which provides a distributed storage service [Thekkath 1997]. Multiple machines run the same frangipani file system code on top of a shared Petal virtual disk, using distributed loc service to ensure coherence. Frangipani is designed to run in a cluster of machines that are under a common administration and can communicate securely.

### ***2.3.5 Network Attached Secure Disks (NASD)***

Network Attached Secure Disks (NASD) was a research project at Carnegie Mellon University. NASD exploits the computational power at storage devices to perform parallel and network file system functions, as well as more traditional storage optimizations [Gibson 1998]. The basic goal of the NASD project is to eliminate the server bottleneck from the storage hierarchy, and make disks directly accessible to clients. This eliminates the need to move all data from the disks, over a storage network, through the memory system of a server machine, over a client network, and to the clients.



**Figure 2-10 Network-attached secure disks (NASD)**

The NASD architecture can be summarized in its three key attributes:

- 1) **Direct transfer to clients;** Data accessed by a filesystem client is transferred between NASD drive and client without indirection (store-and-forward) through a file server machine. NASD are designed to offload more of the file system's simple and performance-critical operations. For example in Figure 2-10, a client, prior to reading a file, requests access to that file from the file manager (1), which delivers a capability to the authorized client (2). So equipped, the client may make repeated accesses to different regions of the file (3, 4) without contacting the file manager again unless the file manager chooses to force reauthorization by revoking the capability (5).
- 2) **Asynchronous oversight by file managers;** Access control decisions made by a file manager must be enforced by a NASD drive. This enforcement implies authentication of the file manager's decisions. The authenticated decision authorizes particular operations on particular groupings of storage. Because this

authorization is asynchronous, a NASD device may be required to record an audit trail of operations performed, or to revoke authorization at the file manager's discretion.

- 3) The abstraction of variable-length objects. The NASD interface abandons the notion that file managers understand and directly control storage layout. Instead, NASD drives store variable-length, logical byte streams called objects. Filesystems wanting to allocate storage for a new file request one or more objects to hold the file's data. Read and write operations apply to a byte region (or multiple regions) within an object. The layout of an object on the physical media is determined by the NASD drive.

To exploit the high bandwidth possible in a NASD storage architecture, the client-resident portion of a distributed file system needs to make large, parallel data requests across multiple NASD drives and to minimize copying, preferably bypassing operating system file caches. Cheops is a storage service that can be layered over NASD devices to accomplish this function. In particular, Cheops was designed to provide this function transparently to the higher-level file systems.

Figure 2-11 depicts how NASD, Cheops, and filesystem code fit together to enable direct parallel transfers to the client while maintaining the NASD abstraction to the filesystem. At each client, a file clerk performs file caching and namespace mapping between high-level filenames and virtual storage objects. The storage clerk on the client receives logical accesses to a virtual storage object space, and maps the accesses onto physical accesses to the NASD objects. Parallel transfers are then carried out from multiple NASDs into the storage clerk.

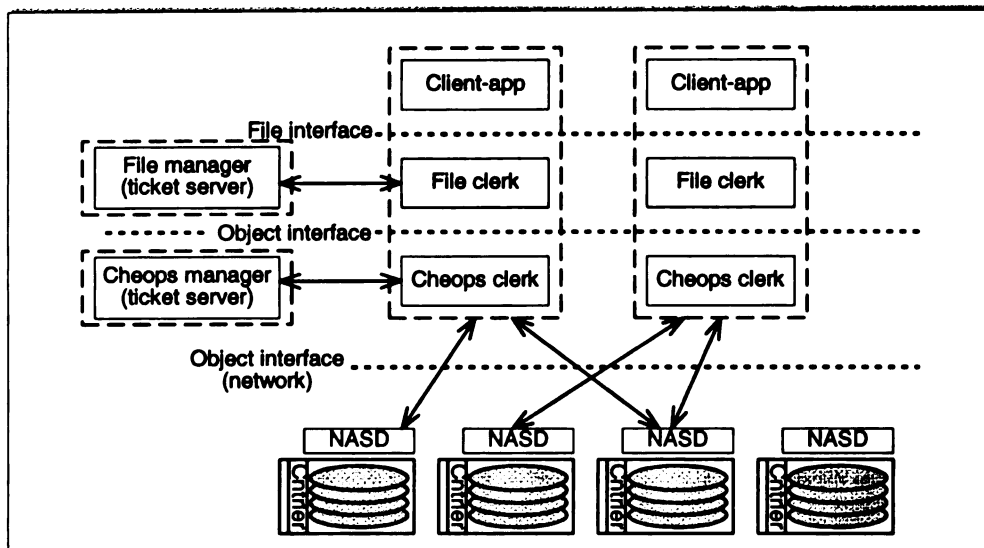


Figure 2-11 NASD/Cheops architecture

Cheops implements storage striping and RAID functions but not file naming and other directory services. This maintains the traditional “division of concerns” between filesystems and storage subsystems, such as RAID arrays. Cheops performs the function of a disk array controller in a traditional system. One of the design goals of Cheops was to scale to a very large numbers of nodes. Another goal was for Cheops to export a NASD interface, so that it can be transparently layered below filesystems ported to NASD. A NASD-optimized parallel filesystem, NASD/PFS, has been implemented using a cluster of workstations simulating NASD devices.

## Chapter 3

# The CoStore Storage Cluster Architecture and Performance Evaluation

### 3.1 Introduction

A CoStore system clusters a variety of network attached storage devices, each capable of providing file interface in addition to *block* interface. Using the same NAS approach, a CoStore cluster offers a *file* interface with built-in fault-tolerance to achieve strong reliability and availability, similar to modern storage appliances. With the system responsibilities evenly distributed across all collaborating storage devices, the proposed architecture provides scalable high-performance high-capacity storage services, traditionally only achievable by high-end storage systems.

Jim Gary predicts that, with IPv6 network interfaces and operating systems, storage bricks have arrived and they will evolve from block servers to application servers [Gray 2002]. Before these NAS disks materialize in the market, we will use commodity PCs with locally attached disks to simulate NAS disks. It is worth to point out that advanced storage systems can also be constructed in the same manner by clustering high-end computers attached with huge SAN devices.

Distributed computing based on COTS (commercial-off-the-shelf) components has been a burgeoning strategy to deliver high performance with superior cost effectiveness and flexibility [TFCC 2001]. The availability of inexpensive high-performance PCs and other high-grade commodity components, such as disk drives, network interface cards, memory chips, has made great sense for COTS-based storage clusters. The author believes that the CoStore cluster architecture using network attached storage devices can achieve the

high performance of advanced storage systems through ensembles of intelligent storage devices.

The potentials of commodity components are often underestimated in high-end storage system arena. With the disk capacity in excess of 100 GB, a PC can easily store up to half TB even with IDE interfaces, whose bandwidths also have reached to as high as 133MB/s [Fido 2001] per channel. Assisted by the latest SCSI320 [Mason 2000] a PC can store multiple TB with transfer rates up to 320MB/s per channel with up to 15 devices. Due to intense market competitions, the speed of processors has been too fast to easily name a CPU killer-application for personal computing. Cheap PC memory prices make extensive caching less expensive. With switched Fast Ethernet has become commonplace, Gigabit Ethernet is getting more affordable recently [Sander 2001a]. With disks scattered into PCs, we do not need to deal with the cooling, packaging and powering challenges commonly encountered in high-density storage servers.

## **3.2 Technology Trends in Networked Storage**

### ***3.2.1 Storage interfaces: block vs. file***

Gibson et al. classified storage device interfaces into two abstractions in [Gibson 2000]. *block*, is a simple, untyped, fixed-size (block), memory-like interface for manipulating nonvolatile magnetic media. Traditional disk drives (IDE or SCSI), disk arrays or even SAN rack systems essentially are all block devices. The other interface is *file*, which is a richer, typed, variable-size (file), hierarchical interface. Network attached storage (NAS) systems provide a *file* interface, which is similar to that of a traditional local file system. Storage appliances are in fact intelligent devices that provide *file* interface storage services

by hiding the details of managing internal nonvolatile media through a *block* interface. NAS systems provide the same functions as computers running distributed file systems with attached disks. Differently, NAS systems are normally based on aggressively specialized hardware and software and are attached internally with SAN systems. Storage appliances are NAS systems engineered to be especially simple to manage and extremely reliable, like a home appliance [Hitz 1997].

### ***3.2.2 Architecture taxonomy***

Gibson et al. present a taxonomy of network attached storage architectures in [Gibson 1996]. Case 0 in the taxonomy is server-attached disks (SAD) as shown in Figure 3-1. Disks locally attach to general server computers. Servers transfer data between server storage devices and memory via traditional I/O buses using protocols like SCSI. The data is transferred to client's memories via network links typically using protocols like TCP/IP.

Case 1 devices, known as Server Integrated Disks (SID), are more specialized computers that only perform distributed file system functions. The disks are still connected locally through I/O channels like SCSI. Case 2 is defined as Network SCSI (NetSCSI), which directly transfers data between clients and storage devices via SCSI over network protocols. NetSCSI is a network-attached disk architecture designed for minimal change in the disk's command interface. NetSCSI devices are block addressable. File manager computers facilitate file system operations and name-space manipulations between clients and storage devices.

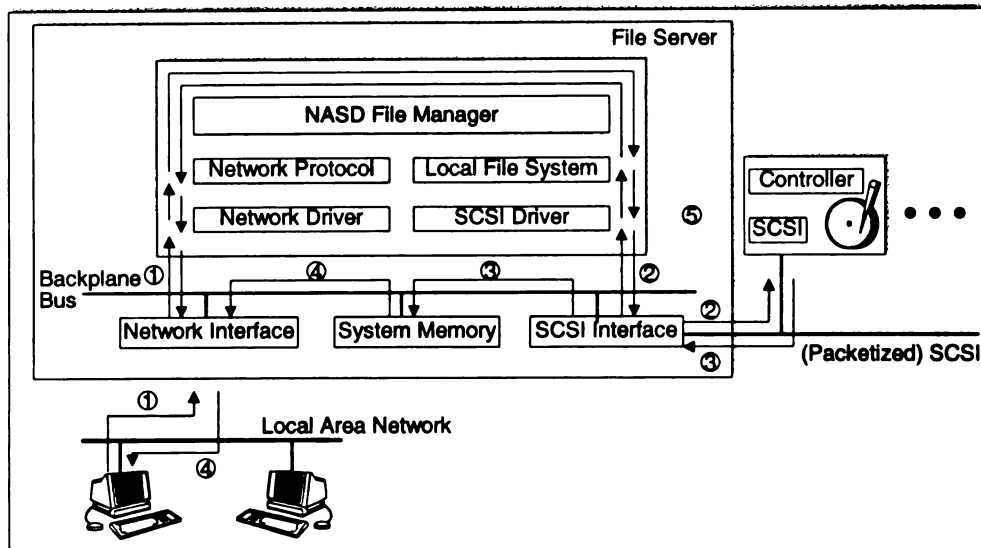


Figure 3-1 Server-attached disks (SAD)

Case 3 approach proposed by Gibson et al. is called Network Attached Secure Disks (NASD). NASD is an enhanced device interface that supports object addressable operations, which is higher level than NetSCSI, which is only block addressable. Objects may be data extents or files. Standalone NASD file manager maintains the name-space by performing standard file system functions. NASD devices authenticate client request via file managers to ensure security. The key enabling technology in NASD is a powerful on-drive microprocessor capable of executing the drive's embedded file system, networking and security code.

The evolution of storage architecture can also be categorized into three generations. Case 0 and case 1 in Gibson's taxonomy are the first-generation, which essentially includes computers connected to local disks via traditional I/O interfaces like SCSI. The more than 20 years old SCSI interface has several limitations: a clumsy parallel cable; limited distance to devices; scalability problem because only up to 16 devices can be connected to

each channel; and limited headroom for bandwidth growth, even though the newer version promising up to 160MB per second.

### ***3.2.3 Storage Area Network (SAN)***

The Storage Area Network (SAN) emerges as data communication platform to interconnect servers and storage devices at Gigabit speed. With a dedicated high-speed storage network, SAN eliminates the bandwidth bottleneck and both scalability and distance limitations haunted the first generation architecture for a long time. The ANSI standard for high-end storage interface is Fibre Channel Protocol for SCSI (FCP) on Fibre Channel – Arbitrated Loop (FC-AL) network. The other interface, Serial Storage Architecture (SSA) from IBM competing a few years ago, failed because of little market acceptance to the proprietary proposal and much lower bandwidth. Both FC-AL and SSA are serial storage interface. If NetSCSI in Case 2 has a separate network for storage devices, or SAN connects disks using the same local area network, then SAN and NetSCSI share the same architecture.

The second-generation storage architecture is based on SAN. All the disks connected are combined together to provide a single virtual huge disk device through interconnect of Fibre Channel (FC) network. Also through FC interface, the huge virtual disk provides a block addressable abstraction to a general purpose computer, which runs as a distributed file system server for Sun Microsystems's Network File System (NFS), or Common Internet File System (CIFS) service [Leach 1997] from Microsoft or the open source Samba project.

The core concept of SAN is to use a fully connected FC-AL network infrastructure, rather than direct-attached SCSI devices, to manage a collection of storage devices. The

single FC-AL network connects storage devices and storage servers, but is still separate from the general-purpose network that connects clients and storage servers. This allows multiple hosts to share the same storage, but still requires clients to access data through intermediate storage servers rather than directly.

#### ***3.2.4 Network Attached Storage (NAS)***

The latest third-generation architecture is Network Attached Storage (NAS) with the concept of storage appliance. Storage appliances are servers that have been specialized to perform only storage service [Hitz 1994]. Storage appliances are designed to be especially simple to manage [Gibson 2000]. With higher-level interface, *file*, to clients, storage appliances provide storage service at the distributed filesystem level through standardized protocols including NFS, CIFS, and HTTP. Storage appliances are seamlessly integrated into the whole infrastructure. Storage appliances authenticate client requests through login servers via interfaces like Light-weight Directory Access Protocol (LDAP), or Sun Microsystems's Network Information Service (NIS) or Microsoft's Windows NT domain controllers.

NAS is more a trend than a new architecture, because most of the NAS devices of recent models are built using SAN technology (Figure 3-2). What really differentiates NAS devices from SAN devices is the former includes a file system and provides file interface while the latter has only block addressable interface [Gibson 2000]. NAS devices normally are standalone server with very specialized software and hardware connected with disk devices via high-speed SAN network, Fibre Channel for example. Compared with general-purpose computers, both the software and hardware are striped down with only the

necessary modules to handle: network communication; data redundancy management; file system management, and distributed file system functions among others. On the contrary, the onboard controller of SAN devices mainly implements data redundancy management. Another important differentiation between NAS and SAN devices is the former has a close association with Ethernet network hardware, while the latter with Fibre Channel network hardware [Gibson 2000]. Both architectures may use Fibre Channel as internal connection interface.

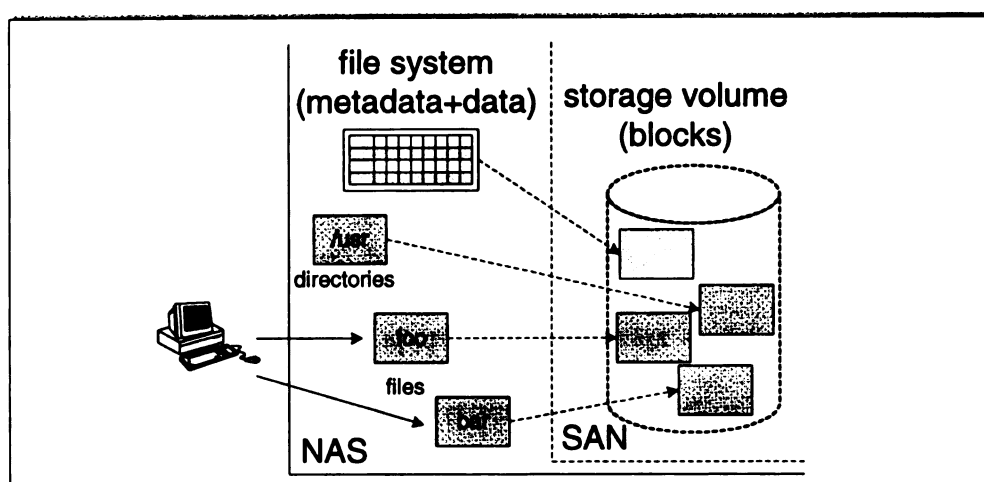


Figure 3-2 Network Attached Storage (NAS) vs. Storage Area Network (SAN)

### 3.2.5 Upcoming storage architectures

Providing an interface of files, the NASD architecture belongs to the third generation storage architecture. Different from most storage appliances, the standalone file manager in NASD connects to disks through local area network such as Ethernet. Not only do these disks have Ethernet interface, they also are more intelligent than standard disks on the market. These intelligent disks will perform more involving functions like object

manipulation than normal disks do with block interface in SAN. Data redundancy is provided as an extra layer called Cheops, running on clients.

Like Fibre Channel's command protocol FCP, Internet SCSI (iSCSI) is a SAN interface. According to [Van Meter 1998], it is a natural way to exploit the influence of the Internet by layering a block-level SAN protocol over Internet protocols, as was demonstrated by the Netstation project at the University of Southern California's Information Sciences Institute.

IPS (IP Storage) working group has chartered to work on security, naming, discovery, configuration, and quality of service for IP storage. iSCSI embodies the effort to generalize in storage-device networking. The IPS working group use TCP as transport mechanism to reliably deliver data over IP. However, the connection oriented-ness and the efficiency of TCP over higher-speed network have prompted some to propose an entirely different congestion-control algorithm that is appropriate for storage traffic.

Instead of the highly specialized super storage appliance, one major alternative approach is the use of PC clusters with a low-latency cluster interconnect based on network interface cards that can offload protocol processing from each machine's main processor [Intel 1997]. Such cluster approaches require specialized software. Network Appliance and Intel have proposed such file system architecture and protocol called the Direct Access File System (DAFS) [NetworkAppliance 2000]. The implementation of the proposed architecture in this study can take advantage of the direct file access protocol.

With the introduction of Virtual Interface Architecture (VIA) [Intel 1997], extremely high performance storage clusters can be implemented. VIA's Remote DMA (RDMA) support is capable of sending data in and out from I/O devices to the network, with none or

little involvement of server nodes' host CPUs. One example is the Direct Access File System (DAFS) project [NetworkAppliance 2000] proposed by Network Appliance and Intel.

### 3.3 A Storage Cluster Architecture Using Network Attached Storage Devices

#### 3.3.1 Architecture overview

A CoStore cluster implements a virtual storage server with a *file* interface as in other storage appliance products using NAS approach. To provide an interface of a single virtual cluster server, each cluster is assigned with a multicast IP address. All participating cluster members join in this multicast group, whose IP address is known to its clients. The main advantage of NAS approach is that internally the design can seamlessly integrate major storage components to work closely together. The close integration of local file system management and fault-tolerance management is essential to the efficiency of CoStore architecture, as in other NAS approach based storage server designs [Hitz 1994].

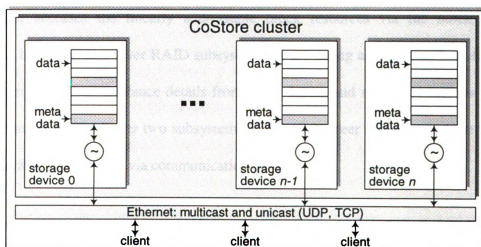


Figure 3-3 The structure of a CoStore cluster

As shown in Figure 3-3, a CoStore cluster consists of a group of network attached storage devices and clients that access the storage resources provided by the cluster. All members in a CoStore cluster work collaboratively to construct a storage system with a unified file namespace, i.e. one root directory. One multicast IP address for a CoStore cluster gives clients an impression of a single server initially. This virtual CoStore server is a serverless design because no central file manager is required to maintain the consistency of the overall namespace in the cluster. Both file system metadata (and hence local file system management) and distributed file system responsibilities are evenly distributed across all participating members.

Each of the storage device members in the cluster runs a program, called daemon, to serve clients' requests. There are three key functional modules in a daemon program (Figure 3-4). The top layer is the distributed file subsystem, which presents a *file* interface to CoStore clients. The middle layer is the local file subsystem, which manages block resources as part of the global namespace of the cluster. The block resources can be raw disks, disk partitions, or even large files in regular local file systems. The local file subsystem accesses the linearly addressable block resources via the *block* interface provided by the bottom layer RAID subsystem. By providing a *block* interface, the RAID subsystem hides fault-tolerance details from upper layers and maintains various levels of data redundancy. The lower two subsystems interact with peer daemons and the top layer interacts with clients, both via communication networks.

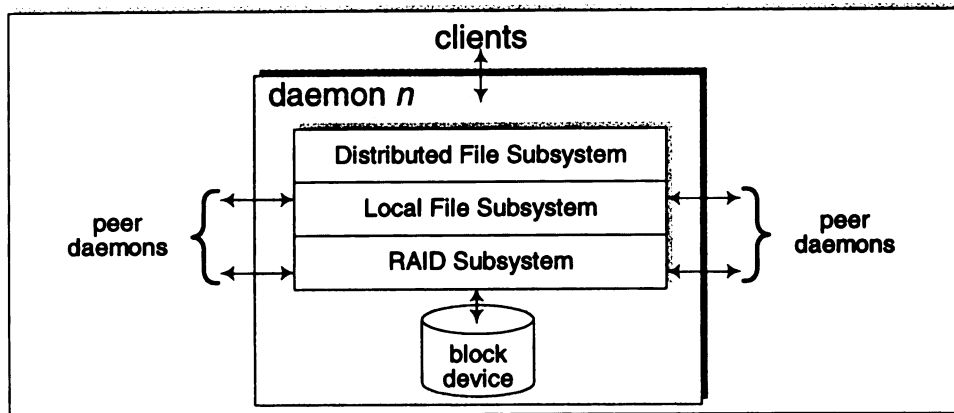


Figure 3-4 Functional modules in a CoStore daemon

### 3.3.2 Distributed file subsystem

Network File System (NFS) has been proved to be an extremely successful distributed file system protocol, whose success should largely attribute to NFS protocol's simple and stateless design [Callaghan 2000]. Not to reinvent the wheel, this virtual CoStore server adopts NFS Version 3 protocol [Callaghan 1995] as its interface to clients. Most NFS servers have been implemented using RPC, which is a mechanism for point-to-point remote procedure calls. However, because of the multi-entity nature of a cluster, traditional RPC library is not capable to support the one-to-many call from a client to a CoStore cluster. In our prototype we resort to low-level socket programming using UDP to implement the NFS protocol between a CoStore cluster and clients. Even though our prototype CoStore cluster implementation strictly follows NFS protocol, at binary level CoStore cluster is not compatible with the many traditional NFS client implementations.

Therefore at client end, a file system module is needed in order to access storage services from a CoStore cluster. Currently a standalone ftp-like program is used to simulate a regular client. Alternatively a CoStore proxy can be implemented to support standard NFS and MOUNTD calls from traditional NFS clients. With such a proxy, it is possible for

CoStore to reach more platforms without having to implement client modules for specific platforms because NFS is supported on most platforms.

Multicast communication is only used when necessary, such as at initialization stage when a client makes the first lookup call. All subsequent requests are sent to individual members' UDP ports using unicast. The reason is that each multicast involves all group members and may generate unnecessary network traffic to those machines that the request was not intended. More efficient unicast using UDP protocol is used for communications between a cluster and clients. Point-to-point TCP protocol is used for internal communications among cluster members.

As in NFS protocol, CoStore enforces Unix semantics for file sharing, that is, the latest write prevails. At current stage in CoStore, there is no caching at either server side or client side except the block buffer cache in CoStore's RAID subsystem. In future study we will look into whether caching at various levels can be exploited to improve performance without violating NFS's stateless design.

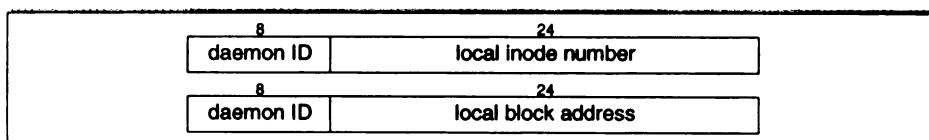
### ***3.3.3 Local file subsystem***

The purpose of a local file subsystem is to manage the linear addressable storage space on block devices. In CoStore, the local file subsystem manipulates an abstract block device through an interface, such as *read\_blocks()* and *write\_blocks()*, provided by the lower RAID subsystem.

In CoStore disk space is efficiently managed by individual local daemons in the same way as any generic local file system. Without loss of generality, we adopt the well-established Unix file system structure in the local file system management. Specifically

CoStore's local file system has the same layout as the *de facto* file system on Linux: Second Extended File System (Ext2) [Card 1986]. More sophisticated local file systems can be utilized in the future. For instance, the latest Ext3 and ReiserFS file systems support journaling [Robbins 2001], which makes file system consistency much more stable. Focusing on the cluster architecture the prototype implementation in this study only adopts local file system as generic as Ext2.

In CoStore Ext2 file system structure is furnished with a special block addressing and inode numbering scheme to unify all the local file systems on individual daemons. Both data blocks and *inodes* on each daemon are numbered in 32-bit integers. However the highest 8 bits are reserved for daemon's identification number (Figure 3-5). 32-bit integers (cluster capacity up to 16 TB) are sufficient in our initial prototype implementation. For large capacity clusters the current design can be easily extended to accommodate 64-bit integers.



**Figure 3-5 Block addressing and *inode* numbering scheme**

With this scheme, multiple local file systems on individual storage devices are combined into a global large file system with one hierarchy of namespace. This file system design is best summed up as Data Anywhere, Metadata at Fixed Locations. This design is efficient because of the locality of metadata management and yet flexible because of the data anywhere layout. The feature of Data Anywhere also includes indirect block pointers.

Figure 3-6 shows an example of three scenarios to allocate blocks for one file. The blocks allocated can be limited to local disk space only, or can be spread across multiple or all storage devices in the cluster.

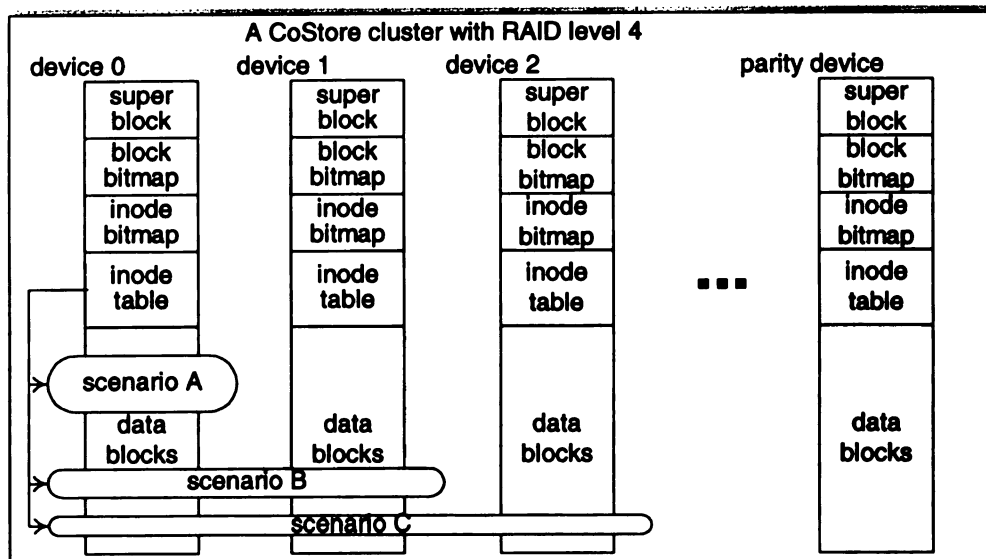


Figure 3-6 File system layout in a CoStore cluster with RAID level 4

Considering the metadata as directories for the data blocks, CoStore architecture resembles a CC-NUMA (Cache Coherency – Non-Uniform Memory Access) distributed shared memory multiprocessor system with directory-based cache coherence protocol as in DASH [Lenoski 1990]. In DASH, the local memory is managed using a directory by a local node. In CoStore, the local space is managed by each daemon using a metadata, essentially a directory (Figure 3-7). Likewise, the consistency of the global file system management in CoStore cluster is achieved with distributed locking on a per file (*inode*) basis. One daemon can request remote daemons to assign *inodes* or allocate blocks. After having obtained the exclusive lock to an *inode* on a remote storage daemon, the daemon can read to write to the data blocks for that *inode* via remote daemon's block interface.

Our CoStore prototype implements the traditional Unix-style security with permission bits in *inodes*. This is safe in a secure environment with daemons trusting the operating systems on client machines. Without major changes, more sophisticated security mechanism can be employed to enhance the CoStore design, which is beyond the scope of this study.

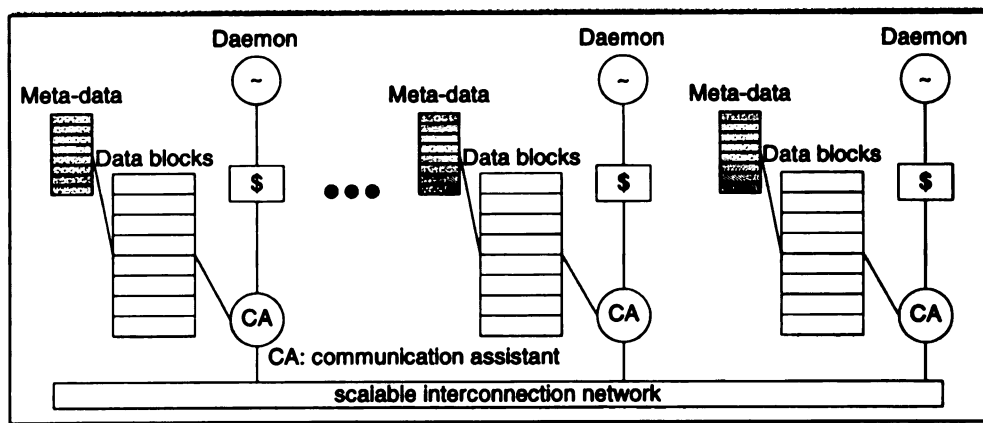


Figure 3-7 Architectural similarities between CoStore and DASH

### 3.3.4 RAID subsystem

The RAID subsystem provides an abstract block device to local file subsystem by hiding the fault-tolerance details. Internally RAID subsystem will read from or write to physical magnetic storage devices while maintaining data redundancy at RAID levels of user's choice. When a modifying *write\_blocks()* is called, distributed RAID semantics with various RAID levels has to be enforced by synchronizing mirror or parity updates to remote storage devices.

Fault-tolerance is essential when each individual storage device in a CoStore cluster is not reliable enough. In CoStore different RAID levels with variable reliability features are supported, each with different overheads in space or latency. The current CoStore

prototype supports RAID level 0, 1, 0+1 (or 10), 4 and 4+1. Support for RAID 5 and 5+1 is still ongoing. RAID level 5 is normally preferred because of its balanced performance. However, the parity-rotating scheme makes the distributed RAID implementation more complicated than other levels.

At the meantime RAID level 4 does have its own advantages: simplicity and incremental expandability. In a cluster of RAID 4, the capacity can grow by simply adding more devices without expensive RAID reconstruction. As a side note, RAID level 5+1, 4+1 and 0+1 are extremely useful in disaster avoidance applications, such as mirroring of clusters in two distant sites as we will explore separately in Chapter 5. One nice feature of RAID 0+1 is that storage devices in one cluster are not required to have identical capacity (except mirror pairs) because logical data blocks are not striped across all devices in a horizontal manner. Instead, all data blocks are managed locally.

With special techniques the parity bottleneck problem in level 4 can be significantly alleviated as has been demonstrated in products from Network Appliance [Hitz 1994]. Because all local file systems have identical layout, stripes in RAID 4 are aligned with respect to block types: metadata or user data as shown in Figure 3-4. The relatively more frequently updated metadata stripes are naturally consolidated into fewer parity updates, assuming that the traffic is balanced in terms of load and space utilization on all storage devices. Due to the stripe alignment, metadata stripes can be differentiated from ordinary data stripes with different priorities to update parity. The metadata consistency is critical to each file system's overall stability. Therefore, it is desirable to enhance local file system in CoStore with journaling support.

Because of longer than traditional delays, the concept of delayed parity updates in AFRAID [Savage 1996] will be helpful to improve performance when distributed RAID is enabled. In future study we will explore the effects of delays on the performance and reliability of CoStore systems.

#### **3.3.4.1 Block buffer cache**

All modern operating systems employ a block buffer cache system based on the concept described by Maurice J. Bach [Bach 1986]. However, the RAID functionality in CoStore requires having direct control of committing modified data blocks to storage devices. Therefore, the RAID subsystem is implemented in an independent block-buffer cache module. In public domain, there is no implementation more famous than the block buffer cache in Linux kernel [Beck 1998]. Our prototype borrows the *buffer\_head* data structure and hence the list-manipulating algorithms from Linux kernel.

*buffer\_head* is a header structure pointing to a block cache of buffered data. One buffer cache system includes a hash table used to accelerate *buffer\_head* searching, and three circular doubly-link lists: *lru\_list*, *dirty\_list* and *lock\_list* (Figure 3-8). At one moment, each *buffer\_head* can only belong to one list. Initially all the *buffer\_head* are stored in a free list. Once used, the *buffer\_head* is first added into the hash table according to the hash value based on block number. Using separate chaining method to solve hashing collisions, each hash table entry is a non-circular doubly-link list, which will be short if the table is relative large and the hash function is well chosen.



To speed up performance, asynchronous write is used to commit modified data blocks to media. Once a write request is issued, the *buffer\_head* will be moved from *dirty\_list* to *lock\_list*, which means that it can only be read and a new write has to block until the pending write is completed by block devices. Once the modified data is committed to the media, the *buffer\_head* can be removed from *lock\_list* and be inserted into *lru\_list* as the *MRU* or into *dirty\_list* if it is being modified again.

### 3.3.5 System operations

NFS protocol is based on a cornerstone called filehandle. Each file or directory has one filehandle in the file system. Each NFS procedure call has to provide a filehandle, upon which the request action can be performed. In production systems, filehandle embodies much more information. Without loss of generality we simply use the 32-bit inode number as filehandle in our CoStore prototype. As in Figure 3-5, there is one field in each filehandle that indicates which daemon this file or directory is hosted on.

The filehandle for the root directory in one namespace is a special one (all 0's), called public filehandle. Therefore clients can start from this well-know filehandle without having to resort to MOUNTD protocols. The most importance one in all NFS procedure calls is LOOKUP, which evaluates the pathnames relative to current path or root to find the target filehandle. NFS version 4 will support multi-component LOOKUP operation, which is not support in CoStore or NFS 3.

#### ***3.3.5.1 Cluster initialization***

At startup, each CoStore daemon reads configuration information from a local configuration file, which includes the daemon's ID number, RAID level, RAID role (storage-, mirror- or parity-daemon). It also includes the multicast IP address for the cluster, TCP and UDP ports, and various buffer sizes etc.

Multicast communication is used in the cluster's initialization process and can potentially serve as system status bus for the cluster. Each CoStore cluster is assigned one IP multicast address. At initialization stage, all daemons join this multicast group, exchange information with other daemons and construct a daemon's identification number to IP address (ID-to-IP) mapping table, shared by all participating daemons. When initialization is finished, each daemon listens on the UDP port of the multicast address and its own UDP port for requests. All daemons also establish a full-connection among them using TCP for intra-cluster communications.

#### ***3.3.5.2 Client initialization***

When a CoStore client first starts up, it needs to get hold of the root directory in a cluster. The client makes a LOOKUP call using the public filehandle by sending a request to the known multicast IP address. All daemons will receive the request. Only daemon 0 which hosts the root directory will answer the LOOKUP request and will piggyback one ID-to-IP table in the reply.

With the ID-to-IP mapping table piggybacked in the reply, the client can send requests directly to individual daemons based on the daemon ID field in the filehandle. The same request can also continue to be sent to the multicast IP address. Every daemon checks

requests received at the multicast address and only processes these requests with the daemon ID in the filehandle matching the daemon's own identification number.

### ***3.3.5.3 File and directory creation***

**File creation.** Files are created in the same way as traditional NFS systems by calling CREATE on target directory with a file name. Only the daemon owning the target directory will process the request. The daemon first validates the filehandle for the target directory in its local file system and then allocates one inode (and therefore the filehandle for the new file) from the inode table on that daemon. If the CREATE indicates the file size, the daemon will also allocate the desired blocks. Then the daemon replies the filehandle for the new file to the requesting client. Upon receiving the new filehandle, the client can write to or read from that file by calling WRITE or READ on that filehandle. All files are created on the same daemon as its parent directory.

**Directory creation.** Directory is a special file and can be created similarly by calling MKDIR. It seems that initially all files and directories will be created on daemon 0 and other daemons will have no business at all. To balance the load to all daemons, the MKDIR procedure is extended by an extra parameter: target daemon number. By default directories are created on the same daemon as its parent directory. The user can manually indicate the target daemon on which the new directory should be stored. For example, *mkdir foo 4* will make the new *foo* subdirectory created on daemon 4. Ideally the target daemon of new directories could be decided by all daemons dynamically to load-balance all new generated directories and files. The CoStore prototype has yet supported dynamic load-balancing.

#### ***3.3.5.4 Intra-cluster communications and deadlock***

A full-connection using TCP among all daemons is established at initialization time for intra-cluster communications. So far there are three different kinds of intra-cluster requests: local file system (LFS), logic block interface, and physical block interface for distributed RAID. The physical block interface is used for parity updates. Because we are using TCP, there is no Ack for each physical block updates, therefore it is deadlock free.

However, we are less fortunate at LFS level and logical block interface level. The main purpose of LFS level is primarily for remote directory creation (MKDIR). By imposing a one-way direction of remote MKDIR we can effectively remove potential loops. That means remote directories can only be created from lower daemons on higher daemons.

A remote MKDIR involves two LFS requests between daemons. First the source daemon sends an LFS\_NEW\_INODE\_NO request for a new inode on target daemon. Second, the source daemon sends an LFS\_ALLOC\_BLOCK request so that blocks are allocated for a directory with a default size, which will have at least two entries: self (.) and parent (..). Then the source daemons writes the initial two entries (. and ..) using the logical block interface. The imposed one-way direction for remote MKDIR also removes the need of read and write to the logical block interface in two-direction, and thus eliminates potential deadlocks at the logical block level. Therefore CoStore is deadlock free.

Because of the three-pass communication involved in each remote MKDIR call, the latency is relatively high. In a regular local MKDIR, the latency is about 0.005 second, while the latency for a remote MKDIR is about 0.18 second. Fortunately remote MKDIRs are rare and most files and directories are created locally.

### ***3.3.5.5 Recovery from node failures***

Without data redundancy from the RAID subsystem, CoStore has a single point of failure because the root directory in the global file system for one cluster is hosted on daemon 0 only. To overcome node failures, CoStore clusters should be configured to have parity in one cluster using RAID 4 or 5, or to mirror individual daemons using RAID 0+1, or both, using RAID 4+1 or 5+1.

When storage daemons are mirrored in a master-slave pair in RAID 0+1, 4+1 or 5+1, failure recovery is relatively easy and straightforward. Slave nodes can upgrade themselves and take the responsibility of master nodes. There may be unrecoverable data lost when master node failed. How to maintain a consistent file system (with journaling) on slave nodes is an important and interesting problem and deserves closer scrutiny in further studies.

For clusters with only parity data, recovery takes longer as we have to reconstruct a new storage node from the surviving node similar to disk failures in disk RAID arrays. The reconstruction is an expensive operation in terms of data traffic and XOR computation. Theoretically the cluster can continue operation while construction is performed in background as a degraded mode disk array. However, it is extremely to implement the online reconstruction in a distributed manner and the file performance from failed nodes would be very slow. So far, none of the recovery features has been implemented in our prototype, however.

### 3.4 Prototype Implementation

A CoStore prototype has been implemented on Windows 2000 and will be ported to other platforms once implementation of all functionalities has been done. CoStore is still an ongoing project. Currently the prototype only supports RAID level 0, 1, 0+1, 4 and 4+1. RAID level 5 and 5+1 is under development as of this writing. We have conducted some measurements on basic file read and write operations on the CoStore prototype, in comparison with other non-cluster distributed file systems: NFS on Linux and the file-sharing service (CIFS) [Leach 1997] on Windows 2000. Our experience told us that the CIFS service from Samba on Linux is slightly outperformed by that of Windows 2000. Therefore we did not include Samba in our experiments.

The limitations of our CoStore prototype are: i). it is only appropriate in a secure environment and it requires strong encryption schemes if use is expanded beyond LAN environment; ii). modifications are necessary to existing NFS client implementations.

CoStore systems can consist of a variety of storage devices: PCs running different operating systems, high-end servers attached to SAN systems, or even onboard controllers on NAS devices. Considering the potential heterogeneity in various platforms, CoStore systems should be implemented as independent of operating systems as possible. For example, CoStore can be implemented in Java so as to take advantage of its Write Once, Run Anywhere feature. To implement CoStore in Java is an increasingly attractive choice as more *JINI* storage devices are available [Heyn 1999].

At this time the CoStore prototype is implemented in C at user-level. Ideally CoStore's block buffer cache and hence the RAID subsystem should be implemented at kernel level of underlying operating system for efficiency and stability, but at the expense of portability.

Planning to port the CoStore prototype to other platforms, the authors have begun investigating the availability of aforementioned requirement in various Unix environments, particularly the open source Linux and BSDs.

A CoStore system requires at least the following support from underlying operating systems: network communications, memory management and block device I/O interface. TCP and UDP sockets are used for communications between clients and the cluster and among cluster members. Large-memory management is needed for block buffer cache subsystem. Asynchronous I/O support is necessary for efficient accesses to storage devices.

### **3.5 Performance Evaluation**

#### ***3.5.1 Experimental setup***

The experiments were conducted on a cluster of PCs. In most tests, PCs for servers and clients are Pentium III 1GHz with 128MB PC133 SDRAM. There are two tests where PCs with different speed are used. In one case PCs with Pentium III 550MHz are used as clients and in another case PCs with Pentium III 1.2GHz are utilized for both clients and servers. A 100Mbps Ethernet switch connects all the PCs. The storage devices on the servers are Quantum Fireball Plus AS20.5 hard drives (Ultra ATA/100 interface and 7200RPM).

For CoStore both client programs and server daemons are running on Windows 2000 Professional. For NFS on Linux and CIFS on Windows 2000 Professional tests, the client side is Linux. The client mounts the remote file share as either an *smbfs* share from Windows 2000 or an *nfs* share from NFS on Linux. A kernel-based NFS server (0.2-1) is running on GNU/Linux (kernel 2.4.2) for NFS tests. The kernel-based NFS service has a slightly better performance than user-level one.

All the tests are a set of consecutive requests for read or write with variable file sizes. We arrange the tests into two groups according to file sizes: 32KB to 512KB for small-size and 1MB to 128MB for large-size. Each file size has 4 different files and an average is taken as the result for that size. All tests are conducted on a *cold* system, i.e. no data is in the cache before read or write requests start.

Unix command *dd* is invoked to read from or write to remote files, with a constant block size 32KB. For example, *dd bs=32k count=8 if=/rmt/256k.dat.a of=/loc/256k.dat.a* is used to read 256KB from remote server to local disk. Unix command *time* is used to measure the total elapsed time for each request. In CoStore, the client measures the elapse time for each request by checking the system clock.

The CoStore prototype can choose to use the standard block buffer cache (OS-buff) from the operating system (Windows 2000), or use an independent block buffer cache subsystem (RAID-buff). When independent block buffer cache subsystem is enabled in CoStore, the reserved memory size for block buffer is 32MB and 1MB for mirror or parity update buffer. When OS-buff is used, read or write accesses to storage devices are buffered by the block buffer from operating systems. In that case, when data is modified, mirror or parity updates have to be sent to the RAID partners no matter how small the modification is. To send out many tiny packets is very inefficient use of the network resources. With an standalone block buffer cache, we could maintain a separate buffer to accumulate small mirror or parity updates into larger ones.

When RAID-buff is used, the block updates can be configured as Transaction Commit or Lazy Commit. Transaction Commit means updates will be committed to devices when one status-modifying NFS request, such as *nfs\_write()*, is processed. Lazy Commit means

dirty block buffers will be flushed to devices periodically (3 seconds in our tests) or when dirty buffers are swapped out. Similarly mirror or parity update buffers can be configured as Transaction Commit or Lazy Commit. Here by committing mirror or parity updates, we only mean that the update has been sent out to the network. It does not mean that the updates have been received by network partners or committed to remote storage devices. Normally block buffer and mirror or parity buffers are configured using the same policy: either lazy or transactional.

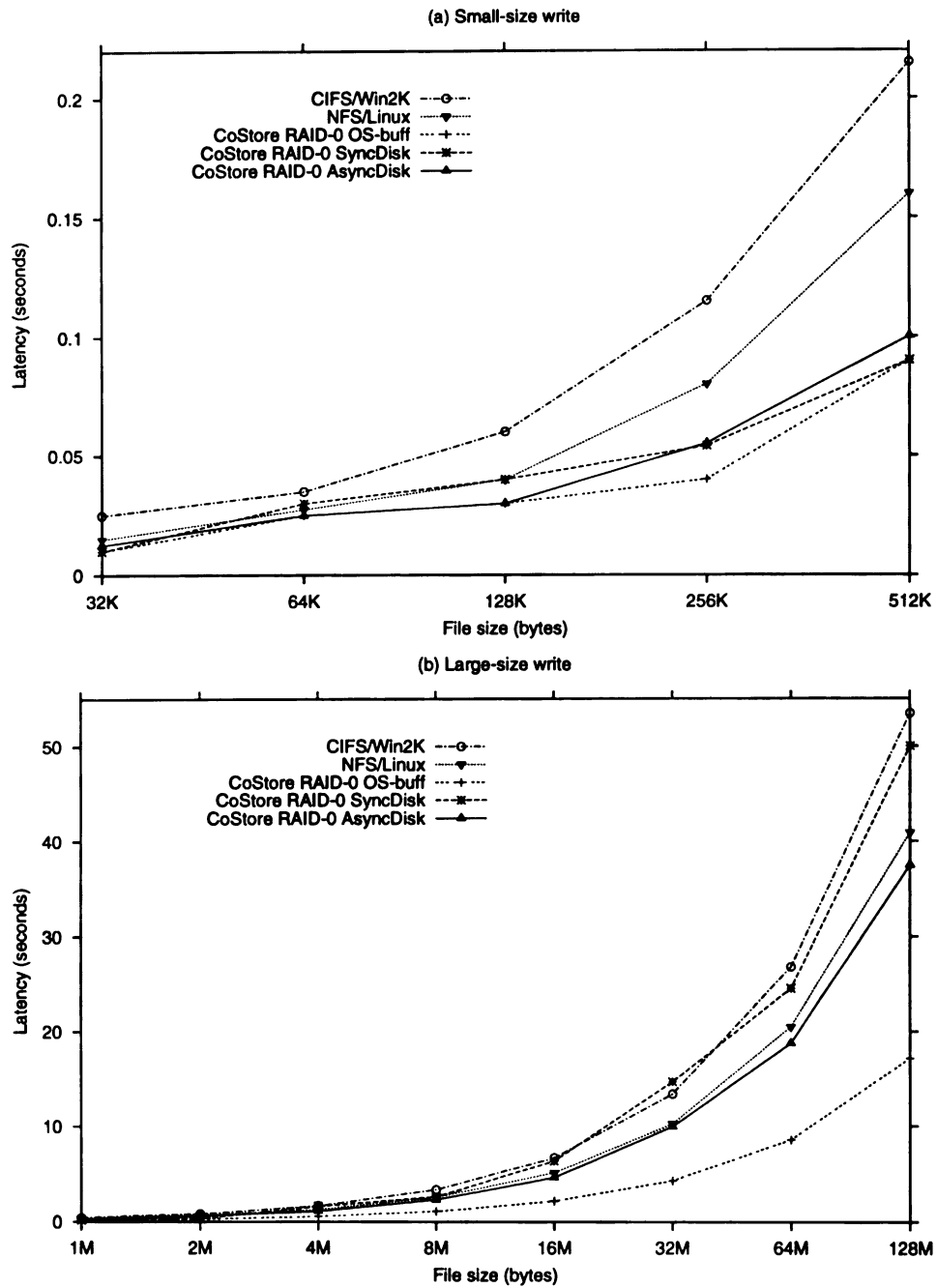


Figure 3-9 The write performance of CoStore

### ***3.5.2 Write performance: CoStore vs. NFS vs. CIFS***

Figure 3-9 compares a single-daemon RAID-0 CoStore's write performance with those of NFS and CIFS. In both small-size and large-size accesses, the performance of CIFS from Windows 2000 lags behind others. In Figure 3-9(a), all CoStore cases outperform NFS. But the result is susceptible to background noise due to very small latencies in tiny accesses.

In Figure 3-9(b), CoStore with asynchronous disk I/O (AsyncDisk) using only 32MB buffer cache slightly outperforms NFS. By enabling AsyncDisk, we also disable the standard block buffer in the operating system for our storage devices. CoStore with synchronous disk I/O (SyncDisk) has a much worse performance, only a little better than CIFS of Windows 2000. The reason is that with SyncDisk, the buffer cache at operating system level for storage devices is forced to flush every 3 seconds.

Interestingly, CoStore using only standard OS block buffer has an outstanding performance in Figure 3-9(b). There are several reasons: Windows 2000 takes advantage of the bulk of free memory as buffer cache (more than 64MB in our case); it has a much longer flushing threshold for dirty blocks; and it also does read-ahead caching. Because of the RAID support and efficiency reason for network mirror or parity updates, we need to maintain a separate block buffer cache and have a more conservative commit policy.

### ***3.5.3 Read performance: CoStore vs. NFS vs. CIFS***

Figure 3-10 shows the read performance of CoStore. It is hard to compare read performance when access sizes are small in Figure 3-10(a). However, in Figure 3-10(b) CoStore underperforms both NFS and CIFS by a large margin. CoStore with SyncDisk or OS-buff has an almost identical performance. But CoStore with AsyncDisk is considerable

slower than SyncDisk or OS-buff. The reasons are two-folded. First the buffers in OS-buff and SyncDisk can occupy almost all free memory. Second the operating system can do optimizations with read-ahead.

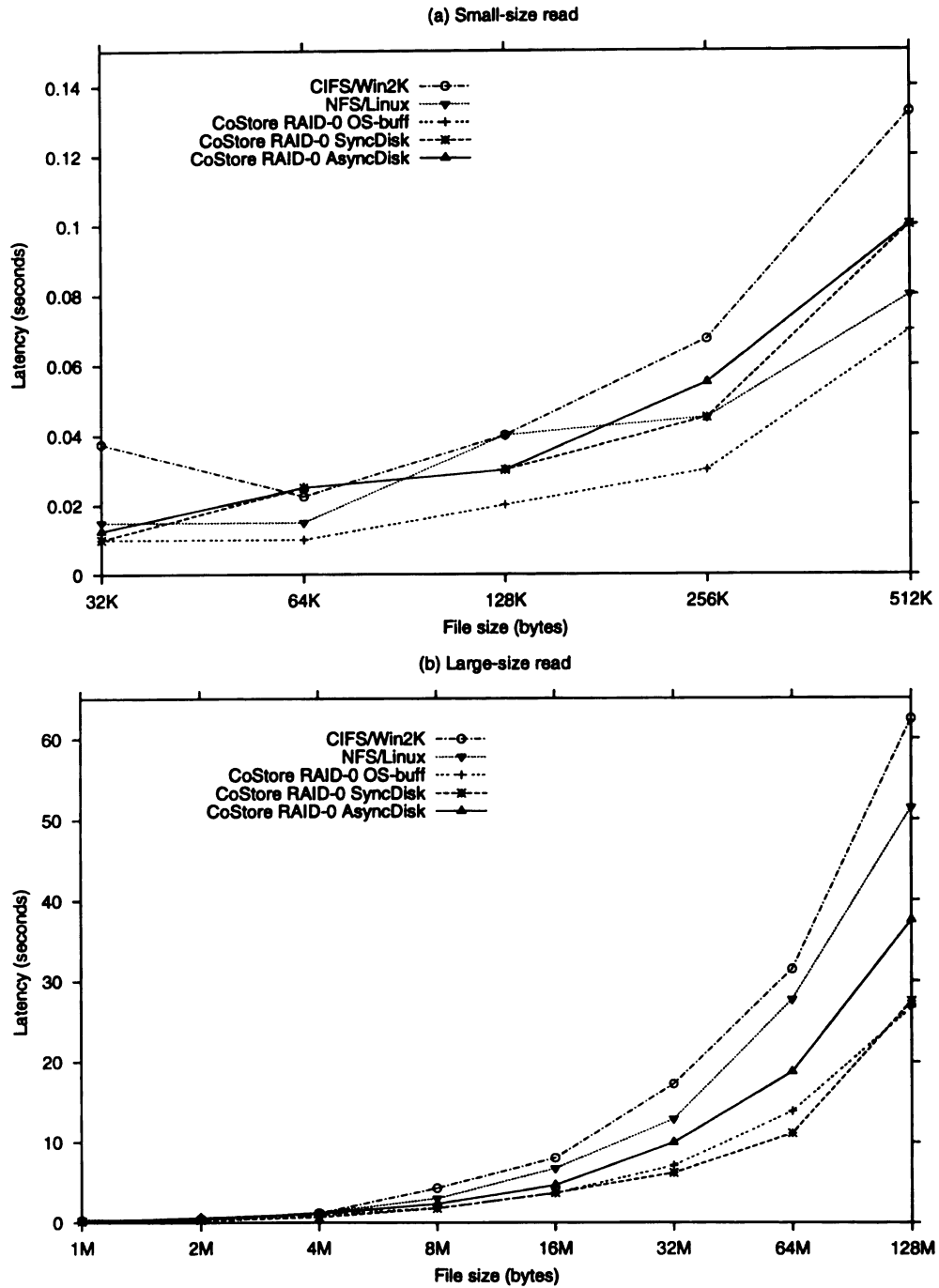


Figure 3-10 The read performance of CoStore

### **3.5.4 Impact of distributed RAID and commit policy**

In Figure 3-11 we evaluate the impact of distributed RAID and the effect of commit policy on the performance of CoStore with only one client. Figure 3-11(a) illustrates the overhead of synchronizing mirror or parity updates to network targets. We first verify that the 1-daemon and 2-daemon RAID-0 CoStore clusters have the same performance when there is one client. In the early tests we found that the 2-daemon CoStore cluster had longer write latency because in the implementation the daemon called *select()* with 1  $\mu$ sec timeout to check requests from peer daemons. With the timeout set to 0, there is no difference between 1-daemon and 2-daemon RAID-0 CoStore clusters for single client. We will further demonstrate the scalability of RAID-0 in Figure 3-12(a).

The overhead of copying updates to network partners in RAID-1 is relatively small. For example, to write a 32MB file in RAID-1 costs 11.12s, or 1.25s more than 9.87s in RAID-0. The extra mirror copying incurs about 12.66% overhead. To write the same file in RAID-4 costs 11.80 second, or 1.93s more than in RAID-0. The extra parity update incurs about 19.55% compared with non-redundant RAID-0. The reserved buffer for either mirror or parity is 1MB. At the mirror and parity target daemon, dirty blocks are committed to devices once a set of updates (up to 1MB) is received.

Figure 3-11(b) demonstrates the effect of commit policy on the latency in RAID-0 CoStore with 1-daemon and 1-client. The Transaction Commit does have significant impact on the latency of each request. To write a 32MB file, the latency is 12.55s with Transact Commit policy, compared to 9.87s with Lazy Commit policy, which times out every 3 seconds. That is a slowdown of about 27.02%.

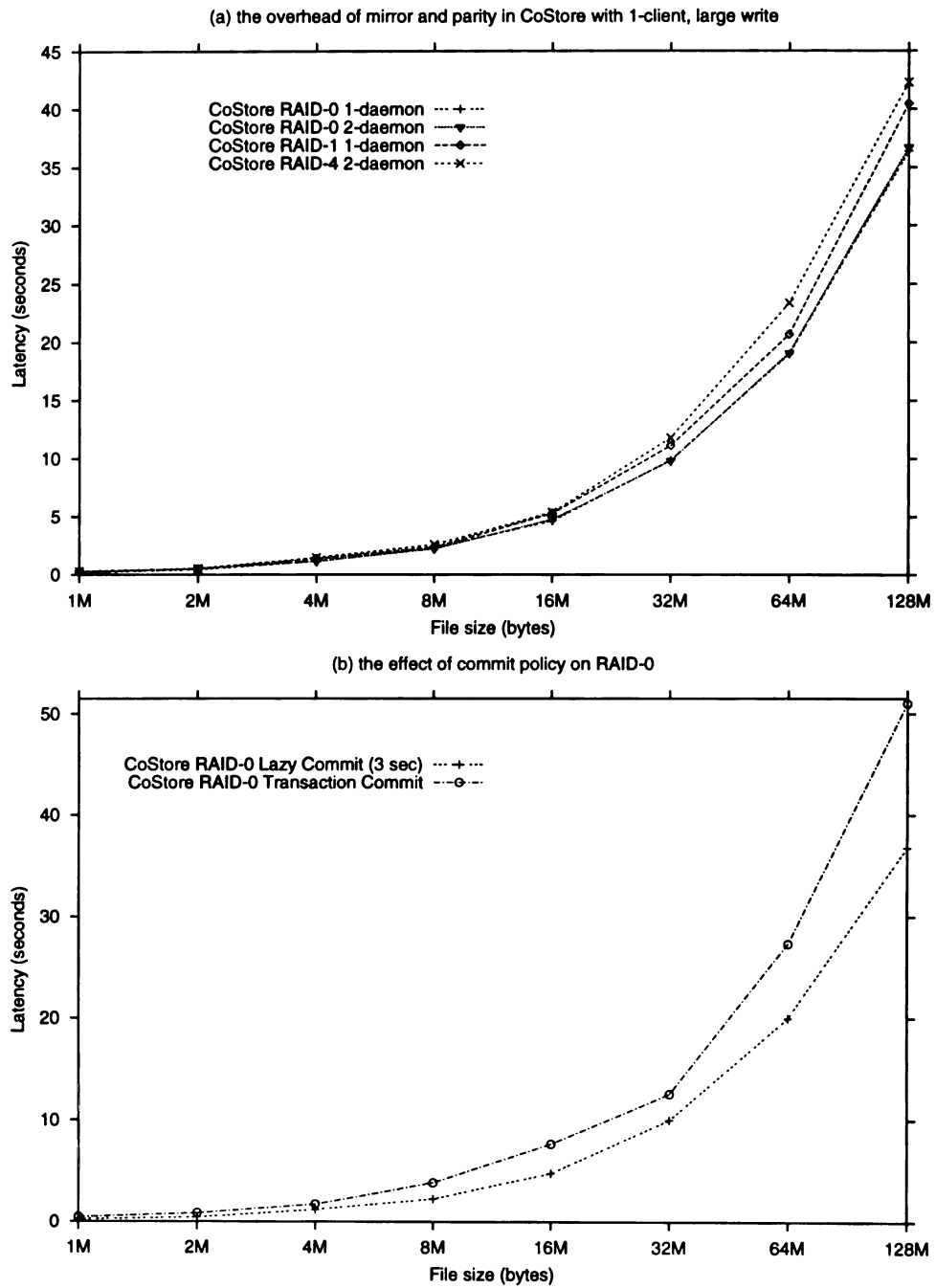


Figure 3-11 The impact of distributed RAID overhead and commit policy

### ***3.5.5 Scalability of CoStore***

Figure 3-12(a) illustrates a 2-daemon RAID-0 CoStore cluster is almost linearly scalable in terms of the daemon number, when each of the two clients is only requesting service from one of the two daemons in an ideal case. The two curves are approximately identical except at the size of 128MB, where the average latency of two clients is a little less than that of single client, possibly a result of noise due to system background activities. However, RAID-4 CoStore clusters are less scalable than RAID-0. Figure 3-12(b) illustrates a RAID-4 CoStore cluster (two daemons plus parity daemon). For two clients to write a 32MB file concurrently, the average latency is about 12.77s, while it takes 11.80s if there is only one client. The slowdown is about 8.2% for an extra client-daemon pair in ideal cases. Please note in Figure 3-12(b) both of the two clients are Pentium III 550MHz PCs; but the two members in the cluster are Pentium III 1GHz PCs.

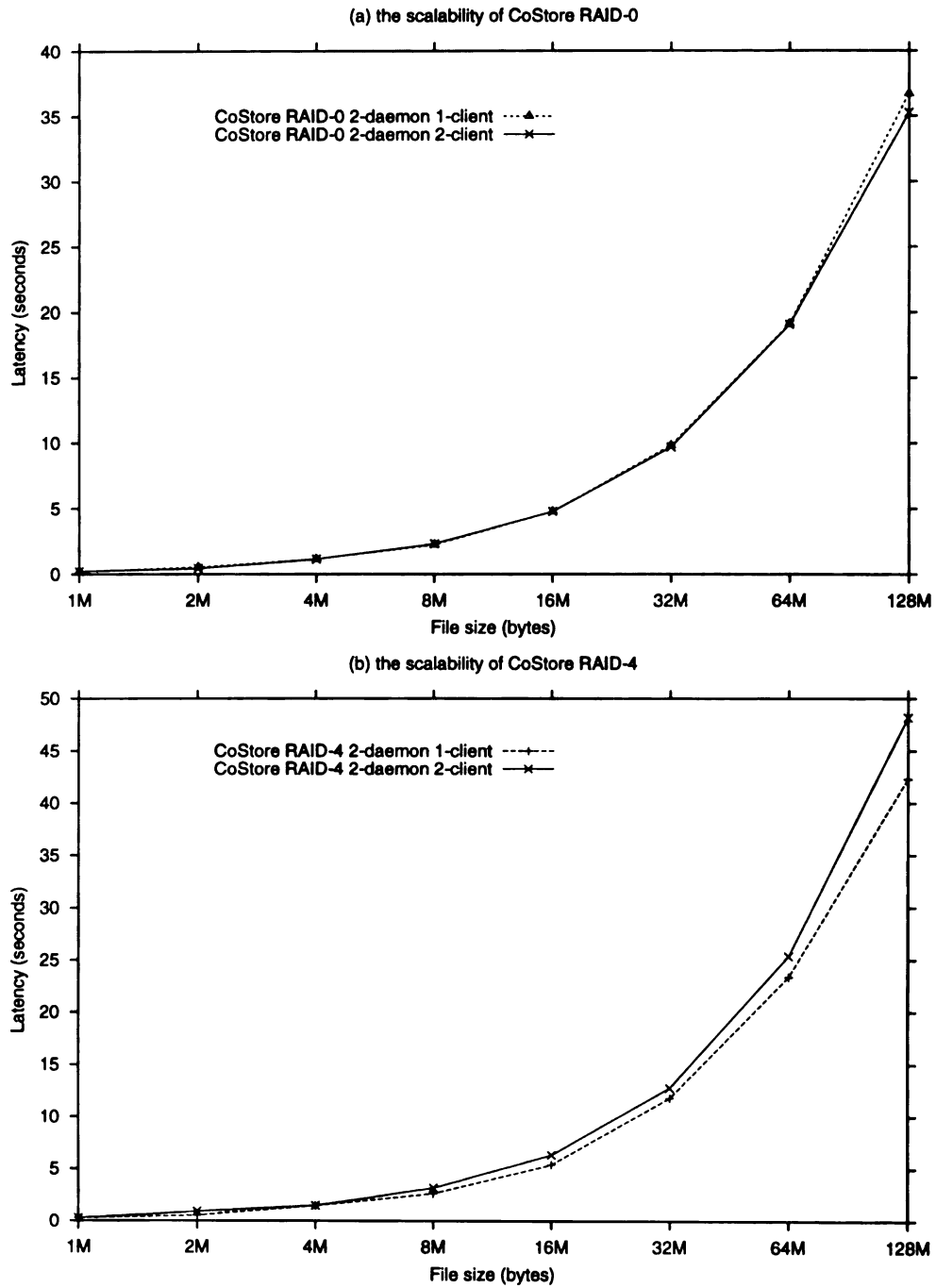


Figure 3-12 The scalability of CoStore clusters

### 3.5.6 Parity daemon's bottleneck effect in RAID-4

To evaluate the parity daemon's bottleneck effect in RAID-4, we also experiment with 4 clients on a CoStore cluster of 5 daemons (one is parity). Even though each client is requesting on one daemon, the average access latency increases gradually as the number of concurrent clients grows from 1 to 3 (Figure 3-13). It seems that the parity node reaches the saturation point where the average latency jumps sharply when there are 4 clients. As the group size and the number of concurrent clients increase, the parity node in RAID-4 may quickly become a potential bottleneck due to limited I/O bandwidth and extensive XOR computation on that node. The parity bottleneck should become less a problem in CoStore when the more balanced RAID-5 support is implemented. Please note in Figure 3-13 all the clients and servers are Pentium III 1.2GHz PCs.

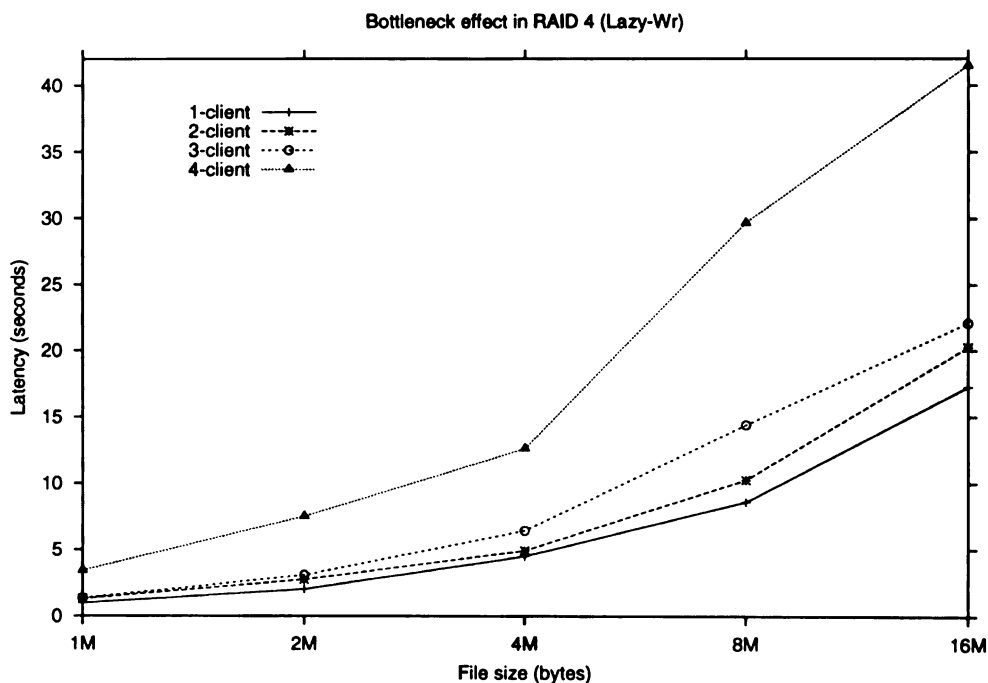


Figure 3-13 Parity daemon's bottleneck effect in CoStore with RAID level 4

### **3.6 Summary**

The NAS approach has been proven to be effective in constructing scalable storage server clusters using network attached storage devices. In CoStore the consistency of a unified file namespace is collaboratively maintained by all participating cluster members without any central file manager. The enabling factor for CoStore's efficiency and scalability is the flexible Data Anywhere, Metadata at Fixed Locations file system layout. The serverless design eliminates any central server bottleneck and provides strong scalability.

The CoStore prototype using COTS-based components demonstrated feasibility of building such scalable storage clusters. The performance results measured on the prototype illustrate the potential of CoStore to achieve scalable high-performance high-capacity storage services with strong reliability and availability.

## **Chapter 4**

### **CoStore Clusters Utilizing Idle Disk Space on Workstations**

#### **4.1 Introduction**

In recent years, the computer industry has made significant advances in magnetic recording technology. At the meantime the economy of volume of storage has improved enormously from the mass production of hard disk drives. Consequently, disk drives are getting higher in capacity, smaller in size and cheaper in cost—a trend that is expected to continue [Ng 1998]. The standard size for disk drives on mainstream computers in the market is about 20-30GB as of March 2001 and is growing continuously over the time.

To an end user, there are two different kinds of storage services provided in a well-organized computing system: local disk space on the client desktop machines and remote space on network storage servers. Most users prefer the network storage for various reasons including:

- 1) Mobility. The users are more on the go and they prefer to be able to access their data through consistent interface from any place they are in. Most client machines are behind firewall and/or may not be able to easily provide local resources to other systems.
- 2) Quality of Service. Normally the network space is stored on high performance highly available storage systems, with built-in redundancy to counter most hardware failures. A well-maintained storage system also has regularly scheduled backups so that information can survive catastrophic accidents or even natural

disasters. On the other hand, most client machines use commodity disk drives and normal do not have local backups.

- 3) Security assurance. System security is much easier to maintain on a centralized storage system managed by professional administrators than a group of systems managed by individual owners. Well-organized systems provide peace of mind for information security.

As a result, in many organizations most of the local disk space on client workstations is only used for operating systems, application programs and temporary files, which in total take up only 2 to 5GB disk space. Douceur and Bolosky measured and analyzed a large set of client machines in a large commercial environment with more than 60,000 desktop person computers [Bolosky 2000; Douceur 1999]. The measurement includes disk usage and content; file activity; and machine uptimes, lifetimes and loads. The result shows that about 53% and 50% of the overall disk space of studied environment is in use in September 1998 and August 1999, respectively. The disparity of space utilization ratios on storage servers and local machines is expected to deteriorate further over the time as the average disk size grows rapidly.

Motivated by the increasingly pervasive resource wasting, the CoStore cluster architecture was originally proposed to construct a storage system utilizing idle disk space on workstation clusters. In this chapter we evaluate the client-computing environment and assess the feasibility of deploying CoStore cluster on existing computing infrastructure.

It is worth to point out that in typical office environment, dedicated system administrators loosely manage most desktop machines. The term of being supported by

technical staff may better describe the relation. It is reasonable to assume that each of seat owners has almost full control of the local resource on his or her desktop machine(s).

There is another common model: centralized system management. In this model, end users are not tied to desktop machines. Instead, they go to the public lab and use whatever system available. The system administration team supervises the whole infrastructure. This model is very popular in universities, public libraries, and technical training centers in large corporate or government organizations. Even in the non-centralized environment, many of the non-technical personnel, for obvious reasons, choose to totally rely on technical support to take care of their systems. This study mainly focuses on the centralized management environment, even though it can also apply to the non-centralized model.

More research work is warranted by the fact that the cost for advanced storage servers continues to be expensive and that there is growing prevalence of idling local disk. Limited work has been done so far, partially because of the complexity of the diverse environment and the administrative overhead involved. To our best knowledge, previously only Microsoft research project Farsite [Microsoft 2000] tries to solve the same problem.

## **4.2 Assumptions and Environment Description**

The subjects in this study are front-end desktop workstations, compared with back-end time-sharing servers. A typical example is client workstations in a public lab or an engineering lab environment. Computing seats in such environments are on a first-come, first-serve basis and the local disks are primarily used to store files for operating system and application programs.

We assume that in such environments there exists a central administration with login authentication servers and storage servers with a unified namespace. All these servers and client workstations are connected in a secure local area network behind a firewall. Therefore, the servers can trust the operating systems on these workstations. This study mainly focuses on the centralized model, even though it can also be applied to the non-centralized one.

The foremost characteristic of such environments is heterogeneity in terms of hardware platforms and operating systems. Generally these workstations are well equipped with fast processors, large amount of memory, and high bandwidth local I/O and network interfaces. Another noticeable fact is that these workstations are susceptible to occasional unexpected reboots, due to software failure, user choice, or system-sharing policy.

The primary objective of this study is to transform the idle disk space into usable storage service with satisfactory reliability and efficiency. A good solution should require little administrative effort; otherwise, the prohibitive human cost may overshadow the gain from recovered resources. An ideal solution should provide additional storage service as a seamless part of the current storage infrastructure. CoStore is not intended to replace main storage servers. Instead CoStore attempts to provide extra storage space supplementary to main storage servers with little or no further investment.

Potentially the recovered storage space can be used for archiving purposes for both end-users and system administrators, such as large multimedia data or system snapshots files for online backup. Other possible usages include web page caching, website replication, or data buffering for search engines. Most of them are not frequently updated and some of them require little or no redundancy at all.

### 4.3 Alternative Solutions

There are different approaches to solve this problem. We can use existing software installed on these workstations to manually setup and combine individual file system resource into one large file system. This is called ad hoc solutions. We can also adopt the concept of virtual disks in Petal [Lee 1996] by using Network Block Device [Breuer 2000] driver available in Linux kernel (2.1.101 or later). The peer-to-peer movement has emerged as an interesting approach to solve the idle space problem in very large scale.

#### 4.3.1 Ad hoc solutions

It is possible to solve the idling disk space problem using existing software installed on current systems. Specifically, on UNIX workstations disk space can be shared via NFS or Samba service; on Windows machines disk space can be shared through CIFS protocol [Leach 1997]. On UNIX platform, *automount* can be used to construct one unique file system namespace. The Distributed File System (DFS) service on Windows 2000 Server has the same function to coalesce multiple file resources on Windows 2000 Server or Samba (2.2.0 or later) into one single namespace. However, this is a manual process and may be cost-prohibitive because of the extensive human work involved in the setup. There is no data fault-tolerance in this *ad hoc* solution except the fact that Microsoft DFS service can provide file-based replication.

#### 4.3.2 NBD based solutions

The distributed RAID approach used in xFS [Anderson 1995b] and Petal [Lee 1996] can be applied to build a *virtual* disk with block interface. On top the *virtual* storage devices,

higher level file system and distributed file system can be deployed, like the *metadata manager* in xFS and Frangipani [Thekkath 1997]. The result is a centralized reliable storage service that can easily be integrated into the existing storage infrastructure.

Using the same concept, we can build one or many virtual disks using the Network Block Device (NBD) driver on Linux. The Network Block Device driver simulates a block device, such as a hard disk or hard-disk partition, on the local host, but connects across the network to a remote host that provides the real physical backing [Breuer 2000]. Locally the device looks like a disk partition, but it is a façade for the remote. The remote host is a lightweight piece of daemon code providing the real access to the remote device and does not need to be running under Linux. The local operating system will be Linux and must support the Linux kernel NBD driver and a local client daemon. NBD setups can be used to transport physical devices virtually anywhere in the world. To introduce redundancy, we can build a software RAID out of multiple NBD devices. On top of block devices from simple NBD or software RAID on top of multiple NBD devices, any local file system can be chosen and so can any distributed file system available on Linux.

However, the relatively independent relation between RAID module and NBD module prevents efficient error handling when there is any network fluctuation to the TCP connections between the Linux server and remote hosts serving the physical block resources. This single server might be a potential bottleneck and it can combine limited number of NBD devices. To manage a local file system on top of network connections without special caching can be an issue in term of both efficiency and reliability. For obvious reasons, these remote hosts can do much more than only serving block resources.

### ***4.3.3 Peer-to-peer solutions***

One of the Internet's recent phenomena is the introduction of peer-to-peer (P2P) computing. The current P2P movement was started by a simple motivation for many people: the need to exchange music files. There have been several prominent peer-to-peer systems offering file-sharing services, such as Napster [Napster] and Gnutella [Gnutella]. The peer-to-peer movement has grown to more than file-sharing. This peer-to-peer phenomenon has reached to many areas, including distributed computing and distributed storage, to take advantage of the abundant processing power and disk resources widely available on millions of PCs.

There have been several peer-to-peer approach based storage systems: Farsite [Douceur 2001], PAST [Druschel 2001], and CFS/Chord [Dabek 2001; Stoica 2001] compared to the so-called server-to-server [Yianilos 2001] based systems such as OceanStore [Kubiatowicz; Rhea]. Many contemporary distributed file systems [Callaghan 1995; Leach 1997] are based on client-server model.

A P2P network distributes information among all member nodes instead of concentrating it at single server [Parameswaran 2001]. Using a P2P approach, Douceur et al. has proposed Farsite, a serverless distributed file system to solve this specific problem [Douceur 2001]. From the measurement of machine availability including uptime and lifetime, [Bolosky 2000] concluded that the measured desktop infrastructure would passably support their proposed system Farsite [Douceur 2001], providing availability on the order of one unfilled file request per user per thousand days.

The lack of security and central authority in P2P solutions makes such solutions less ideal in public labs environments that we are trying to solve. The efficiency of P2P

solutions remains to be evaluated. P2P is best for information sharing in large scale and its candidacy in storage is still an open question.

### 4.3 Feasibility Assessment of Deployment on Existing Desktop Computing Infrastructure

#### 4.3.1 Reliability theory in RAID

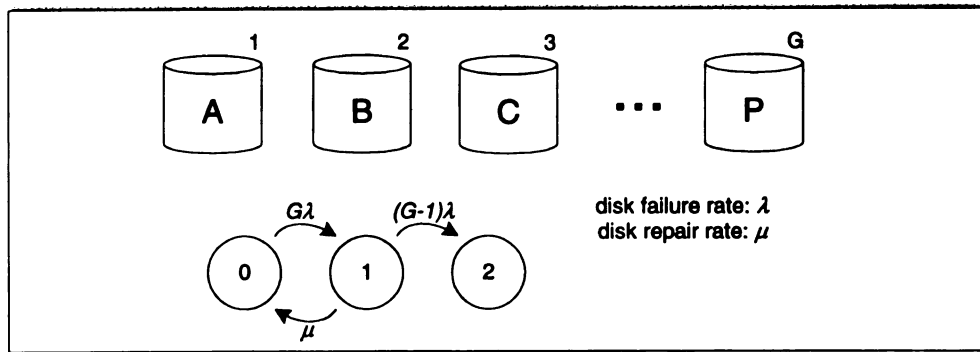


Figure 4-1 Queuing theory in the reliability of disk arrays.

Patterson et al. analyzed the reliability of disk arrays using queuing theory [Patterson 1988]. Assuming a constant disk failure rate – that is, an exponentially distributed time to failure – and that failures are independent – both assumptions are made by disk manufactures when calculating the Mean Time To Failures ( $MTTF_{\text{disk}}$ ) or Mean Time Between Failures (MTBF). In Figure 4-1, disk failure rate  $\lambda$  is  $1/MTTF_{\text{disk}}$ ; and disk repair rate  $\mu$  is  $1/MTTR_{\text{disk}}$ . Without any parity, a disk array's reliability is:

$$MTTF_{\text{disk array}} = \frac{MTTF_{\text{disk}}}{\text{number of Disks in the Array}} .$$

To overcome the reliability challenge, extra disks are introduced to provide redundant information to recover the original information when a disk fails. Disk arrays are broken

into reliability groups, with each group having extra check disks containing the redundant information. When a disk fails, it is assumed that within a short time the failed disk will be replaced and information will be reconstructed on to the new disk using the redundant information. This time is called the Mean Time to Repair ( $MTTR_{disk}$ ).  $MTTR_{disk}$  is far smaller than  $MTTF_{disk}$ .

A few notations for the following discussion:  $G$ : number of data disks in a group;  $C$ : number of check disks in a group,  $C$  is 1 for RAID level 4/5. There will be data loss when a second disk fails before the first failed disk is repaired within  $MTTR_{disk}$ . The probability  $P_{data\ loss}$  for this to happen is:

$$P_{data\ loss} = [1 - (e^{-MTTR_{disk} / MTTF_{disk}})^{(G+C-1)}].$$

Since  $MTTR_{disk} \ll MTTF_{disk} / (G+C)$  and  $(1 - e^{-x})$  is approximately  $x$  when  $0 < x \ll 1$ , therefore:

$$P_{data\ loss} = MTTR * (G+C-1) / MTTF_{disk};$$

and  $MTTF_{group} = \text{Expected}[\text{time between failures}] * \text{Expected}[\text{number of disk failures until data loss}]$

$$\begin{aligned} &= \frac{\text{Expected}[\text{time between failures}]}{P_{data\ loss}} \\ &= \frac{MTTF_{disk}}{G+C} * \frac{1}{MTTR_{disk} * (G+C-1) / MTTF_{disk}} \\ &= \frac{(MTTF_{disk})^2}{(G+C) * (G+C-1) * MTTR_{disk}}. \end{aligned}$$

Similarly, the proposed architecture reliability depends on the reliability of each individual system whose idling disk space is utilized. Likewise we define the following notations. For disks the Mean Time To Failure is denoted as  $MTTF_{disk}$  and the Mean Time

to Repair as  $MTTR_{disk}$ .  $MTTR_{disk}$  is far smaller than  $MTTF_{disk}$ . For disk arrays the Mean

Time To Failure is  $MTTF_{group}$ . Therefore,  $MTTF_{group}$  is  $\frac{(MTTF_{disk})^2}{(G + C) * (G + C - 1) * MTTR_{disk}}$ .

With the latest disk manufacturing technologies, modern disk drives on client machines are very reliable. The main failing factor in this study would be the system availability of desktop workstations. Due to failures in operating system, or user's choice to reboot the system for whatever reason, the mean uptime of client machine is very small compared to  $MTTF_{disk}$ . Thus, we consider the  $MTTF_{disk}$  of individual disk drive infinitely large.

However, when the system reboots, it normally can restore to working condition in 2 or 3 minutes, including grace time for all services to get started. This period of time for system to reboot is very small compared to the mean system uptime. We define  $MTTF_{ws}$  as the mean system uptime; and  $MTTR_{ws}$  as the mean time that each system reboot needs to restore to fully working status.  $MTTR_{ws}$  is far smaller than  $MTTF_{ws}$ .

When the number of concurrent rebooting workstations exceeds  $C$ , the service of this cluster will be interrupted, even though this does not render permanent data-loss as it would when too many physical disks failed in a RAID group. To evaluate the service interruption rate, we apply the same formula and thus the  $MTTF_{cluster}$  for a cluster consisting of desktop

workstations is  $\frac{(MTTF_{ws})^2}{(G + C) * (G + C - 1) * MTTR_{ws}}$ . For RAID 0+1, data are striped across on

$G$  mirrored workstation pairs without check data. For RAID 4+1, data are striped across on  $C$  mirrored workstation pairs with check data on another mirrored pair. As  $MTTF_{RAID1}$  is

$\frac{(MTTF_{ws})^2}{2 * MTTR_{ws}}$  and  $MTTF_{RAID4}$  is  $\frac{(MTTF_{ws})^2}{(G + 1) * G * MTTR_{ws}}$ , therefore  $MTTF_{RAID0+1}$  is

$$\frac{MTTF_{RAID1}}{G} = \frac{(MTTF_{ws})^2}{2 * G * MTTR_{ws}}; \text{ and } MTTF_{RAID4+1} \text{ is } \frac{(MTTF_{RAID1})^2}{(G+1) * G * MTTR_{ws}} =$$

$$\frac{(MTTF_{ws})^4}{4 * (G+1) * G * MTTR_{ws}^3}.$$

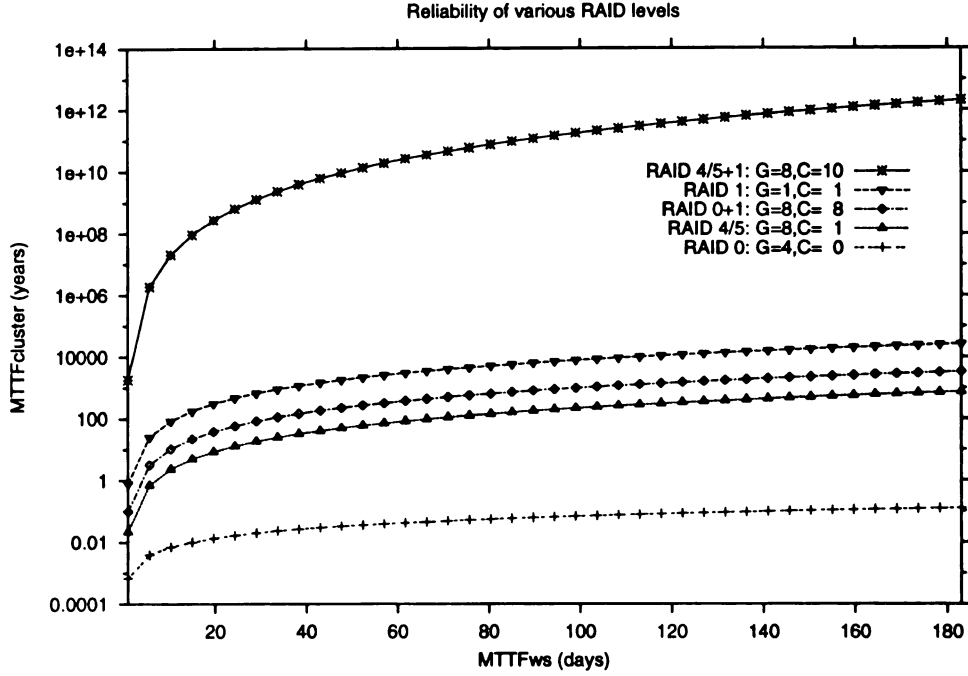


Figure 4-2 The relationship between  $MTTF_{cluster}$  and  $MTTF_{ws}$

To illustrate the relationship between  $MTTF_{ws}$  and  $MTTF_{cluster}$  with variable RAID configurations and different group sizes, we plot the above formulas in Figure 4-2. The mean reboot time  $MTTR_{ws}$  is 3 minutes. On the  $x$  axle, the mean system uptime  $MTTF_{ws}$  ranges from 1 day to 180 days. Typical system uptime  $MTTF_{ws}$  measured in real environment goes from 5 to 20 and 40 days.

#### ***4.3.2 Feasibility assessment***

To assess the feasibility of constructing a CoStore cluster on an existing desktop computing infrastructure, we collected system uptime data from public labs in Computer Science and Engineering Department at Michigan State University. There are seven instructional labs (18 to 20 seats each): four Sun UltraSPARC labs and three PC labs. The system activity monitor has recorded all rebooting events into a database. All the machines have been monitored during the school year of 2000-2001, except that the monitoring of three PC labs was started in January 2001. Table 4-2 shows the mean system uptime MTTFws and the measured reboot time MTTRws for each of the labs.

Assume that we construct 14 independent clusters from workstations in 7 labs. Normally the lifetime for desktop workstations is 3 years. Therefore, within three years we accept one occurrence of service interruption. In other words the number of concurrent rebooting workstations exceeds the number of workstations for check data in one cluster. Statistically the  $MTTF_{cluster}$  is equivalent of  $3*365*14$  or 15330 days. With this number in mind as a hypothetical criterion, we evaluate different RAID configurations in Table 4-2.

Table 4-1  $MTTF_{ws}$  and  $MTTR_{ws}$  based on system uptime data

Lab	Architecture	OS	Start Date (midnight)	End Date (midnight)	Average Reboot numbers	Average uptime (days)	Reboot time (sec.)
Aleutians	SparcUltra1	Solaris 2.8	28AUG00	05MAY01	15.23529	19.02635	120
Bones	SparcUltra10	Solaris 2.8	28AUG00	05MAY01	10.57895	25.75677	120
Indonesians	SparcUltra1	Solaris 2.8	28AUG00	05MAY01	11.88235	23.24829	120
Sounds	SparcUltra10	Solaris 2.8	28AUG00	05MAY01	15.00000	18.21376	120
Mountains	P-II/333	GNU/Linux 2.2	01JAN01	05MAY01	2.94737	42.22810	150
Rivers	P-II/400	Win2000 Pro.	01JAN01	05MAY01	25.58333	5.461001	180
Snakes	2xP-III/600	Win2000 Pro.	01JAN01	05MAY01	31.35714	4.925493	150

Table 4-2 shows the  $MTTF_{cluster}$  value for each cluster with different RAID configurations based on the  $MTTF_{ws}$  data from Table 1. From Table 2 we can determine that, using RAID level 4/5 or 0+1, the last two labs cannot construct a cluster with satisfactory reliability according to our previous criterion. The reason is that they both are too unstable or reboot too often (every 5 days). With RAID level 4/5, only three of the first five more stable labs can construct a cluster with RAID 4/5 when group size is small ( $G=4$ ). When using RAID 0+1, almost all the first five labs can construct reliable clusters with group size up to 8. When RAID 4/5+1 is used, even with group size 8 all the seven labs can make up clusters with exceptional reliability. The reason is that RAID 4/5+1 is an extremely reliable configuration and the check disks are more than data disks. How much impact the reliability of RAID level 4/5+1 will have on the performance remains to be evaluated in our further study.

Table 4-2 The reliability ( $MTTF_{cluster}$ ) of clusters

Lab	MTTF node (days)	MTTR node (sec.)	MTTF RAID4/5 G=4 (days)	MTTF RAID4/5 G=8 (days)	MTTF RAID4/5 G=16 (days)	MTTF RAID1 G=1 (days)	MTTF RAID0+1 G=4 (days)	MTTF RAID0+1 G=8 (days)	MTTF RAID4/5+1 G=8 (days)
Aleutians	19.0	120	13032.1	3620.0	958.2	130321.0	32580.2	16290.1	1.70e+11
Bones	25.7	120	23882.8	6634.1	1756.1	238828.0	59707.0	29853.5	5.70e+11
Indonesians	23.2	120	19457.4	5404.8	1430.7	194574.0	48643.5	24321.7	3.79e+11
Sounds	18.2	120	11942.7	3317.4	878.1	119427.0	29856.7	14928.3	1.43e+11
Mountains	42.2	150	51356.5	14265.7	3776.2	513565.0	128391.0	64195.6	2.11e+12
Rivers	5.4	180	715.7	198.8	52.6	7157.4	1789.4	894.7	3.42e+08
Snakes	4.9	150	698.7	194.1	51.4	6987.0	1746.8	873.4	3.91e+08

Table 4-2 confirms that theoretically it is feasible to deploy the CoStore architecture on existing desktop workstation clusters. With extra handling of the rebooting effect in daemon's implementation, the reliability of resultant clusters can be further improved.

#### 4.4 Summary

Using an NAS approach, CoStore is a novel solution to build reliable storage service with efficiency, utilizing idle disk space on desktop workstations. The enabling factor for system efficiency is the flexible Data Anywhere, Metadata at Fixed Locations file system layout. The serverless design eliminates any central server bottleneck and can provide improved scalability. System uptime data collected from production systems confirmed the viability of constructing storage cluster systems utilizing the idle disk space on desktop computing infrastructure.

## **Chapter 5**

### **Reliable and Highly Available Storage Systems Using CoStore Clusters**

#### **5.1 Introduction**

According to Todd Gordon of IBM, 70,000 workstations and 10 million square feet of rentable space were lost in the devastation that occurred when the World Trade Center collapsed in the attacks of September 11 [Hoard 2001]. However, no significant amounts of data were destroyed because these (mostly financial services) companies had the foresight to maintain backups remotely [Hoard 2001].

In the aftermath of this tragedy, disaster recovery services used storage networks to restore thousands of terabytes of business data and get hundreds of companies running [Noblett 2001]. In the wake of this previously unimaginable disaster, storage providers like StorageTek and EMC experienced increased customer interest in storage networks that include complex business continuance capabilities. One service provider claimed that the request for business continuance services increased 1,000% since September 11 [Noblett 2001].

In a world with anything but certainty, storage system administrators must prepare for any catastrophe, imaginable or not. For disaster recovery, the most prominent approaches have been tape backup, redundancy and snapshot technologies.

#### **5.2 Backup and Archiving Techniques for Recovery**

The purpose of backing up data is to protect from data loss or corruption due to user errors or hardware failures. Archiving data is a process to create a complete, self-consistent

replica of a collection of data, typically a directory hierarchy, appropriate for bringing back online in the future. Therefore backups protect from errors and archives allow business to stop and resume work (possibly at different locations) [Brown 2001].

Restoration from backup most commonly entails the recovery of a single file or directory contained within the backup (complete file system restores are rarely necessary) [Brown 2001]. In contrast, archives are usually restored in their entirety. Disaster recovery is a closely related subject that shares characteristics with both backup and archiving [Brown 2001]. Like backup, the intent is to protect from errors, but disaster recovery concentrates only on catastrophic failures. Where backup is concerned with users accidentally destroying files or individual hardware components failing, disaster recovery addresses natural disasters wiping out an entire building or geography. This scale of concern makes disaster recovery similar to archiving, in that the data saved must be self-consistent and restorable in its entirety to allow work to continue at the recovery site [Brown 2001].

Backups allow individual files or entire file systems to be restored after a hardware failure or user error. Archives allow entire collections of data to be resumed after a disaster. Disaster recovery allows businesses to resume operations at a remote site after a major disaster. There is a hierarchical relationship: copies of backup tapes may be sufficient to archive a project, and sufficient (offsite) archive tapes may be enough to recover from a disaster [Brown 2001].

### **5.2.1 Tape backups**

Traditionally system and user files have been backed up onto tape cartridges and preferably the tapes should be stored offsite. However, in today's multi-terabyte environments, tape backup has become more and more impractical for organizations that generate enormous amount of data daily. The primary issue for backup and archiving is speed. Even with the latest technology, it will take more than tens of hours to backup or restore one terabyte of data. The ultimate limitation for backup speed is the backup media itself: modern tape drives are capable of only a few megabytes per second [Brown 2001]. Software and system I/O bandwidth typical limit performance even further.

More problematic than the speed issue is the issue of data integrity. Because it takes a finite amount of time to write data to tape, there is always the risk that a file being backed up while the tape is being written. If this occurs, there is no guarantee that the data on tape can even be restored meaningfully, regardless of tape speed [Brown 2001].

Modern commercial backup software packages provide a variety of mechanisms to perform safe live file system backups. It remains a fact, however, that a significant portion of the worlds IT assets are written to tape on the assumption that no application is writing to the dataset while it is being backed up [Brown 2001]. NetApp's Snapshot technology provides system administrators with a powerful tool set that virtually eliminates these problems.

With the per megabyte price of hard drives continuing to drop and its superior performance advantage over tape backups, it has been suggested to use hard drives for backup. Unfortunately hard drives are not designed to be easily plugged or unplugged. We have not seen many backup systems taking advantage of the inexpensive high-capacity disk

drives. The tape backup remains to be the most widely used method for system archiving purposes. Many sites are investigating dumping to disk devices (or disk arrays) as temporary staging areas prior to dumping to tapes [Brown 2001].

### ***5.2.2 Snapshots***

Snapshots are a series of images of the data on a file system that is stored onto a protected area of the same system [Hitz 1994]. Although a few approaches to snapshots have been proposed, it appears that the one developed by Columbia Data Systems may be the most successful [Sander 2001b]. This technology has been included in systems offered by IBM, and has been integrated under license by Microsoft in its Server Appliance Kit (SAK). For example, Maxtor's MaxAttach 4300 is based on Windows 2000 with SAK extensions.

The other popular snapshot is the on Network Appliance's filer products based on the Write Anywhere Filesystem Layout (WAFL). The WAFL file system can freeze frame itself at any point in time, and make the frozen versions of the file system available via special subdirectories that appear in the current (active) file system [Hitz 1994]. Each freeze from version of the file system is called a snapshot. Snapshots usually require only a tiny disk space premium to maintain and only start to consume disk space as the file system changes after a snapshot is created.

This facility allows users to recover accidentally damaged or deleted data. Rather than requesting an administrator to locate and mount a backup tape containing the user's file, and waiting for the tape to seek and load the file in question, the user can simply copy from a hidden snapshot directory a recent version of the file (prior to the accident). If configured

and scheduled to take snapshots regularly, it also enables users to retrieve old versions at particular date.

Because snapshots provide a consistent, read-only view of the file system, they provide an elegant solution to the problem of live file system backups. Prior to beginning the backup, a snapshot is taken (a process of a second or two). The backup may then be taken from the highest level Snapshot directory. As long as backups read from a snapshot, changes that occur to the file system while the backup is running will not affect what gets written to tape. Users always see an active file system, fully read-write at all times, the backup device sees a stable, read-only Snapshot of the file system taken an instant before the backup began. This ability to create guaranteed consistent, restorable archives at any time (especially during working hours) is a very valuable feature.

Snapshots provide a particularly convenient method for backing up relational data files. Traditionally, the only way to guarantee a consistent state for such databases prior to backup is to shutdown the application that controlled the database. Backups consist of shutting down the application, performing the backup, then restarting the application. The downtime is completely dependent on the speed of the backup, typically several minutes to several hours.

With snapshots, the downtime can be reduced to a few seconds, the time it takes to create a snapshot. The sequence becomes: shutdown the application, create a snapshot, restart the application, then dump to the backup media from the top level Snapshot directory. Such backup/archive images are guaranteed to be consistent and immediately usable to the end application. Equally importantly, restorable snapshot images of such

databases can be left online for significant periods of time. If a database ever experiences some form of corruption, the time to recover can be dramatically reduced.

The snapshot technology is particularly useful for user data backup. However, snapshot alone does not improve the reliability of physical storage systems. If the snapshot image resides on a server, the server or drive fails, recovery from a snapshot is impossible, because the device that holds the snapshot is not available. Therefore, the concept of mirroring an entire data volume over a network is a more reliable approach to overcome disasters.

### **5.2.2 Redundancy**

RAID (Redundant Array of Independent Disks) [Patterson 1988] is one of the most important and most successful inventions in storage technology. The success of redundancy in *system area networks* (SANs) has been extended to *wide area networks* (WANs). Data centers are thriving to provide storage-mirroring services between remote sites hundreds or thousands of miles away. Storage systems can be mirrored using point-to-point fibre connections or through WANs or *Virtual Private Networks* (VPNs) connections. With the imminence of multiple gigabits Ethernet on the horizon, the concept of data redundancy between geographically separated locations is gaining more popularity in IT market.

In this study we will construct a reliable and highly available storage system using CoStore clusters. To the best of our knowledge there is little literature available discussing the performance of mirroring storage between remote sites. There are several products from major storage vendors, for example, the Symmetrix Remote Data Facility (SRDF) from EMC [EMC 2001], and the SnapMirror from Network Appliance [Brown 2001]. Both of

them are mirroring between point-to-point storage volumes. A CoStore cluster with mirroring can consist of one pair or multiple pairs.

### **5.3 Reliable and Highly Available Storage Systems Using CoStore Clusters**

In information technology, high availability refers to a system or component that is continuously operational for a desirably long length of time. Reliability is an attribute of any computer-related component that consistently performs according to its specifications. The RAID approach has been ubiquitously utilized to improve reliability and high availability in storage systems. The success of RAID is mainly driven by its data striping technique, which achieves high performance through the parallelism of multiple disks and fault tolerance by the extra check data parity.

The redundancy in RAID arrays has enabled to overcome component (disk) failures. In the event of a disk failure, a RAID array can still serve in degraded mode while automatic reconstruction takes place on a hot-spare disk in background. Level 1 (mirror) and level 5, along with variations (such as 0+1) are the most commonly used in RAID configurations. To add more availability, a pair of clustered servers can be connected to the same disk array as failover partners. They can be configured as master/slave, or master/master each sharing half of the service load. In the event of one server failure, the other one will take over the whole responsibility. This is the traditional cluster of two standby failover hosts.

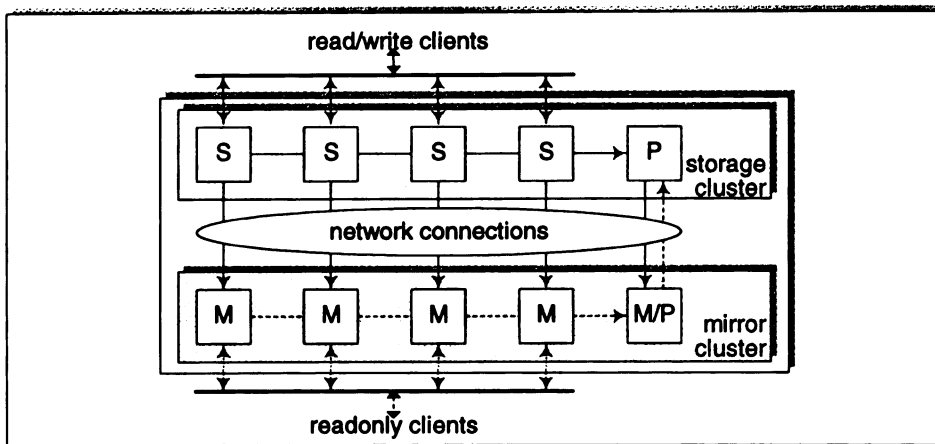


Figure 5-1 A CoStore cluster with RAID level 4+1

In this study we use cluster in a broader sense. A CoStore cluster can consist of a single computer or a group of computers, each equipped with its own block devices. The same RAID semantics of various levels is maintained in a distributed manner. Borrowing the concept from RAID variations levels (0+1, 4+1 and 5+1), we introduce a second level redundancy by mirroring the original storage cluster. Figure 5-1 shows one example cluster with redundancy equivalent to RAID level 4+1. The main cluster is called the storage cluster, which handles read-write client requests. The mirror cluster stores a mirror copy of the storage cluster and can handle requests from read-only clients. Similar to the roles of disks in RAID arrays, daemons can assume different roles in CoStore clusters: storage daemons (S), parity daemons (P), mirror daemons (M), and possibly hot-spare daemons (H).

### 5.3.1 Benefits and implications

Just as read performance is doubled in RAID 1 disk arrays, the mirroring of clusters can also improve performance in addition to system availability. In a LAN environment, members in a mirror cluster can be configured as standby failover partners and at the same

time they can also serve read requests. When a member in storage cluster has failed, the corresponding member in mirror cluster can upgrade itself into storage cluster and take the place of failed member to avoid time- and bandwidth-consuming reconstruction. Without expensive reconstruction, storage cluster can also avoid switching into the degraded mode, which has a seriously reduced performance. Therefore this second level redundancy improves availability as well as read performance.

Following the trend of extending RAID's success from SANs to WANs, we move the mirror cluster away from the storage cluster with a range of distances: to different rooms, to different buildings in the same campus, or to campuses in different geographical regions. Mirroring between remote sites is particularly beneficial and desirable in certain situation. For example, in a business environment, some information needs to be deployed at remote sites for read-only applications. In this way, a full-time online almost up-to-date file system is being replicated. The distribution of data is very efficient because requests from remote clients are served by the local mirror cluster at remote sites.

Another benefit of mirroring clusters between remote sites is improved disaster recoverability. Disasters have various scopes in terms of geographical sizes, ranging from rooms, floors, buildings, campuses, cities, and to regions. The longer the distance between remote sites, the more likely at least one of the mirrored clusters is able to survive a major catastrophe. The success of data centers should mainly attribute to their replication services between geographical regions so as to provide maximum survivability to customers' storage systems.

Compared with a traditional point-to-point clustered pair, a multi-entity CoStore cluster with mirroring has several advantages: larger volume size, higher aggregated network

bandwidth and performance. Due to the number of active members serving in a cluster, there are multiple members in a cluster each with local disk resources and there exist multiple network connections between storage and mirror clusters. In this study we will evaluate the impact of network bandwidth and latency on the performance of CoStore cluster with mirroring.

Using queuing theory the theoretical reliability of RAID arrays could be quantified [Patterson 1988]. As we have analyzed in Chapter 4, the reliability of some of the RAID variations with two levels of redundancy, level 4+1 or 5+1 in particular, can provide extremely high reliability, compared to traditional RAID levels. However, this high reliability comes at a price in increasing overhead, i.e. more disk space used for mirror copies; and reduced performance because mirror data have to be copied. In fact, the space overhead in RAID 4+1/5+1 is so high that in extreme cases only 1/3 of the total disk space is usable and that the redundancy overhead is always more than 50%.

Transparent to upper layers, data redundancy management is actually implemented in a standalone block buffer cache subsystem, which provides a block interface to local file subsystem. With a standalone buffer cache system, multiple tiny updates can be bundled together and sent in relatively fewer network operations and hence improved overall performance.

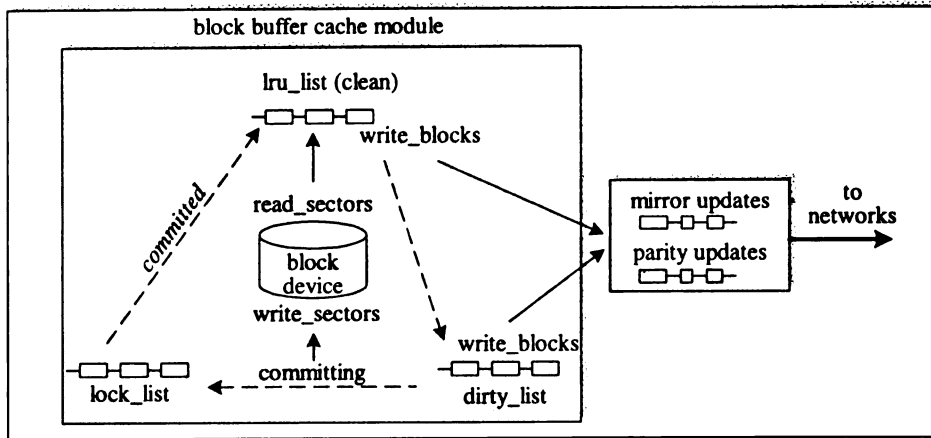


Figure 5-2 RAID buffers and block buffer cache in CoStore

## 5.4 Experimental Setup

The tests are conducted on a cluster of PCs, each equipped with 1.2GHz Pentium III processor, 256MB SDRAM memory, 100Mbps network interface, and 20GB hard disks (ATA100/5400RPM/2MB cache). All of them are running Windows 2000 Professional. Locally all PCs are connected by 100Mbps Fast Ethernet switches.

To measure the impact of mirroring distance on the performance of CoStore, we conduct the experiments such that the mirror cluster is located in a series of different network environments. These environments include *A)* a single switched Ethernet LAN; *B)* two LANs connected by 100Mbps router/switches within one department; *C)* two LANs connected by a Gigabit building router; *D)* two LANs connected by multiple Gigabit routers on a Gigabit Ethernet campus backbone ring; *E)* two LANs in two universities connected by the Internet with a physical distance of about 140km; *F)* two LANs connected by an IP emulator with a latency of 30ms. (The latency through Internet between East and West coast of the United States is about 60ms).

**Table 5-1 Test sets and the characteristics of network environments**

Test sets	Network environments	TCP throughput (Mbps)	ping latency (ms)
<i>A</i>	switched LAN	94.72	0.18
<i>B</i>	cross switched LANs	94.72	0.20
<i>C</i>	cross building router	94.38	0.30
<i>D</i>	cross campus backbone	94.35	0.40
<i>E</i>	cross Internet	10.68	10.0
<i>F</i>	cross WAN emulator	32.79	30.0

The network bandwidth (or TCP throughput) and latency for each of the testing environments is shown in Table 5-1. The TCP throughput is measured by Test TCP (TTCP) [TTCP]. The latency is measured by the utility program *ping*. Figure 5-3 The network diagram of testing environments shows the network diagram of the testing environments and interested readers can refer to campus network diagrams for MSU and WSU from [MSU-network 2001; WSU-network 2001] and MichNet backbone diagram from [MichNet-Backbone 2001].

CoStore clients and the storage cluster are always on the same Ethernet switch of LAN A. Test Set A is the performance baseline as the mirror cluster is on the same switch as well. In Test Set B, we move the mirror cluster onto LAN B. The two LANs are connected by a department network backbone: Cisco 7200 router and Catalyst 5500 switch in the Department of Computer Science and Engineering (CSE) at Michigan State University (MSU). The closely coupled router and switch are configured to ensure that only the first packet goes through router and that all subsequent packets will be switched following the path of the first packet. In Test Set C, the mirror cluster is located on a farther LAN C and the network traffic has to go through the Engineering Building router. This Gigabit Ethernet Foundry router is one node on MSU's Gigabit Ethernet campus backbone ring. In

Test Set *D*, the mirror cluster is moved onto LAN *D* in National Superconducting Cyclotron Lab (NSCL) building (a quarter mile away) at MSU. Both buildings are connected to the Gigabit Ethernet campus backbone via Foundry routers, with five routers in between.

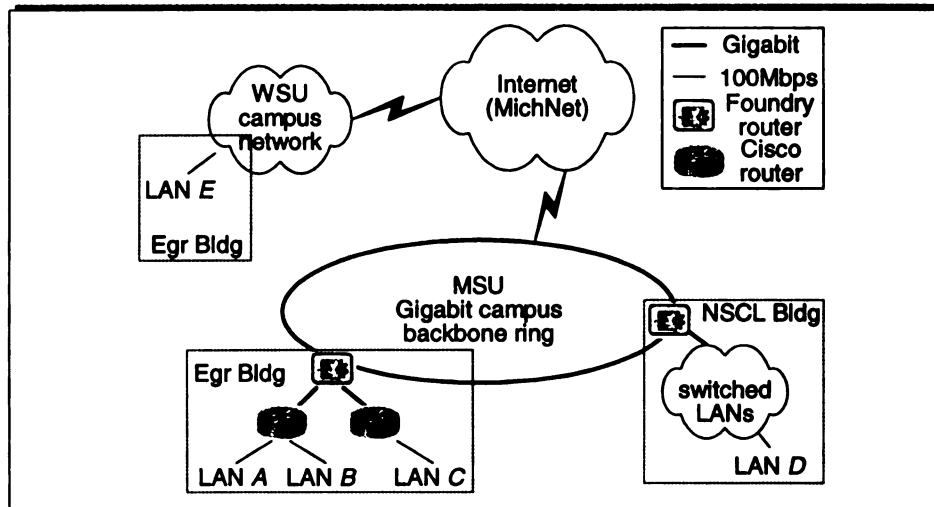


Figure 5-3 The network diagram of testing environments

In Test Set *E*, the mirror cluster is placed on LAN *E* in the Department of Electrical and Computer Engineering at Wayne State University (WSU), which is located in another metropolitan area. Both universities are connected to the Internet2 via MichNet [MichNet-Backbone 2001].

#### 5.4.1 Network emulator EMIP

Unable to further expand the distance, in Test Set *F* we resort to network emulator to evaluate the performance of CoStore with the mirror cluster located far away from the

storage cluster. The network emulator in Test Set *F* is EMIP (Emulation of IP Networks) [Huang 1998].

The EMIP emulator provides the functionality of applying a set of network conditions and traffic dynamics to the traversing network traffic. EMIP, in contrast to a pure software simulation solution like ns-2 [ns-2], is a real laboratory network on which real protocol and application implementations can be developed and tested both for correctness and for efficiency against a selected set of real network conditions and traffic dynamics.

In EMIP, each physical network interface is attached with a virtual device module in Linux kernel (Figure 5-4). The virtual device module consists of five policy units: MTU (minimum transfer unit) unit, delay unit, bandwidth unit, loss unit and bit error unit. By intercepting outgoing network traffic to be sent out from one of the emulator's network interface, the virtual device module can impose bandwidth control, delay control, packet loss rate control, bit error rate control or MTU changes for each outgoing packet sending out from that network interface.

In addition, background traffic can be generated within the emulator host such that the impact of the background traffic to the emulation experiment can be studied. EMIP also supports trace-driven emulation, in which a trace file of real network traffic is used as the input to the emulator instead of specific protocols and applications. The virtual device module is implemented as a kernel loadable module on Linux. A GUI interface is also provides to easily configure network conditions, traffic dynamics, and background traffic to be applied. Useful traffic statistics can be viewed on the GUI interface. A project called EMPOWER is currently underway to design and implement a more scalable network emulation framework [EMPOWER].

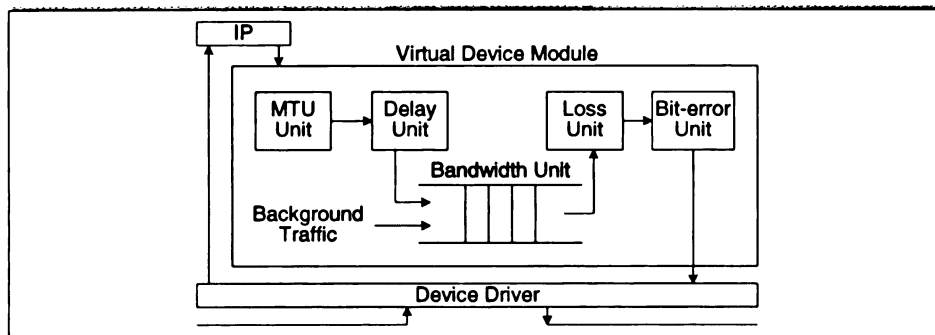


Figure 5-4 The internals of EMIP

In Test Set *F*, the EMIP emulator acts as a router between the storage cluster and the mirror cluster and the virtual device module of EMIP is used as a traffic shaper to delay traversing network traffic through the emulator host. The delay is set to 30ms in either direction and the measured TCP throughput is 32.8Mbps using TTCP.

For all the tests we only evaluate write performance, which is the central topic in RAID performance. All the tests are a set of consecutive write requests with variable file sizes. We arrange the tests into two groups according to file sizes: 32KB to 512KB for small-size and 1MB to 128MB for large-size (For clearer curves not all large samples are shown in some of the figures.) Each file size has 4 different files and an average is taken as the latency for that size. Currently the CoStore prototype only supports RAID level 0/0+1 and level 4/4+1 with various cluster sizes. In RAID 0/0+1, the group size is set to one because larger size would not change the performance of CoStore. In RAID 4/4+1, we only evaluate single-client cases. The bottleneck effect in RAID 4/4+1 with multiple clients deserves separate scrutiny in further studies.

The independent block buffer cache subsystem in CoStore is enabled in all the tests. All the disk and network I/O operations are called synchronously. There is another block

buffering at the operating system level. This extra buffering from OS can be disabled and we will investigate its effect along with that of asynchronous I/O in upcoming studies. For all tests in this study, the block buffering at the operating system level is left on and the only negative effect it might cause is the use of extra memory, which is abundant on the test PCs.

## **5.5 Performance Evaluation**

There are two options in deciding when to commit modifications to devices or to mirror or parity daemons. One choice is to commit the modification for every client request before sending out the reply. This policy is called transactional write (transact-Wr). The other option is to periodically commit dirty buffers, which is called lazy write (lazy-Wr).

For parity and mirror updates, when the buffer is full, the system will force the flushing before more updates can be inserted into buffers. Therefore, the timing of committing parity and mirror is controlled by two parameters: the flushing interval and the capacity of buffer size. We will demonstrate the effect of transact-Wr in Test Set A. For all other tests we use lazy-Wr only; we reserve 32MB for block buffer and 1MB for either mirror or parity update buffer and the buffer committing period is 5 seconds.

### ***5.5.1 Local mirroring***

Figure 5-5 shows the results for RAID 0/0+1 in a switched Ethernet environment (Test Set A). For both small and large size accesses, the transactional commit policy considerably lowered the performance of either RAID level 0 or level 0+1. The overhead of copying updates to mirrors is relatively small compared to the total latency for each access.

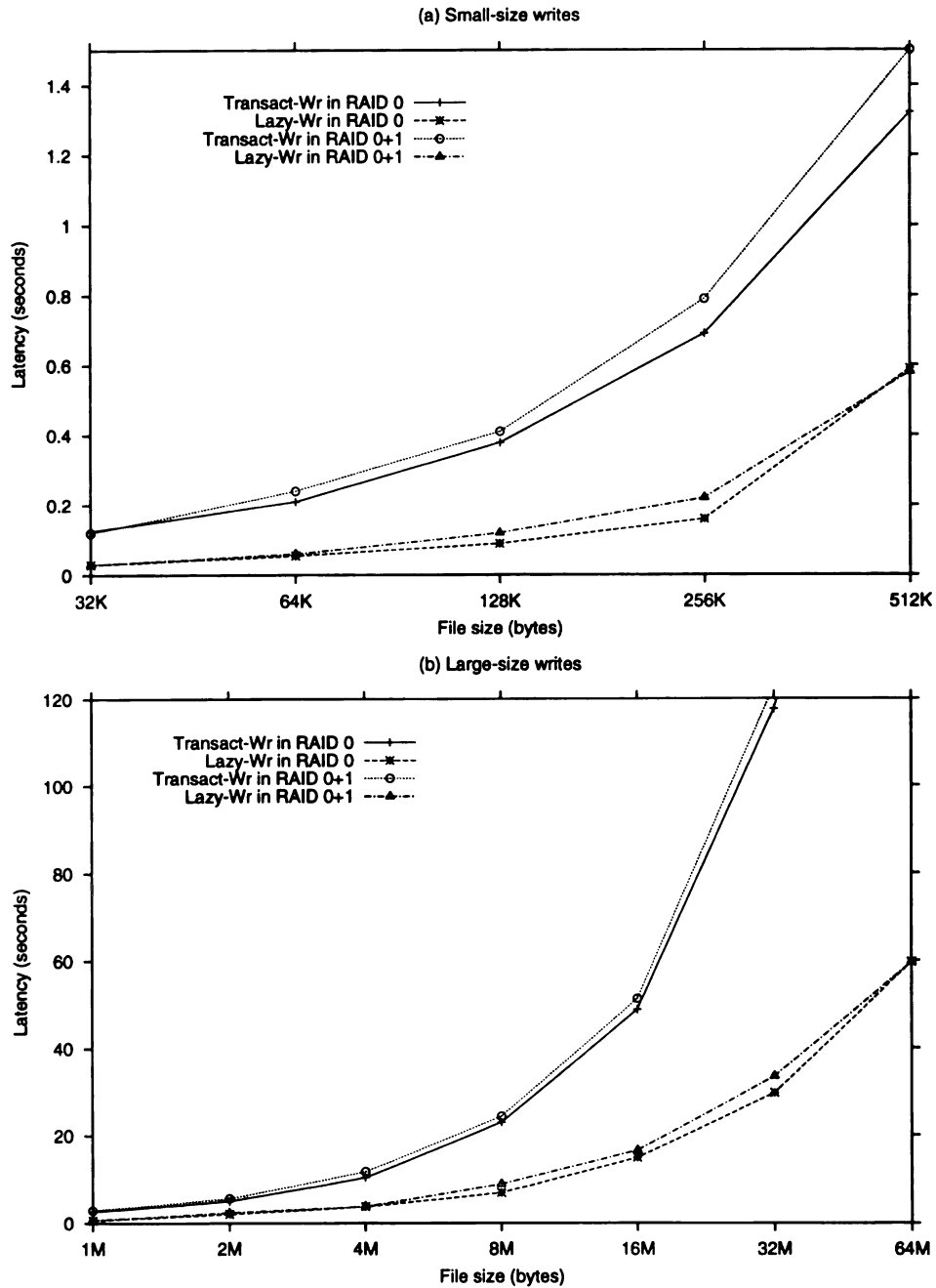


Figure 5-5 A CoStore cluster with RAID level 0/0+1 in a switched LAN

Figure 5-6 shows the results for RAID 4/4+1 with 4 storage daemons (Test Set A). Again, just as in RAID 0/0+1, the transactional commit significantly lowered the performance of RAID 4. Due to the large buffers, the performance for small writes in RAID 4 or 4+1 are indistinguishable in Figure 5-6(a). However, for large size accesses RAID 4+1 is slower than RAID 4 because of the extra mirroring effort. For example, for a 64MB file, the writing in RAID 4+1 is about 15% slower than RAID 4.

Special techniques can be applied to reduce the overhead in RAID 4+1. In the implementation, we have an option about how parity and mirror buffers are updated. A default method called S-Both (solid lines in Figure 5-1) is to let storage daemons update both buffers: to the parity daemon and to its corresponding mirror daemon. Alternatively, storage daemons can choose to only update mirror daemons. Upon receiving the updates from storage daemons, the mirror daemons then notify the changes to the parity daemon in the mirror cluster, which in turn will update the parity daemon in the storage cluster. This option is called M-Parity (dotted lines in Figure 5-1). In other words, the parity updates are postponed until the parity daemon in the mirror cluster is updated. From Figure 5-6(b), the M-Parity brings the RAID 4+1 performance very close to that of RAID 4, however still about 2.4% slower.

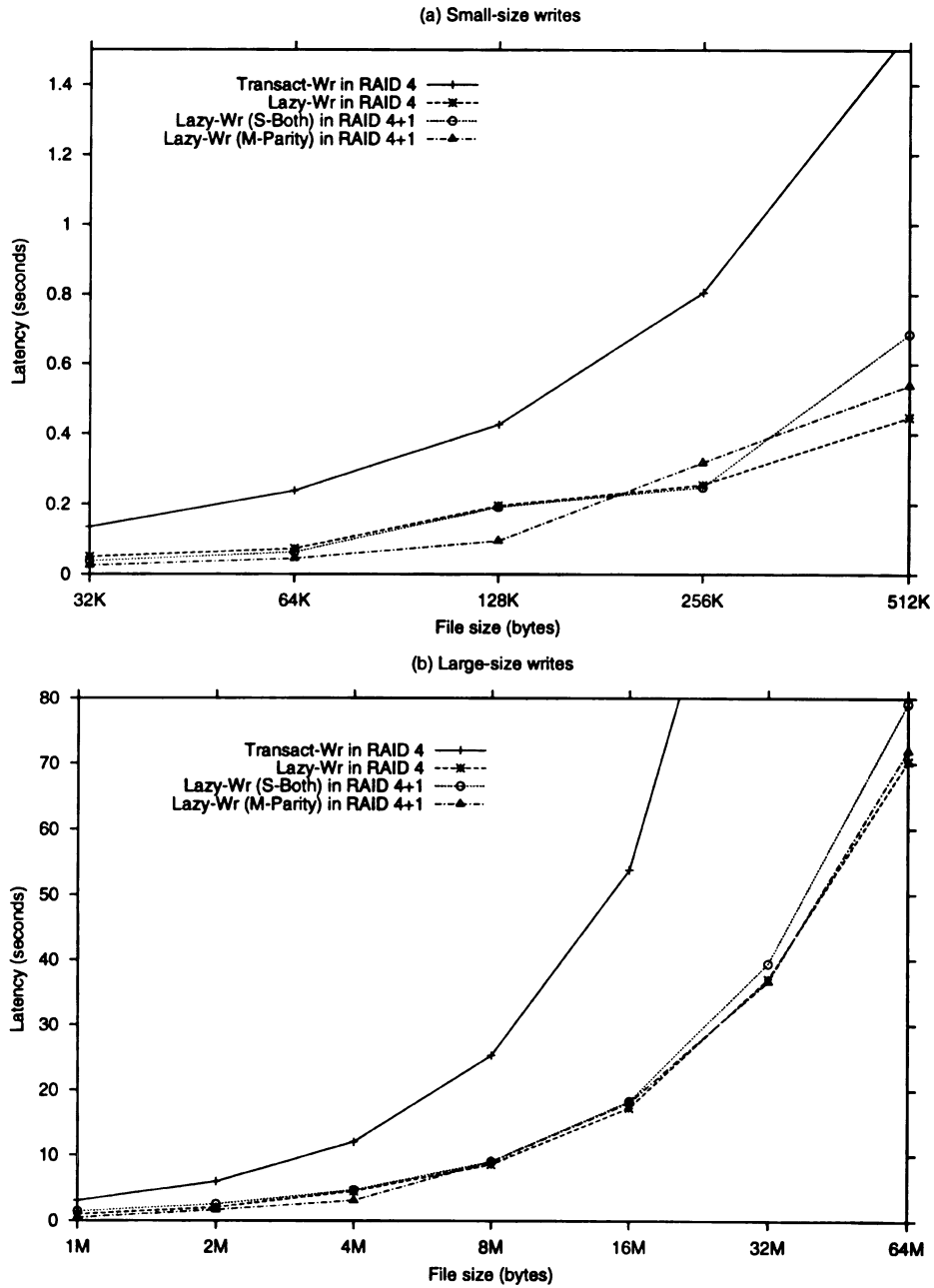


Figure 5-6 A CoStore cluster with RAID level 4/4+1 in a switched LAN

Comparing RAID 4+1 (Figure 5-6(b)) with RAID 0+1 (Figure 5-5(b)), we can find that the extra parity has an overhead of about 31.6% using standard S-Both option, or about 12% if using the special M-Parity option. For extra redundancy, this may be a small performance price to pay. The M-Parity technique effectively removes the parity bottleneck in storage clusters and makes parity updates out of storage daemon's critical path in serving the front-end clients request traffic. The benefits of M-Parity remain to be further investigated when there are multiple concurrent clients.

### *5.5.2 Remote mirroring*

We evaluate the effect of CoStore clusters mirrored in a switched Ethernet LAN environment in Test Set A. Next, we conduct similar tests on RAID 0+1 and 4+1 on different network environments with varying network bandwidth and latency (distance). All the following tests are conducted with option lazy-Wr and S-Both.

From Figure 5-7(a) and Figure 5-7 (b) we can find that both the performance of RAID 0+1 and RAID 4+1 clusters are virtually unaffected by the fact that the mirror clusters were located on different subnets, ranging from in the same room, to different wings in same building, and finally to different buildings in the same campus. The high bandwidth gigabit campus backbone and the high performance routers/switches make possible mirroring storage clusters located as far as a campus networks can reach. With such distances, the storage systems can survive most natural disasters or even unexpected catastrophes like the September 11 attacks. The network infrastructure in MSU is state-of-the-art, but by no means rare. Many large universities, business organizations have such infrastructures in place. Many of the data centers have better network resources at their disposal.

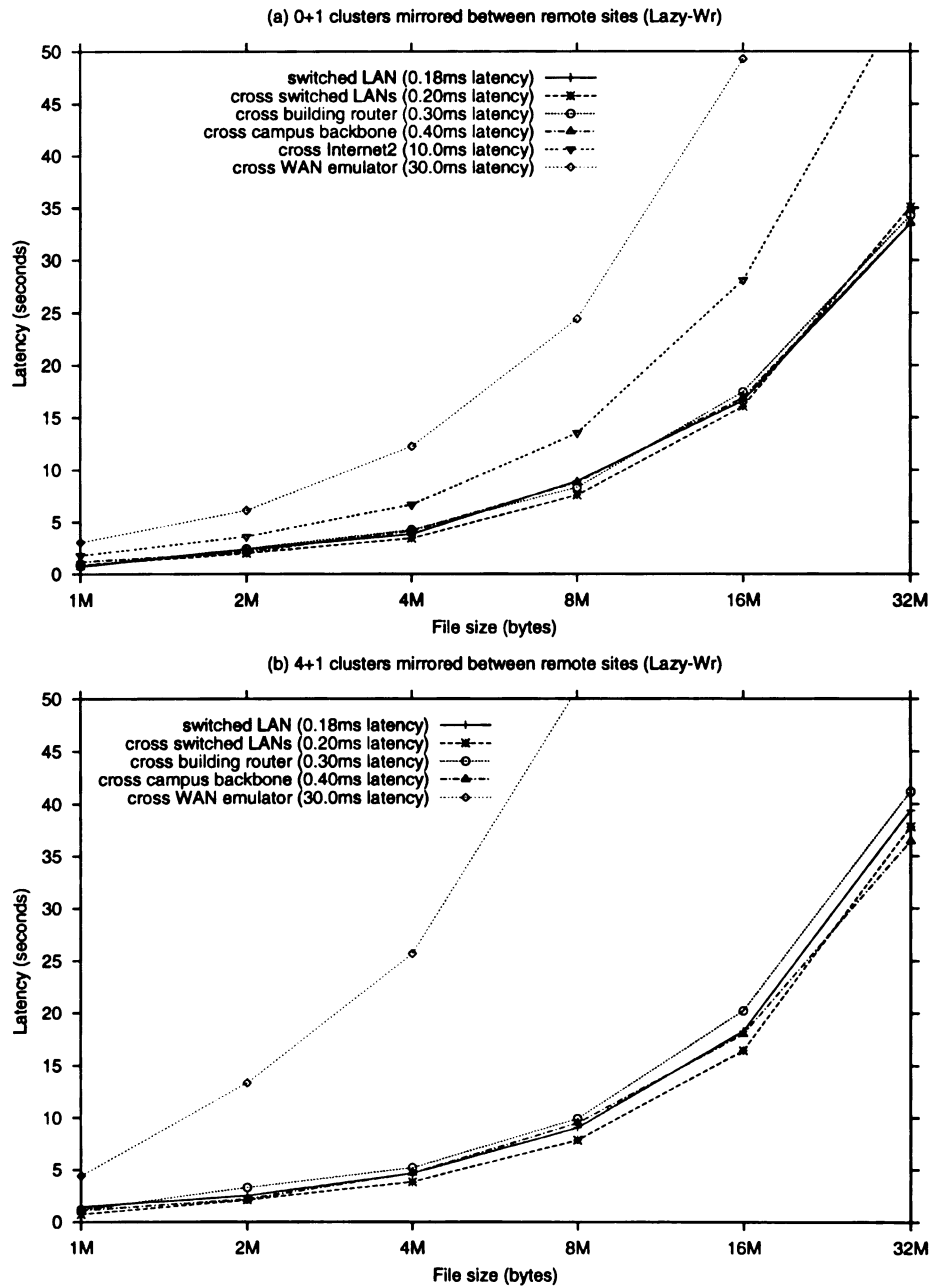


Figure 5-7 A CoStore cluster mirrored in various network environments

In Test Set *E*, we measure the performance of RAID 0+1 between two university campus networks. The baseline throughput of CoStore cluster with RAID 0+1 is roughly 1MB/s in Test Set A (Figure 5-5(b)). Even though the available bandwidth (10Mbps) from the connection is adequate, the throughput of the mirrored CoStore cluster is slowed to 0.57MB/s, a 43% drop, by the 10 ms latency along the 10-hop path through the Internet connection. In the emulated IP network with 30ms latency, the throughput has dropped to 0.32MB, or 68% off from the baseline performance. Again the network bandwidth (32.79Mbps) is more than enough and it is the much longer delay along the mirroring path that has slowed the performance of mirrored CoStore clusters. Due to the physical long distance and limited time, we choose not to conduct tests with RAID 4+1 in Test Set *E*.

In Figure 5-7(b) the Test Set *F* shows that the performance of RAID 4+1 is doubly penalized (0.16MB or 84% off) by the long delays because both data mirrors and parity mirrors have to be updated crossing the long-delayed emulated WAN connections. This 30ms latency is equivalent of going through 10+ Internet backbone routers with a physical distance of up to 3000km. This latency is drastically too big for most commercial organizations (such as data centers). However most business can afford direct links or only-a-few-hops links with very low latency and high bandwidth. At the meantime the longer the distance separating the mirrored clusters, the more likely one cluster can survive large-scale disasters.

### ***5.5.3 Performance enhancing techniques***

By having larger block buffers or longer commit interval, the system performance can be improved. However, we risk more data loss if storage daemons crash before updates are

committed to disk or parity/mirror daemons. The availability of NVRAM (Non-volatile RAM) certainly can help; unfortunately NVRAM is very expensive and only high-end systems can afford.

So far the daemon in CoStore is only implemented as a single threaded program. It makes perfect sense to introduce a second thread to take care the synchronization of RAID updates (parity or mirror) in background so that the main thread can be more responsive to clients' requests. Another potential technique to improve performance is to use asynchronous I/O operations for disk and network accesses. By issuing an I/O request asynchronously, daemons can return right away to handle more client requests without waiting for the request to be completed.

As of this writing we have just finished the implementation of asynchronous disk and network I/O. Unfortunately there is not enough time for further experiments or to implement multi-threaded daemons. We plan to evaluate the effect of multithreading and asynchronous I/O on the performance of mirrored CoStore clusters in future study.

## **5.5 Summary**

The CoStore cluster architecture could construct storage systems with strong reliability and high availability. The performance evaluation of the CoStore prototype demonstrates that there is little impact on performance even if the cluster is mirrored in different buildings on a large campus as long as the network bandwidth and latency are satisfactory.

The test results also show that the latency between remote sites plays an important role in the performance of the cluster even if the bandwidth is adequate. With direct communication channels or efficient VPN links, the latency could be kept low even if the

distance is long. Therefore, with dedicated networks the CoStore cluster architecture could effectively improve storage systems' preparedness for disaster recovery without sacrificing performance. However, direct deployment on public Internet may be less desirable.

## Chapter 6

### Conclusions and Future Work

#### 6.1 Related Work

##### *6.1.1 Storage cluster architecture*

Gibson et al. have built a high performance storage system based on directly network-attached secure disks (NASD) [Gibson 1996]. With a similar architecture, the CoStore is also proposed to take advantage of the emerging network attached storage (NAS) devices. CoStore currently assumes that clients authenticate through a central authentication server while the NASD system uses a dedicated file manager to enforce access control. More importantly in CoStore the namespace consistency is maintained collaboratively by participating cluster members, different from NASD whose central file manager maintains the namespace consistency for the whole system.

Previously proposed serverless distributed file system xFS [Anderson 1995b] is a file cluster server supporting high-performance distributed computing applications on network of workstations (NOW) [Anderson 1995b]. The xFS file system inherits the log-structured file system from LFS [Rosenblum 1992.] and the distributed RAID striping from Zebra [Hartman 1993] to provide data redundancy.

Compaq/DEC's research project Patel [Lee 1996] provides a distributed storage system, offering a block-level virtual disk interface. Block-level interface rather than a richer file-level interface is chosen to handle heterogeneous client file systems. The authors of Patel were concentrated on replication-based redundancy schemes like chained-declustering [Hsiao 1989]. Patel logically could be viewed as a RAID system implemented

on a symmetric multiprocessor [Gibson 2000]. Similarly, the architecture in our design is virtually a directory-based approach in distributed shared-memory multiprocessor system. Frangipani [Thekkath 1997] is a distributed file system built separately on top of Patel's virtual disk.

Distributed RAID approach was introduced in [Stonebraker 1990] and was used in xFS [Anderson 1995b] and Petal [Lee 1996] to build virtual disks with block interface. On top virtual block devices, higher level file systems and distributed file systems were built, like the metadata manager in xFS and Frangipani [Thekkath 1997]. Differently, CoStore offers a file interface using NAS approach, which seamlessly integrates local file system and data redundancy management. Both xFS and CoStore are a serverless design. However, CoStore evenly distributes metadata and the management across all storage devices in the cluster. Metadata in CoStore systems are stored at fixed locations while data can be stored anywhere, compared with the Anything, Anywhere file layout in xFS.

We borrow the concept of A Frequently RAID from [Savage 1996]. The flexibility of parity-update-delay parameter ranges from strictly full-time redundancy to variable-delayed frequent redundancy. This makes possible the idea of configurable tradeoff between reliability and performance. LFS technique is mainly used to improve small-write efficiency, which is critical for overall performance in typical RAID systems. For simplicity reason our design will not employ the LFS technique, but we will use the journaling concept for fast recovery after system crash.

The CoStore design is also influenced by Network Appliance's design in their storage appliance products [Hitz 1994]. We use the concept of storage appliance to integrate RAID management and file system management functions. Similarly, RAID 4 and RAID 0+1 are

first chosen for the ease to expand the volume capacity and the simplicity in implementation.

### ***6.1.2 Remote storage replication***

The most relevant work is Microsoft Research's Farsite project [Microsoft 2000], which builds a serverless distributed file system with a large group of collaborative desktop computers across campus-wide corporate network. The fault-tolerance of Farsite is achieved through replication of files assisted by Byzantine-fault-tolerant algorithm [Douceur 2001]. Compared with our proposal, this higher-level approach works well in an environment of mostly with seat owners. Their previous study demonstrated the feasibility of a serverless distributed file system based on existing desktop PCs at Microsoft Corporation [Bolosky 2000].

Farsite [Douceur 2001], PAST [Druschel 2001] and CFS [Dabek 2001] are all peer-to-peer approach based storage systems. OceanStore [Kubiatowicz; Rhea] is server-to-server based. Farsite and PAST are replication based data archiving systems. CoStore uses distributed RAID, a specific form of the erasure-resilient coded [Bloemer 1995] fragments (or block) based approach taken by OceanStore and CFS [Dabek 2001]. Both Farsite and OceanStore exploit Byzantine agreement protocol [Castro 1999] to realize fault tolerance and high availability. PAST and CFS only serve as read-only archival systems and are mainly used for file-sharing purposes like the first generation peer-to-peer systems. Differently, Farsite, OceanStore and our proposal all provide traditional filesystem semantics. PAST, OceanStore and CFS are designed for global scale deployment, while Farsite and our proposal are targeted for organizational scale.

Both using distributed RAID, xFS [Anderson 1995b] and CoStore shares their decentralized architecture with peer-to-peer systems. However xFS is only intended for general-purpose filesystem in a LAN environment and CoStore was targeted at departmental level in organizations.

The efficiency of locating data items in large-scale peer-to-peer systems is critical. In this regard several routing and location schemes have been designed, including Pastry [Rowstron 2001] in PAST, Tapestry [Zhao 2001] in OceanStore and Chord [Stoica] in CFS. Farsite uses a distributed directory service to keep track of replica and file version locations.

### ***6.1.3 Storage service utilizing idle space***

EMC's SRDF [EMC 2001] and Network Appliance's SnapMirror [Brown 2001] are examples of point-to-point mirroring between remote storage servers. Typical CoStore systems are clusters with a group storage hosts with various degrees of redundancy, possibly with two levels of redundancy.

## **6.2 Conclusions**

In this dissertation study, we have proposed a cluster architecture to construct storage systems and have demonstrated the CoStore has the potential to achieve: higher scalability due to the effective file system layout; high performance because of its root in the efficient NFS protocol; high reliability by using distributed RAID; and high capacity by pooling multiple devices together.

### ***6.2.1 Storage cluster architecture***

Specifically, the NAS approach has been proven to be effective in constructing scalable storage server clusters using network attached storage devices. The key to CoStore's efficiency and scalability is the flexible Data Anywhere, Metadata at Fixed Locations file system layout. In CoStore the consistency of a unified file namespace is maintained by all cluster members collectively without any external assistance. The serverless design eliminates any central server bottleneck and provides strong scalability.

The CoStore prototype using COTS-based components demonstrated feasibility of building such scalable storage clusters. The performance results measured on the prototype confirm the potential of CoStore to achieve scalable high-performance high-capacity storage services with strong reliability and availability.

### ***6.2.2 Remote storage replication***

The CoStore cluster architecture could construct storage systems with strong reliability and high availability. The performance evaluation of the CoStore prototype demonstrates that there is little impact on performance even if the cluster is mirrored in different buildings on a large campus as long as the network bandwidth and latency are satisfactory.

With dedicated networks the CoStore cluster architecture could effectively improve storage systems' preparedness for disaster recovery without sacrificing performance. However, direct deployment on public Internet may be less desirable.

### ***6.2.3 Storage cluster utilizing idle space***

Using an NAS approach, CoStore is a novel solution to build reliable storage service with efficiency, utilizing idle disk space on desktop workstations. System uptime data collected from production systems confirmed the viability of constructing storage cluster systems utilizing idle disk space on desktop computing infrastructure.

## **6.3 Contributions**

We make several contributions in this dissertation study. Four main contributions are:

First, we adopted the contemporary NAS approach in constructing a storage cluster. Distributed file system, local file system and RAID management have been seamlessly integrated in each cluster member to achieve efficiency in a CoStore cluster.

Second, we designed the Data Anywhere and Metadata at Fixed Locations file system layout using a simple but effective block and inode addressing scheme. This flexible layout is the key to achieve the efficiency and scalability in a CoStore cluster and its serverless feature.

Third, we investigated cluster mirroring to enable remote replication and to provide an online almost up-to-date backup copy. This extra redundancy is invaluable in disaster recovery planning.

Finally, we actually implemented a prototype CoStore cluster and demonstrated its advantages. This prototype implementation can also serve as a concept proof tool or test-bed for further research and experiments.

Other contributions of this research include: we applied the directory-based CC-NUMA approach from distributed shared-memory systems to the architecture of storage cluster

systems; we assessed the feasibility deploying the proposed cluster architecture on existing desktop-computing environments.

## **6.4 Future Work**

### ***6.4.1 Implementations***

Future work includes the implementation of the RAID management module with full support for different RAID levels, particularly the more balanced level 5/5+1. With only a client simulator, the evaluation of the current prototype is only limited to synthetic test sets. We will implement an installable module at client end so that experimenting with comprehensive benchmark programs is possible.

To address the heterogeneity of potential platform, the CoStore prototype is implemented at user level and can be relatively easy to port to other platforms. To reach more platforms, like the open source Linux and BSD, the effort to port the prototype is currently on going.

We will implement a second thread to manage the synchronization of RAID updates (parity or mirror) in the distributed RAID. We will investigate the effect of asynchronous I/O operations for network and disk accesses on the performance of mirroring CoStore clusters. Further study topics also include the investigation of importing more sophisticated security mechanism and journaling file system management into CoStore systems.

### ***6.4.2 Future research areas***

In the future we will look into DAFS and NFS version 4 and investigate the potential of DAFS-enabled CoStore in taking advantage of modern memory-to-memory networking

technologies. Another interesting area is the Quality Of Service (QoS) on storage. Storage out-sourcing has gradually becoming standard business conduct.

However, there has been a lack of mutual-agreeable and enforceable terms in defining the kind of storage service a customer is paying for in service contracts between data centers and customers. Storage services nowadays are mostly best effort. We believe that the CoStore cluster architecture has the potential to facilitate the fulfillment and enforcement of QoS Storage contractual agreements and will explore in future studies.

## Bibliography

- Alvarez, G., K. Keeton, A. Merchant, E. Riedel, and J. Wilkes, "Tutorial on Storage Systems Management," in *SIGMETRICS*, 2000.
- Anderson, D., *Universal Serial Bus System Architecture*, Addison-Wesley, 1997.
- Anderson, T., D. Culler, et al., "A case for NOW (Networks of workstations)," in *IEEE Micro*, 1995a, pp. 54-64.
- Anderson, T., M. Dahlin, et al., "Serverless Network File Systems," *ACM Transactions on Computer Systems (TOSC)*, 1995b.
- ANSI, "Small Computer System Interface-2 (X3T9.2)," Draft Revision 10k ed: American National Standard Institute, Inc., 1993.
- Bach, M., *The Design of the UNIX Operating System*, Prentice Hall, 1986.
- Beck, M., et al, *Linux Kernel Internals*, 2nd Ed. ed. Addison Wesley Longman, 1998.
- Bever, M., et al., "Distributed Systems, OSF DCE, and Beyond," in *DCE-The OSF Distributed Computing Environment*, Schill, A., Ed. Berlin: Springer-Verlag, 1993.
- Bloemer, J., et al., "An XOR-Based Erasure-Resilient Coding Scheme," The Int'l Computer Science Inst., Berkeley, Calif. tech. report TR-95-048, 1995.
- Bolosky, W. J., J. R. Douceur, et al., "Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs," in *Proceedings of the 2000 International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2000.
- Breuer, P. T., et al., "The Network Block Device," in *Linux Journal Magazine*, 2000.
- Brown, K., J. Katcher, R. Walters and A. Watson, "SnapMirror and SnapRestore: Advances in Snapshot Technology," Network Appliance Inc. TR-3043, 2001.
- Callaghan, B., *NFS Illustrated*, Addison Wesley, 2000.
- Callaghan, B., B. Pawlowski, and P. Staubach, "NFS Version 3 Protocol Specification," June 1995.
- Card, R., T. Ts'o, and S. Tweedie, "Design and Implementation of the Second Extended File system," in *Proceedings of the First Dutch International Symposium on Linux*, 1986.

- Castro, M., and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proc. Usenix Symp. Operating Systems Design and Implementation (OSDI)*, 1999.
- Chen, P., E. Lee, G. Gibson, R. Katz, and D. Patterson, "RAID: High-Performance, Reliable Secondary Storage," in *ACM Computing Surveys*, 1994, pp. 145-185.
- Chudnow, C., "Christmas Comes Early for IP Storage," in *Computer Technology Review*, 2002.
- Dabek, F., M. F. Kaashoek, D. Karger, R. Morris, "Wide-Area Cooperative Storage with CFS," in *18th ACM Symposium on Operating Systems Principles*, 2001.
- Douceur, J. R., and R. P. Wattenhofer, "Optimizing File Availability in a Secure Serverless Distributed File System," in *Proceedings of 20th IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2001.
- Douceur, J. R., and W. J. Bolosky, "A Large-Scale Study of File System Contents," in *Proceedings of the 1999 Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 1999.
- Druschel, P., and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility," in *Proc. HotOS VIII*, Elmau Germany, 2001.
- Du, D., et al., "Emerging Serial Storage Interfaces: Serial Serial Storage Architecture (SSA) and Fiber Channel - Arbitrated Loop (FC-AL)," Computer Science Department, University of Minnesota TR 96-073, 1996.
- Duplessie, S., "From DAS/NAS/SAN to 'utility-class storage'," in *InfoStor*, 2001.
- EMC, "Symmetrix Remote Data Facility (SRDF) - Connectivity Overview (Engineering White Paper)," EMC Corp. Dec. 2001.
- EMPOWER project website: <http://elans.cse.msu.edu/empower>
- Fido, I., "Review: Promise Ultra133 TX2 ATA133 Controller & Maxtor D740X ATA133 Drive," LittleWhiteDog.com Nov. 2001.
- Gibson, G., et al., "A cost-effective, high-bandwidth storage architecture," in *Proceedings of the ACM 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1998.
- Gibson, G. A., *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*, MIT Press, 1992.
- Gibson, G. A., and R. Van Meter, "Network Attached Storage Architecture," in *Communications Of the ACM*, vol. 13, 2000.

- Gibson, G. A., et al., "A Case for Network-Attached Secure Disks," TR-CMU-CS-96-142, Sept. 1996.
- Gnutella project website: <http://gnutella.wego.com>
- Gray, J., "Storage Bricks Have Arrived (talk in the Session on The Future of Storage Technology)," USENIX Conference on File and Storage Technologies (FAST '02). Jan. 2002.
- Gray, J., A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993.
- Grochowski, E. G. a. R. F. H., "Future trends in hard disk drives," *IEEE Transactions on Magnetism*, vol. Vol. 32, 1996.
- Hartman, J., and J. Ousterhout, "The Zebra striped network file system," in *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, 1993.
- Heyn, T., "Jini: A Pathway for Intelligent Network Storage," Seagate Technology, Virtual Press Room online literature Jan. 1999.
- Hitz, D., "An NFS file server appliance," Network Appliances TR3001, Jan. 1997.
- Hitz, D., J. Lau, and M. Malcolm, "File systems design for an NFS file server appliance," in *USENIX Winter 1994 Technical Conference Proceedings*, 1994.
- Hoard, B., "Disaster recovery, interoperability are recurrent themes at Storage Networking World," in *Storage Networking World Online*, 2001.
- Hsiao, H.-I., and D. J. DeWitt, "Chained declustering: A new availability strategy for multiprocessor database machines," University of Wisconsin, Madison CSTR-854, June 1989.
- Huang, M., and L. M. Ni, "Emulation of IP-based Networks," Department of Computer Science and Engineering, Michigan State University, Technical Report 1998.
- Intel, "Virtual Interface (VI) Architecture," Dec. 1997.
- Jayaram, H., E. Torng, Y. Chen, S. Wagner, L. M. Ni, P. Hodges, "The Impact of Smart Disks and Spatial Reuse Property on RAID-5 Storage Systems," in *Workshop on Architectural and OS Support for Multimedia Applications*, 1998.
- Kazar, M. L., et al., "Decorum File System Architectural Overview," in *Proceedings of Summer 1990 USENIX Conference*, 1990.
- Kleiman, T., "Vnodes: An Architecture for Multiple File System Types in Sun UNIX," in *Proceedings of the Summer 1986 USENIX Conference*, 1986.
- Kozierok, C. M., "Storage Review's Reference Guide: Hard Disk Drives," 2001.

- Kubiatowicz, J., et al., "OceanStore: An Architecture for Global-Scale Persistent Storage," in *Proc. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, 2000.
- Leach, P., and D. Naik, "Common Internet File System (CIFS/1.0) Protocol Preliminary Draft," Internet-Draft Dec. 1997.
- Lee, E., and C. Thekkath, "Petal: Distributed virtual disks," in *Proceedings of the ACM 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1996.
- Lenoski, D., J. Laudon, K. Gharachorloo, A. Gupta and J. Hennessy, "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor," in *Proceedings of the 17th International Symposium on Computer Architecture*, 1990.
- Mason, H., "SCSI, the Industry Workhorse, Is Still Working Hard," in *IEEE Computer*, 2000.
- McKusick, M., et al., "A Fast File System for UNIX," *ACM Transaction of Computer Systems (TOSC)*, 1984.
- Mearian, L., M. Hall, "What's After Fibre Channel?," in *Computer World*, 2001.
- Menon, J., and J. Kasson, "Methods for improved update performance of disk arrays," IBM Almaden Research Center, San Jose, CA, Technical Report rJ 6928 (66034), July 1989.
- MichNet Backbone diagram: <http://www.merit.edu/michnet/maps/backbone.html>
- Farsite project website: <http://www.research.microsoft.com/research/sn/Farsite/>
- MSU campus network diagram: <http://mrtg.cl.msu.edu/topology/gigabit1.html>
- Napster website: <http://www.napster.com>
- NetworkAppliance, "DAFS: Direct Access File System Protocol, Version 0.54," Network Appliance and Intel Oct. 2000.
- Ng, S., "Advances in Disk Technology: Performance Issues," in *IEEE Computer*, 1998, pp. 75-81.
- Noblett, T., "Storage networking technology plays prominent role in September 11 disaster recovery," in *Storage Networking World Online*, 2001.
- Norman, B., F. Lee, "Implementing Serial ATA in Next-Generation Computer Systems," in *Computer Technology Review*, 2002.
- Network Simulator ns2: <http://www.isi.edu/nsnam/ns/>

- Parameswaran, M., et al., "P2P Networking: An Information-Sharing Alternative," in *IEEE Computer*, 2001.
- Patterson, D. A., G. Gibson, and R. H. Katz, "A Case for Redundant Array of Inexpensive Disks (RAID)," in *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD)*, Chicago, IL, 1988.
- Paulson, L. D., "Faster Storage Connectivity Technologies Emerge," in *IEEE Computer*, vol. 35, 2002.
- Rhea, S., et al., "Maintenance-Free Global Data Storage," in *IEEE Internet Computing*, 2001, pp. 40-49.
- Riedel, E., C. Faloutsos, et al., "Active Disks for Large-Scale Data Processing," in *IEEE Computer*, 2001.
- Robbins, D., "Introducing Ext3," IBM developerWorks Nov. 2001.
- Rosenblum, M., and J. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Transactions on Computer Systems*, 1992.
- Rowstron, A., and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, 2001.
- Rubini, A., et al., *Linux Device Drivers*, 2nd Ed. ed. O'Reily & Associates, 2001.
- Ruemmler, C., and J. Wilkes, "An Introduction to Disk Drive Modeling," in *IEEE Computer*, 1994, pp. 17-28.
- Sachs, M., A., Leff, and D. Sevigny, "LAN and I/O convergence: A survey of the issues," pp. 24-32, 1994.
- Sandberg, R., D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network File System," 1985.
- Sander, S. L., "Gigabit Ethernet: settling in as a storage networking standard," in *Storage Networking World Online*, 2001a.
- Sander, S. L., "Disaster recovery in the data center," in *Storage Networking World Online*, 2001b.
- Satran, J., et al., "iSCSI (Internet SCSI), Internet draft," IETF Dec. 2000.
- Satyanarayanan, M., "Coda: A Highly Available File System for a Distributed Workstation Environment," in *Proceedings of the Second IEEE Workshop on Workstation Operating Systems*, 1989.

- Satyanarayanan, M., "Scalable, Secure, and Highly Available Distributed File Access," in *IEEE Computer*, 1990, pp. 9-20.
- Savage, S., and J. Wilkes, "AFRAID -- A Frequently Redundant Array of Independent Disks," in *Proceedings of 1996 USENIX Technical Conference*, 1996.
- Seltzer, M. I., G. R. Ganger, M. K. McKusick, et. al., "Journaling versus Soft Updates: Asynchronous Meta-data Protection in File Systems," in *Proceedings of USENIX Annual Technical Conference*, 2000.
- SSA, "Serial Storage Architecture: A Technology Overview, Version 3.0," SSA Industry Association 1995.
- Stodolsky, D., G. Gibson, and M. Holland, "Parity logging: overcoming the small write problem in redundant disk arrays," in *Proceedings of 20th International Symposium on Computer Architecture*, San Diego, CA, 1993.
- Stoica, I., R. Morris, D. Karger, M. F. Kaashoek, and H. BALAKRISHNAN, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *Proc. ACM SIGCOMM*, 2001.
- Stonebraker, M., G. A. Schloss, "Distributed RAID - a new multiple copy algorithm," in *Proceedings of the Sixth IEEE International Conference on Data Engineering*, 1990.
- Tanenbaum, A. S., *Structured Computer Organization*, 4th Ed. ed. Prentice Hall, 1999.
- Teigland, D., H. Mangelshagen, "Volume Managers in Linux," in *FREENIX Track USENIX Annual Technical Conference*, 2001.
- TFCC, "Organization statement," The IEEE Computer Society Task Force on Cluster Computing (TFCC) 2001.
- Thekkath, C., T. Mann, et al., "Frangipani: A Scalable Distributed File System," in *Proceedings of the 16th Symposium on Operating Systems Principles (SOSP)*, 1997.
- TTCP website: <http://www.pcausa.com/Utilities/pcattcp.htm>
- Tweedie, S., "EXT3, Journaling Filesystem," Ottawa Linux Symposium 2000.
- Van Meter, R., "A Brief Survey of Current Work on Network Attached Peripherals," in *ACM Operating Systems Review*, 1996, pp. 63-70.
- Van Meter, R., G. Finn, and S. Hotz, "VISA: Netstation's virtual Internet SCSI adapter," in *Proceedings of the ACM 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, CA, 1998.

VERITAS, "RAID for Enterprise Computing," VERITAS Software Corporation 2000.

Webster, J., "The overwhelming importance of storage," in *Storage Networking World Online*, 2001.

Williams, R., "InfiniBand I/O standard will enhance storage networking," in *Storage Networking World Online*, 2001.

WSU campus network diagram: <http://network.ucomm.wayne.edu/>

Yianilos, P. N., and S. Sobti, "The Evolving Field of Distributed Storage," in *IEEE Internet Computing*, 2001, pp. 35-39.

Zhao, B. Y., J.D. Kubiawicz, and A.D. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," Univ. of California at Berkeley tech. report UCB/CSD-01-1141, April 2001.

MICHIGAN STATE UNIVERSITY LIBRARIES



3 1293 02334 0106