Distributed Software RAID Architectures for Parallel I/O in Serverless Clusters*

Kai Hwang, Hai Jin, and **Roy Ho**
Internet and Cluster Computing Laboratory
University of Southern California
Email contact: kaihwang@usc.edu

Abstract: In a serverless cluster of computers, all local disks can be integrated as a *distributed software* RAID (ds-RAID) with a single I/O space. This paper presents the architecture and performance of a new RAID-x for building ds-RAID. Through experimentation, we evaluate the RAID-x along with RAID-5, chained-declustering, and RAID-10 architectures, all embedded in a Linux cluster environment. All four ds-RAID architectures aim to scale in aggregate I/O bandwidth. The RAID-x is unique with its *orthogonal striping* and *mirroring* (OSM) architecture. The reliability comes from orthogonal mirroring, while the bandwidth is enhanced from distributed striping. To support single I/O space, *we* have developed *cooperative disk drivers* (CDD) at the Linux kernel to enable fast remote disk accesses without using a central file server such as the NFS.

The performance of the RAID-x is experimentally proven superior in three areas: (1) significant improvement in I/O bandwidth especially in parallel write operations, (2) pipelined mirroring in the background with low overhead, and (3) enhanced scalability and reliability in cluster computing with a single I/O space. These claims are supported by Andrew and Bonnie benchmark results, obtained on the USC cluster of 16 Linux PCs. Reliable cluster middleware and Linux extensions are developed to enable not only single I/O space, but also shared virtual memory and global file hierarchy. Toward this end, we compare the RAID-x with four related parallel or distributed RAID projects: Digital Petal, Berkeley Tertiary Disk, Princeton TickerTAIP, and HP AutoRAID. Their relative strengths, shortcomings, and applications are discussed along with suggested further research.

Index Terms: Distributed computing, **parallel I/O, software RAID, single I/O space, fault tolerance, Andrew and Bonnie benchmarks, scalability analysis, and Linux clusters.**

Table of Contents:

* **Manuscript submitted to** ***IEEE Trans. on Parallel and Distributed Systems*** **on Sept. 25, 2000. Preliminary results of various aspects of this work have been presented in the** ***9th IEEE High Performance Distributed Computing Symposium*** **(HPDC-9), Pittsburgh, August 2000, and the** ***20th International Conference on Distributed Computing Systems*** **(ICDCS 2000), April 2000, Taiwan. This research was supported in part by the University of Southern California and in part by the Hong Kong RGC grant HKU 7022/97E. All rights reserved by the co-authors.**

1. Introduction

A *centralized* RAID (*Redundant Array of Inexpensive Disks*) [7] is built with packed disks under a single controller. Such a hardware-integrated RAID is often attached to a central storage server or used as a network-attached I/O subsystem [13]. The classic disk arrays are most built as centralized RAIDs, accessible from all clients through a network file server (NFS). In a serverless cluster of computers, no central server is used. The client-server architecture does not apply here. Instead, all clients in the cluster must divide the storage server and file management functions in a distributed manner [1]. Thus, a serverless cluster often demands a *distributed software RAID* (denoted as ds-RAID henceforth) architecture, which embodies dispersed disks physically attached to client hosts in the cluster [22].

A ds-RAID is also known as a *distributed* RAID or a *software* RAID, due to the fact that distributed disks are glued together by software or middleware in a network-based cluster. This paper presents a new RAID-x architecture for building ds-RAIDs in serverless clusters. The Princeton TickerTAIP project [5] offered a *parallel* RAID architecture for supporting parallel disk I/O with multiple disk controllers. However, the TickerTAIP was still implemented as a centralized I/O subsystem. The HP AutoRAID [47] was built as a hierarchy of RAID-1 and RAID-5 subsystem, which was inspired by the earlier work on AFRAID [39]. We consider both I/O subsystems as parallel RAIDs, not really ds-RAIDs in a strict sense.

The distributed RAID concept was originally explored by Stonebraker and Schloss [40] in 1990. Prototyping of ds-RAID started with the Petal project [30] at Digital Laboratories and with the Tertiary Disk project [3][41] at UC Berkeley. The Petal was built with a disk mirroring scheme, called chained declustering [19]. The Tertiary Disk adapted a parity-checking RAID-5 architecture, using the serverless xFS file system [1]. Our RAID-x was built as a distributed architecture in the USC Trojans cluster project. The level x in RAID-x is yet to be assigned by the RAID Advisory Board.

The basic concept of single I/O space in a ds-RAID was treated in our earlier paper appeared in IEEE *Concurrency Magazine* [22]. Preliminary results of different aspects of the RAID-x have been presented in the IEEE *HPDC-9* [24] and the 20-th *ICDCS* [17] conferences. A grouped sector approach was suggested in Jin and Hwang [25] for fast accessing a ds-RAID subsystem. This journal paper gives a comprehensive treatment of the RAID-x with substantial revisions and extended experimental results. We also compare the RAID-x with other ds-RAID architectures that have been built in the past. Besides architectural innovations, we report benchmark performance results obtained recently on RAID-x, RAID-5, RAID-10, and chained declustering architectures.

To build a ds-RAID, one must establish three fundamental capabilities: (i) a *single I/O space* (SIOS) for all disks in the cluster, (ii) high scalability, availability, and compatibility with I/O-centric cluster applications, and (iii) local and remote disk I/O operations performed with almost the same latency. These requirements imply a total transparency to all users. We concentrate on ds-RAIDs that are specially designed for parallel I/O in cluster computing. We emphasize distributed environment in which hardware disks may be scattered around a local-area network. It is the software or middleware that glues the distributed disks together for collective I/O operations.

To benefit the readers, notations and abbreviations used in this paper are summarized in Table 1. Let *n* be the size of the disk array and *B* be the maximum bandwidth per disk. *R* and *W* stand for the *average read* or *write times* (latencies) per block (or *stripe unit s*) of 32KB. Normally, W is slightly longer than R, both in the order of a few ms based on today's disk technology. The *file size*, denoted as *m*, is indicated by the number of blocks contained in an I/O file. Note that all the blocks in a file are striped across disks distributed over all cluster nodes.

Table 1. Notations and Abbreviations Used

Parameter	Definition	Abbr.	Meaning
n	No. of disks in a ds-RAID	ds-RAID	Distributed software RAID
B	Max. bandwidth per disk	RAID-x	Redundant array of inexpensive disks at level x
s	Disk block size (stripe unit)	OSM	Orthogonal stringing and mirroring
R	Ave. read time per disk block	SIOS	Single I/O space in a cluster
W	Ave. write time per disk block	CDD	Cooperative disk drivers
m	No. of disk blocks in a file	NFS	Network file server

The rest of the paper is organized as follows: Section 2 introduces three approaches to building *single I/O space* (SIOS) in ds-RAIDs embedded in clusters. Section 3 introduces the OSM architectural features and functions. Section 4 specifies the detailed RAID-x design and benchmark experiments performed in the USC Trojans cluster. Section 5 describes the architecture and data consistency mechanisms of the *cooperative disk drivers* (CDD). The relative merits in using CDDs and NFS for parallel I/O are discussed here.

The main performance results are presented in Sections 6 and 7. Section 6 reports the aggregate I/O bandwidth of four competing ds-RAID architectures. Section 7 reports the Andrew and Bonnie benchmark results on RAID-x and other competing ds-RAID architectures. Major parallel and distributed RAID projects are assessed in section 8. Finally, we summarize the research contributions, identify the strengths and weaknesses of competing ds-RAID architectures. The results obtained can be applied to distributed I/O applications in either Unix or Linux clusters. We also suggest directions for further research, development, and applications of ds-RAIDs in general. The emphasis will be on the integration of hardware, software, and middleware for cluster I/O applications.

2. Basic Concepts of Distributed Software RAIDs

In a cluster with a *single I/O space* (SIOS), any cluster node can access either local disk or remote disk without knowing the physical location of the I/O devices [22]. Such a collection of distributed disks form essentially a ds-RAID subsystem. All distributed disks can access each other in a single address space. The SIOS capability is crucial to building a scalable cluster of computers. The ultimate goal is to realize a *single system image* (SSI) [37] in the cluster.

The SSI capability covers a wide areas of services including cluster job management, parallel programming, collective I/O, and availability support, etc. [22]. Our primary design objective of a ds-RAID is to achieve a single address space for all disk I/O operations. Without the SIOS, remote disk I/O must go through a sequence of Unix or Linux system calls over a centralized file server (such as the NFS). Once the SIOS is established, all dispersed disks in the ds-RAID can be used collectively as a single *global virtual disk* as illustrated in Fig. 1.

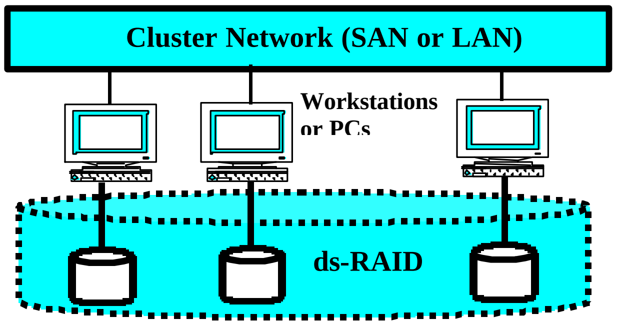


Figure 1. A distributed software RAID with a single I/O space embedded in a cluster of computers

There are three advantages to have the SIOS design in ds-RAID. First, the problem of unequal latencies in local and remote disk accesses is greatly alleviated. Remote disk access does not have to be handled by system calls from the requesting host. Instead, the requester can address the remote disks directly by a light-weight process. Second, the SIOS supports a persistent programming paradigm. It facilitates the development of a number of other SSI services such as *distributed shared memory* (DSM) and *global file hierarchy* [17]. All I/O device types and their physical locations are now transparent to all users. Third, the SIOS allows striping across distributed disks. This accelerates parallel I/O operations in the cluster environment.

In the past, three approaches have been attempted to achieve the SIOS capability at the user level, file-system level, and device-driver level. The *user-level SIOS* is implemented with I/O service routines or libraries. The Parallel Virtual File System [6], abstract-device interface ADIO [42] and the remote I/O project [12] are typical examples of user-level SIOS. This approach has two shortcomings: First, users still have to use specific APIs and identifiers to explore parallelism in I/O. Second, using system calls to perform network and file I/O are still needed, which is rather slow to meet real-time or cluster computing requirements.

A *distributed file system* achieves the SIOS across distributed storage servers [8][10][14][21][34]. The file system must distribute the data within the SIOS. The serverless xFS system built at Berkeley [1] and the Frangipani system developed in the Petal project [43] are two good examples. However, this file-level approach has its own shortcomings. Changing the file system does not guarantee higher compatibility with current applications. Instead, it may discourage the deployment of the distributed file systems in clusters running well-established operating systems.

What we want to achieve is a SIOS supported by device drivers at the kernel level. The *device* *driver* must be designed to enable the SIOS to both users and the file system applied. Digital Petal [30] developed a device driver at the user space to enable remote I/O accesses. All physically distributed disks are viewed as a collection of a large virtual disk. Each virtual disk can be accessed as if it is a local disk. This approach has the difficulty to control the file distribution pattern. Many device driver designers have their own custom-designed file system to optimize the performance. Petal project has developed such a distributed file system, called Frangipani [43].

The main objectives of our SIOS design are identified below. These features can be applied to implement any ds-RAID architectures, not necessarily restricted to the new RAID-x.

1. *A single address space for all disks in the cluster* - We build the SIOS by designing a *cooperative disk driver* (CDD) for use in each local disk. These CDDs will be described in Section 5. The CDDs cooperate with each other to form the single address space. They work collectively to maintain the data consistency among distributed data blocks. The file system running on in each node has the illusion that there is only one large virtual disk shared by all clients in the cluster.
2. *High availability supported* - High availability is desired in a cluster by building some fault-tolerant features in the ds-RAID. We have implemented in RAID-x a new disk mirroring mechanism for this purpose. Other architectures such as RAID-5, RAID-10 and chained declustering also have deployed some specific features for availability enhancement.
3. *Performance and size Scalability* - There is no central server in our cluster design and the CDDs only maintain a peer-to-peer relationship among themselves. So, size scalability must be guaranteed in the serverless cluster design. Furthermore, we stripe the data blocks across distributed disks. Performance gain by parallel striping will be verified by benchmark results in Sections 6 and 7.
4. *High compatibility with cluster applications* - This is a “clean” design in the sense that we concentrate on device driver modification, without changing the file system or user libraries. However, the ds-RAID design must be compatible with existing cluster I/O applications. In particular, the modification of existing file systems should be avoided. This will reduces implementation effort in users.

In subsequent sections, we prove that the RAID-x architecture outperforms the RAID-5, RAID-10 and chained declustering RAID, especially in parallel write operations. The RAID-x has also lower access times than either RAID-10 or chained declustering. These claims will be verified by real benchmark results in Sections 6 and 7. To sum up, the RAID-x scheme demonstrates scalable I/O bandwidth with much reduced latency in a serverless cluster environment. Using the CDDs, a cluster can be built serverless and offers remote disk access directly at the kernel level. Parallel I/O is made possible on any subset of local disks forming a SIOS. No heavy cross-space system calls are needed to perform remote file accesses.

3. Orthogonal Striping and Mirroring

Our new RAID-x architecture was influenced by the AFRAID design [39] at Hewlett Packard, by the chained declustering concept [19] proposed at the University of Wisconsin, and by the Digital Petal project [30]. Our RAID-x differs from these architectures in two aspects. First, the RAID-x is built with a new clustered mirroring technique, called *orthogonal striping and mirroring* (OSM). Second, we have developed *cooperative disk drivers* (CDD) to implement the OSM at the Linux kernel level. The CDD maintains data consistency directly without using a NFS or demanding any inter-space Unix system calls.

In this paper, we present the design details of RAID-x and prove its effectiveness through experimentation. Figure 2 shows the basic concepts of four ds-RAID architectures, namely the RAID-5 (Fig.2a), the RAID-10 (Fig.2b), the RAID-x (Fig.2c), and the chained declustering RAID (Fig.2d). We have implemented all these four ds-RAIDs in our Trojans cluster. Previously, the RAID-5 has been built in the TickerTAIP, Tertiary Disk, AFRAID, and partially in the AutoRAID architectures. The RAID-10 was also partially built in the AutoRAID. The chain declustering was adopted in the Petal project.

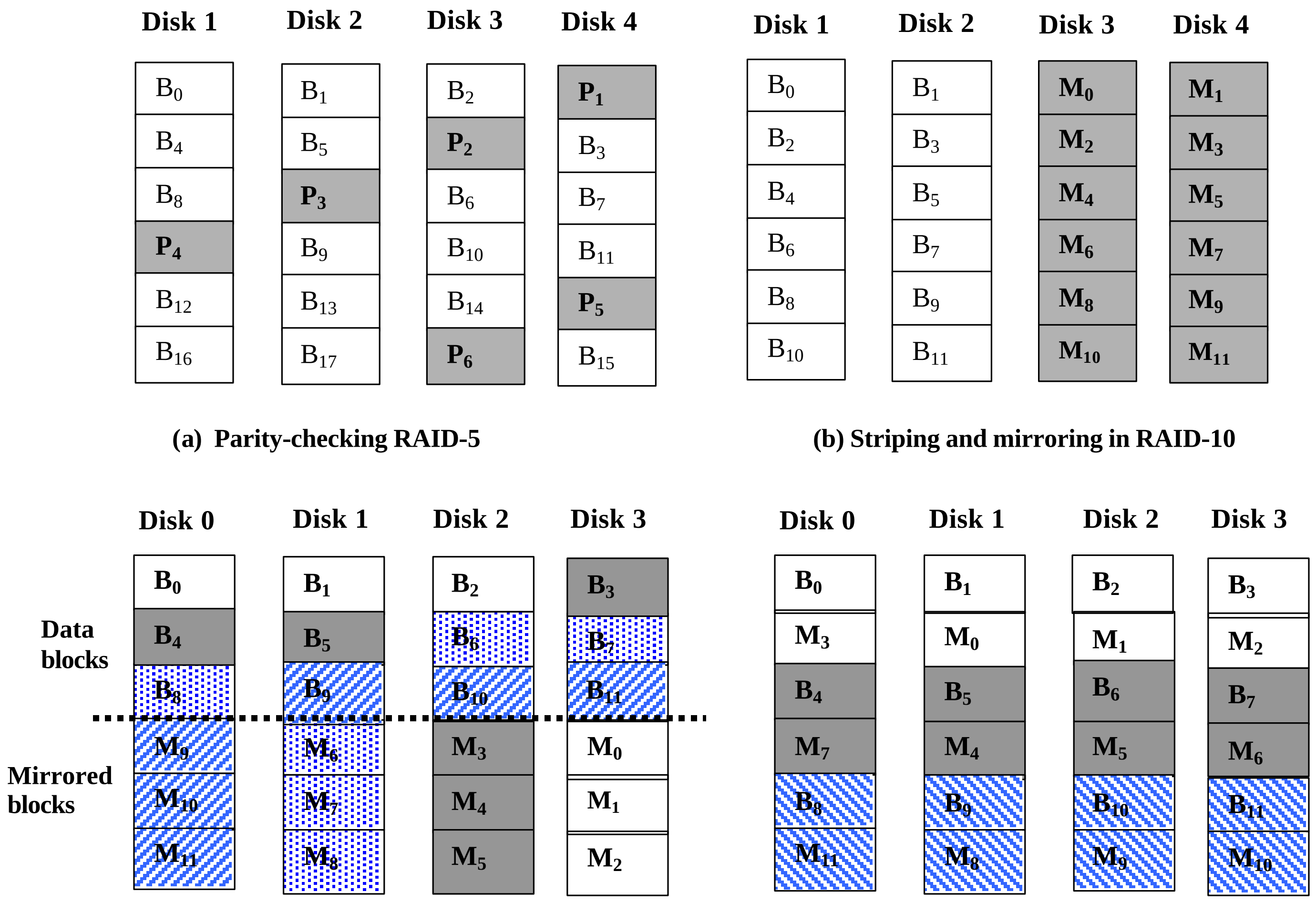


Figure 2. Four RAID architectures being evaluated for building the ds-RAIDs

The RAID-5 uses parity checking. All others use some forms of disk mirroring. The original *data blocks* are denoted as B_i in the unshaded boxes. The *mirrored images* are marked with M_i in shaded boxes. The RAID-10 duplicates each disk image on an extra disk without striping, thus it is easier to implement. Chained declustering [19] allows a skewed striping of image blocks. The concept of OSM is based on orthogonal striping and clustering, which we shall prove has the advantages of both RAID-10 and chained declustering.

As shown in Fig.2c, data blocks in RAID-x are striped across the disks on the top half of the disk array. Low latency and high bandwidth of RAID-10 are preserved in RAID-x architecture. The image blocks of other data blocks in the same stripe are clustered in the same disk vertically. All image blocks occupy the lower half of the disk array. On a RAID-x, the images can be copied and updated at the background, thus reducing the overhead time.

Consider the top stripe of data blocks B_0, B_1, B_2, and B_3 in Fig.2c. Their image blocks M_0, M_1, and M_2 are stored in Disk 3, while the image block M_3 in disk 2. The rule of the thumb is that *no data block and its image should be mapped in the same disk*. Full bandwidth is thus achievable in parallel disk I/O across the same stripe. For large write, the data blocks are written in parallel to all disks simultaneously. The image blocks are gathered as a long block written into the same disk with a reduced latency. In case of small write of a single block, the writing is directed to the data block, while the image block is postponed to a write to the disk until all clustered image blocks are ready.

Two mapping functions are given below to specify how to map the logical data blocks and their images to physical RAID blocks. Let b be the number of blocks per disk. A logical data block addressed by d is mapped to the i-th disk and the j-th block, where $i = d \bmod n$ and $j = (2d/n) \bmod n$. The mirrored image block of d is mapped to the i-th disk and j-th block, where

$$i = n - 1 - (d/(n-1)) \bmod n \tag{1}$$

$$j = b/2 + (d/(n-1)\, n)\,(n-1) + d \bmod (n-1) \tag{2}$$

Table 2 lists the theoretical peak performance of four RAID architectures. The entries in Table 2 are theoretical peak performance of parallel disk I/O operations, excluding all software overhead or network delays. The *maximum bandwidth* of a disk array reflects the ideal case of parallel accesses to all useful data blocks. In the best case, a full bandwidth of nB can be delivered by RAID-10, RAID-5, RAID-x, and chained declustering. But the RAID-5 has a lower write bandwidth due to the access of redundant parity blocks.

The parallel read or parallel write of a file of m blocks depends on the read or write latencies denoted as R and W per block. In case of large reads, mR/n latency is expected to perform m/n reads simultaneously. For a small read, all RAID architectures require R time to complete the read of a single block. For parallel writes, RAID-10 and chained declustering requires to double the number of disk accesses. In RAID-x, the image blocks are clustered written into the same disk. That is, $m/n(n\text{-}1)$ image blocks are written together to each disk. The large write latency is reduced to $mW/n + mW/n(n\text{-}1)$.

Table 2 Theoretical Peak Performance of Four RAID Architectures

Performance Indicators		RAID-10	RAID-5	Chained Declustering	RAID-x
Max. I/O Bandwidth	**Read**	$n B$	$n B$	$n B$	$n B$
	Large Write	$n B$	$(n\text{-}1) B$	$n B$	$n B$
	Small Write	$n B$	$nB / 2$	$n B$	$n B$
Parallel Read or Parallel Write Time	**Large Read**	mR / n	mR / n	mR / n	mR / n
	Small Read	R	R	R	R
	Large Write	$2\ mW / n$	$mW / (n\text{-}1)$	$2\ mW / n$	$mW / n + mW / n(n\text{-}1)$
	Small Write	$2W$	$R+W$	$2W$	W
Max. Fault Coverage		$n/2$ disk failures	Single disk failure	$n/2$ disk failures	Single disk failure

For small writes, chained declustering needs to write the data and image blocks in $2W$ time, because interleaved data and images must be written separately. Our RAID-x takes only a single parallel write in W time to all data blocks involved. The writing of the image blocks is done later when all stripe images are loaded. This clustered writing is done at the background, overlapping with the regular data writes. In a centralized RAID-5, it may take 2W + 2R time to finish a small write cycle. However, the time can be reduced to W + R in a distributed RAID-5, because both reads and writes can be done in parallel in the distributed cluster environment. The maximum I/O bandwidth of distributed RAID-5 is $nB / 2$ instead of $nB / 4$ as in a centralized RAID-5.

The RAID-x shows the same bandwidth potential as provided by the RAID-10 and the chained declustering RAID. This is because all the disks can be fully accessed in parallel in these architectures. RAID-x can tolerate all single-disk failures, same as that of RAID-5. It is true that both RAID-10 and chained declustering are more robust than RAID-x. The bottom row of Table 2 shows the maximum number of disk failures that each RAID architecture can tolerate.

As revealed in Sections 7 and 8, the major strength of RAID-x lies in its much improved write performance than RAID-5, RAID-10, or chained declustering. In term of fault-tolerance, RAID-x is as good as the RAID-5. But RAID-x performs much better than RAID-5 in coping with the small write problem. The major weakness of RAID-x lies in its 50% redundancy, same as that required in a RAID-10 and in a chained declustering RAID. The experimental results will verify all claimed advantages and identify the shortcomings of competing ds-RAID architectures as well.

4. The RAID-x Architecture and Experiments

All ds-RAID architectures, including the new RAID-x, are implemented in an experimental Linux-base PC cluster built at USC. As shown in Fig. 3, the prototype cluster was built with 16 Pentium PCs running the Linux operating system. These PC nodes are connected by a 100-Mbps Fast Ethernet switch. At present, each node is attached with a 10-GB IDE disk. With 16 nodes, the total capacity of the disk array is 160 GB. All 16 disks form a single I/O space, which became operational since 1999. All PC nodes in the Trojans cluster are homogeneous and supported by middleware packages such as DQS, Condor, MPI, PVM, TreadMarks, NAS, Linpack, STAP, netperf, etc.

At different experimenting stage, we reconfigure the harware disk array in Trojans cluster into four different ds-RAID configurations, namely the RAID-5, RAID-10, chained declustering, and RAID-x. The Redhat Linux 6.0 version 2.2.5 has already built-in features to support the RAID-0, RAID-10, and RAID-5 configurations. Without the single I/O space built in the Trojan cluster, one cannot implement any ds-RAID configurations. Our major design and implementation effort was exerted on the RAID-x and chained declustering RAID configurations, which are not commercially available. The centralized NFS is also used as a baseline for comparison purposes.

Figure 3 The Prototype Trojans cluster built at the USC Internet and Cluster Computing Laboratory (Visit Trojans Cluster home page: http://andy.usc.edu/trojan/ for details)

The mapping (Eqs. 1 and 2) of the mirrored blocks in the RAID-x is done by a special address translation subroutines built in each disk driver. We will describe the structure of the disk drivers in Section 5. Similar mappings are implemented for the chained declustering RAID, which is really extended from shifting the RAID-10 configuration. All of these ds-RAID configurations were made possible with the SIOS feature we have built in the Trojans cluster. To study the asymptotic aggregate I/O bandwidth of the disk arrays, we have to lift the caching restrictions on local disks. This was done in the Linux kernel by issuing a special *sync* command. The caching effects will be further studied in Section 9.

Figure 4 shows the orthogonal RAID-x architecture with 3 disks attached to each node. We call this an orthogonal 4 x 3 configuration. All disks within the same horizontal stripe group, (B_0, B_1, B_2, B_3), are accessed in parallel. Consecutive stripe groups, such as (B_0, B_1, B_2, B_3), (B_4, B_5, B_6, B_7), and (B_8, B_9,

B_{10}, B_{11}) are accessed in a pipelined fashion, because they are retrieved from the same disk groups attached to the same SCSI buses on different PC nodes.

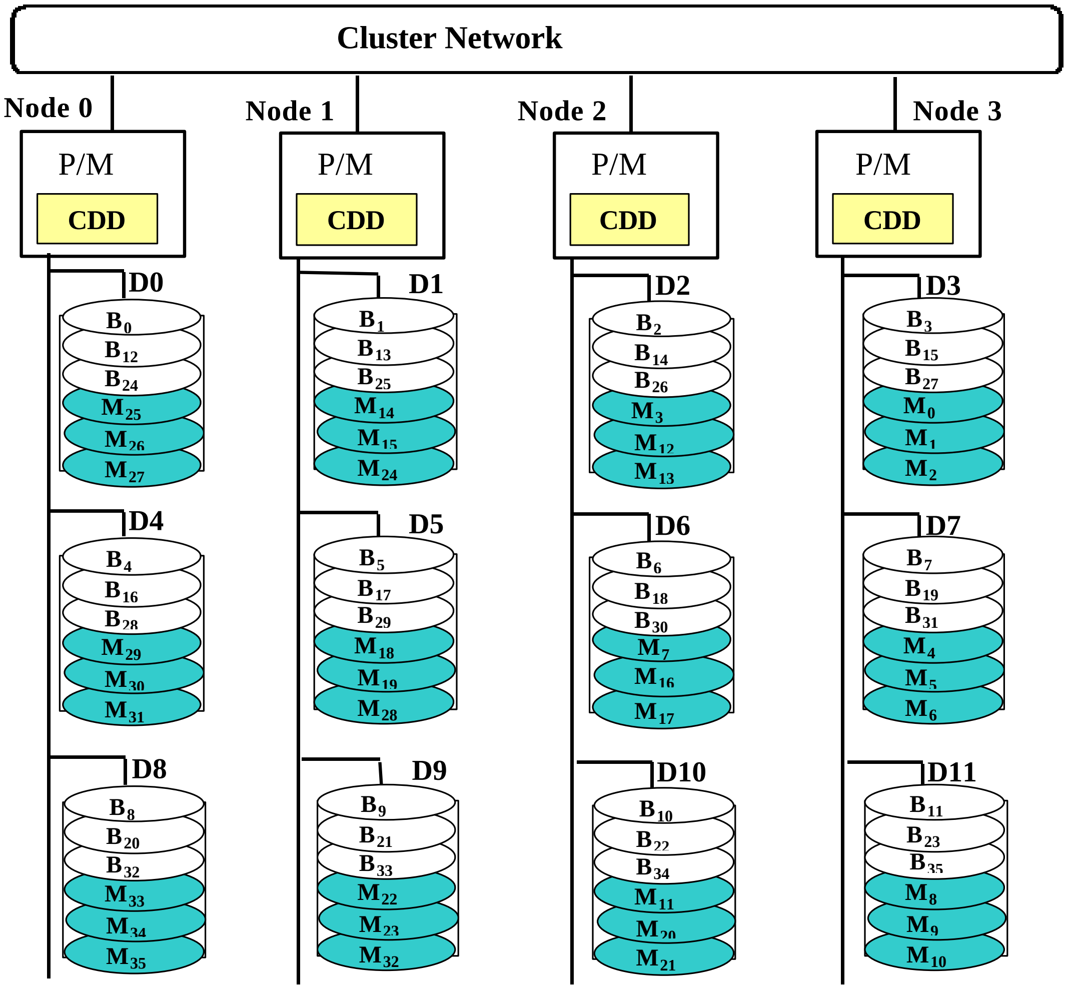


Figure 4 A 4 x 3 RAID-x architecture with orthogonal striping and mirroring
(P: processor, M: memory, Dj: the j-th disk, B_i: the i-th data block,
M_i: the i-th mirrored image in a shaded block)

In general, an *n*-by-*k* RAID-x configuration has a *stripe group* involving *n* disk blocks residing on n disks. Each *mirror group* has *n*-1 blocks residing on one disk. The images of all data blocks in the same stripe group are saved on exactly two disks. The block addressing scheme stripes across all *nk* disks sequentially and repeatedly. The parameter *n* represents the *degree of parallelism* in accessing the disks. The parameter *k* implies the *depth of pipelining*. Tradeoffs do exist between these two orthogonal concepts for parallel processing.

The pipelining depth of successive stripe groups depends on the I/O bus used. We plan to extend the Trojans cluster with 4 disks per node. Using 20 GB SCSI disks, the RAID-x array may have 1.28 TB on 64 disks. In next phase of construction, we may extend the Trojans cluster to hundreds of PC nodes using the next generation of microprocessors and Gigabit switches. For example, with a 128-node cluster and 8 disks per node, the disk array could be enlarged to have a total capacity exceeding 20 TB, suitable for any large-scale, database, multimedia, or scientific applications. With an enlarged array of 128 disks, the cluster must be upgraded to use a Gigabit switched connection.

In our experiments, each disk block (stripe unit) is set as 32 KB. This means that a 20-MB file is striped uniformly across all 16 disks in consecutive stripe groups. For small reads or writes, the data size is set 32KB, retrieved from one block of a disk. We have performed three benchmark experiments. These experiments measure the parallel I/O performance in terms of the *throughput* or the *aggregate I/O bandwidth*. The first experiment tests the throughput scalability of three ds-RAID configurations against the NFS for increasing number of *client requests*. The second test checks the bandwidth scalability against the disk array size. The third test reveals the effects of the data block size on the throughput. We have also evaluated these ds-RAID architectures using the Andrew and Bonnie Benchmarks, which reflect the industrial testing standards.

For both large and small writes, the RAID-x has a shorter access time than either the RAID-10 or the chained declustering RAID. These claims will be verified by the benchmark results in Section 6 and 7. To sum up, the RAID-x scheme demonstrates scalable I/O bandwidth with much reduced latency in a cluster environment. Using the CDDs, the cluster is built serverless and offers remote disk access directly at the kernel level. Parallel I/O is made possible on any subset of local disks, because all distributed disks form the SIOS. No heavy cross-space system calls need to perform in accessing remote files.

5. Cooperative Disk Drivers in Clusters

This section distinguishes the differences in remote disk access by using CDD versus NFS. Then we describe the functional structure of the CDD design and its implementation considerations. In particular, we specify the same CDD mechanisms used to maintain data consistency in all four ds-RAID architectures being evaluated.

5.1. Remote Disk Access Using the CDDs

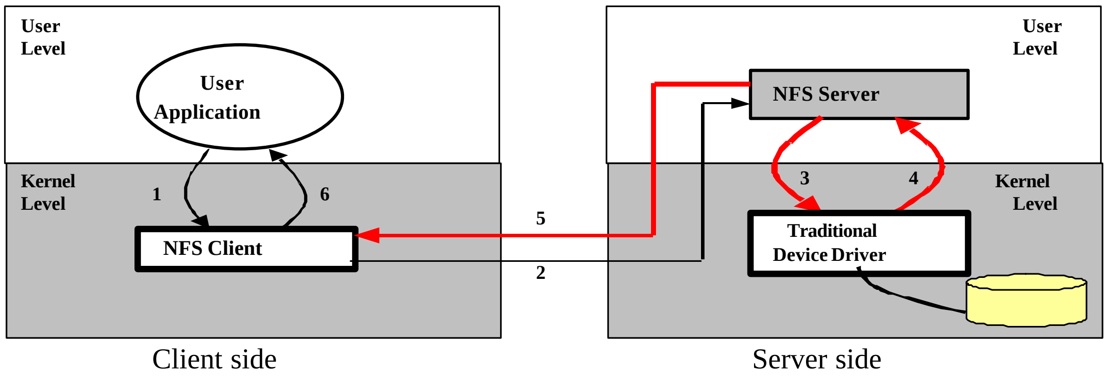
We choose to design the RAID-x with SIOS support at the Linux driver level. This design provides a SSI to the users, demanding no file system modification. The SIOS is supported by a set of CDDs at the kernel level. There is no need to use a central server in our design. Each CDD maintains a peer-to-peer relationship with other CDDs. Figure 5 illustrates the difference of using the NFS and CDDs for remote disk accesses.

Figure 5a shows six steps to access remote disk files using a central NFS server. Note that Unix or Linux system calls take place in Steps 3 and 5 crossing the boundary between user space and kernel space. This causes a long delay in transferring data files among distant disks.

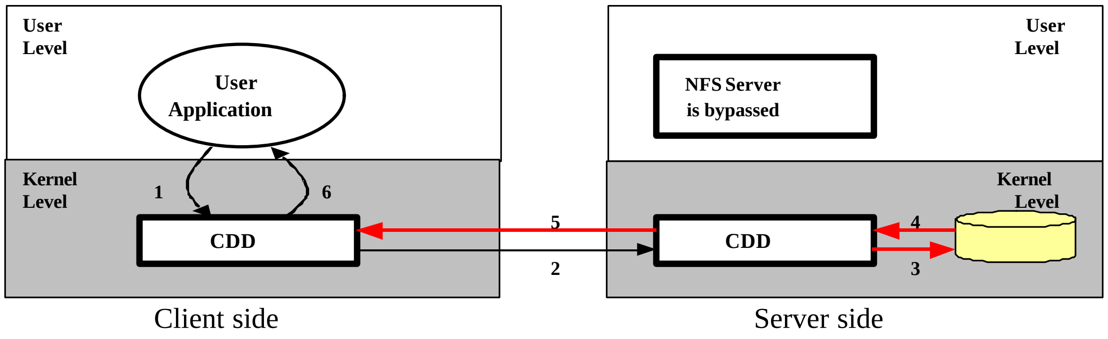
Step 1 - User requests to access the remote disk

Step 2 - Requests are redirected to the remote NFS server

Step 3 - NFS Server accesses the local disk by an I/O system call
Step 4 - Data is copied back to the NFS server in the user space
Step 5 - NFS server sends the data back by a network system call
Step 6 - Data are transferred to the user application



(a) Parallel disk I/O using the NFS in a server/client cluster.



(b) Using CDDs to achieve a SIOS in a serverless cluster.

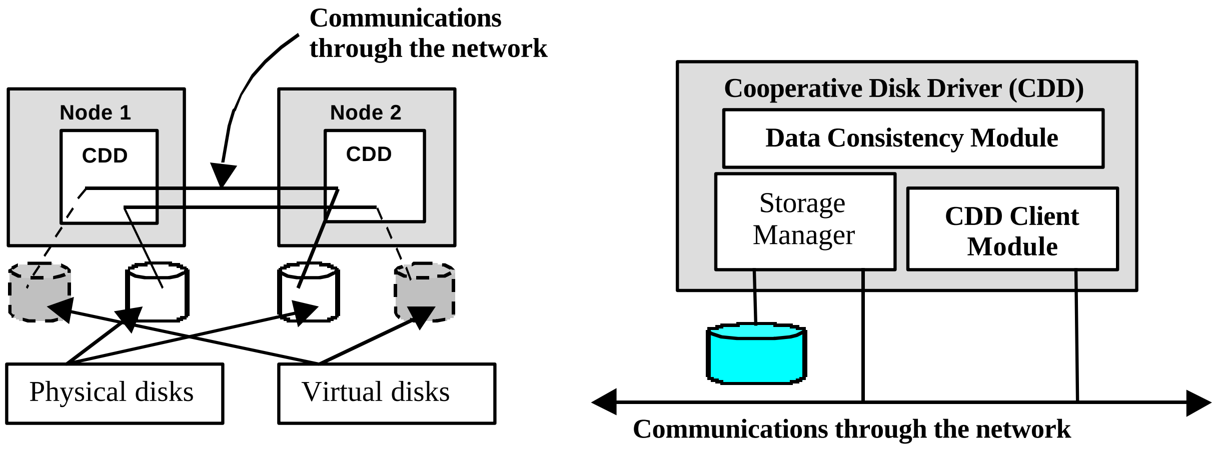
Figure 5 Remote disk access steps using a central NFS server versus the use of cooperative disk drivers in the RAID-x cluster.

Figure 5b shows the modified six steps that are implemented differently using the CDDs for the same purpose. First, system calls are avoided in Steps 3 and 5. Second, cross-space data copying is avoided in Steps 2 and 4. Steps 1 and 6 are identical to those in using the NFS. In a serverless cluster design, each node can be both a client and a server. Therefore, the scalability is improved when this architecture is applied to build larger ds-RAID in clusters.

5.2. The CDD Architecture

The *device masquerading technique* [17] is the key concept behind designing the CDDs. The idea is to redirect all I/O requests to remote disks. The results of the requests, including the requested data, are transferred back to the originating nodes. This mechanism gives an illusion to the operating systems that the remote disks are attached locally. Figure 6a illustrates the device masquerading technique. For simplicity, consider the case of only one disk attached to each cluster node. Multiple CDDs run cooperatively to redirect I/O requests to remote disks.

Each node perceives the illusion that it has two physical disks attached locally. Figure 6b shows the internal design of a CDD. Each CDD is essentially made from three working modules. The s*torage manager* receives and processes the I/O requests from remote c*lient modules*. The Client Module redirects local I/O requests to remote Storage Managers. The d*ata consistency module* is responsible for maintaining data consistency among distributed disks. A CDD can be configured to run as a storage manager or as a CDD client, or both at the same time. This means that there are three possible states of each disk: (1) a manager to coordinate use of local disk storage by remote nodes, (2) a client accessing remote disks through remote disk managers, and (3) both of the above functions.



(a) Device masquerading **(b) CDD architecture**

Figure 6 Architecture design of the cooperative device drivers

The ds-RAID architectures can be implemented easily with the support of CDDs. All data transfers and consistency issues are handled transparently within the CDDs. The RAID drivers only need to maintain data distribution and related policies. Combining the CDD and OSM, the RAID-x disk array shows four advantages. First, it preserves the RAID-0 bandwidth, while adding the fault tolerance capability. Second, it creates a SIOS across all distributed disks. Third, the CDD can be used to implement other RAID configurations as well. Last, data consistency is maintained by the CDD instead of the file system.

5.3. Data Consistency Checking

The data consistency problem arises when multiple cluster nodes have cached copies of the same set of data blocks. The xFS and the Frangipani maintain data consistency at the file system level. In our RAID-x design, data consistency checking is supported at the kernel level using specially designed disk drivers. Our approach simplifies the design of the distributed file management services. Blok-level data consistency is maintained by all CDDs in the same stripe group. Figure 7 shows a scenario that global file directory /*sios* is mapped to each cluster node.

In a global file hierarchy, all files in a cluster are accessed with a global directory. When a client executes a file operation, the global file hierarchy (*inode*) is mapped to the local *vfs* file system at that client. The *File3* is distributed among disks of *node1*, *node2* and *node3*. Data consistency modules in CDDs of *node1*, *node2* and *node3* collectively maintain the consistency of *file3 inode* entry in a global file directory */sios*, which is kept consistent after any local disk update.

Similar to that used in the Frangipani file system, we use multiple-read and single-write locks for to synchronize I/O operations among the cluster nodes. A write lock on a data block permits a client driver to read, to modify, and to cache a modified copy in its local memory. A read lock permits a client driver to cache a read-only copy of the data block. A CDD must use the designated lock to read or write a data block. If a block is not locked, any client can read or write that block.

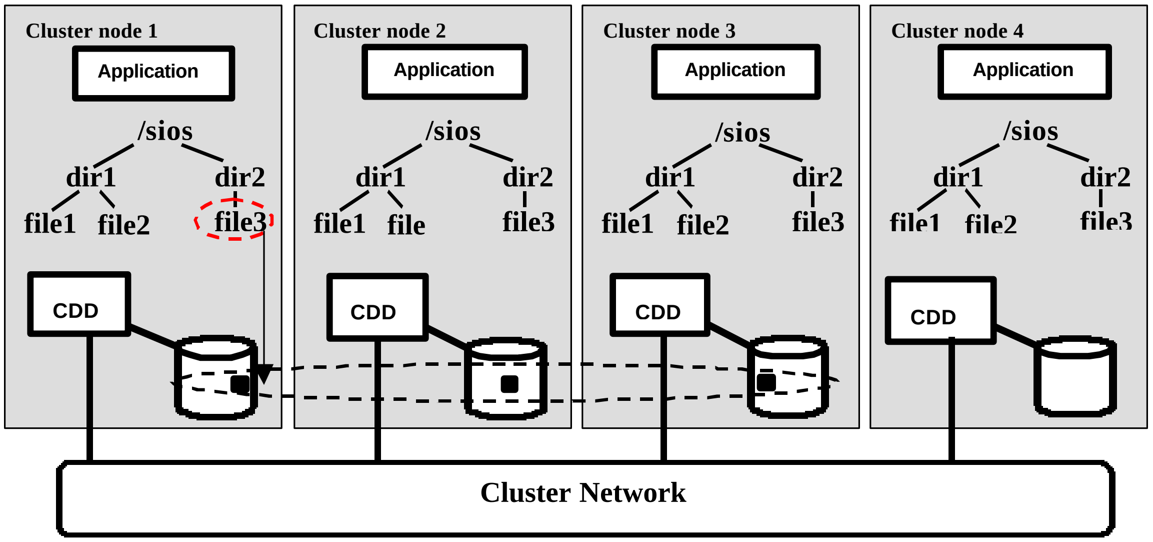


Figure 7 Maintaining consistency of the global directory */sios* by all CDDs in any of the ds-RAID configured on the Trojans cluster

Only one CDD client can hold the write lock of a data block at a time, however the read locks may be granted to multiple CDD clients at the same time. If one CDD client holds a write lock while another is requesting a read/write lock on the same data block, the first client needs to flush its cache entries to the disks. It releases the lock if a write lock is requested; otherwise the lock will be downgraded to a read lock. If one CDD client holds a read lock while another is requesting a write lock, the first client will invalidate its cache entries and release the lock.

We introduce a special *lock-group table* to facilitate distributed file management services. Each entry in this table corresponds to a group of data blocks that have been granted to a specific CDD client with write permissions. The write locks in each entry are granted and released atomically. This lock-group table is replicated among the data consistency modules in the CDDs. A set of query functions was developed to support the checking purposes. The consistency module not only checks the consistency of the data blocks cached into the buffer, but also maintains the consistency of the data structure *inode* in the

Linux file system. Based on the above facilities, the CDDs guarantee that some file management operations are performed in an atomic manner. Distributed file systems running on top of the CDDs are needed with a concurrent access policy. In our experiments, the *ext2* file system in Linux was used.

Security, accounting, and cooperative cache can be also built on top of the CDDs. We overcome the scaling problem associated with the Network File System (NFS) [38] and Andrew File System (AFS) [18]. When the number of clients becomes very large, the SIOS provides a scalable performance through its serverless cluster design. The global file hierarchy is particularly useful for process migration, where the migrated process can access the opened files from any cluster nodes.

6. Aggregate I/O Bandwidth in ds-RAIDs

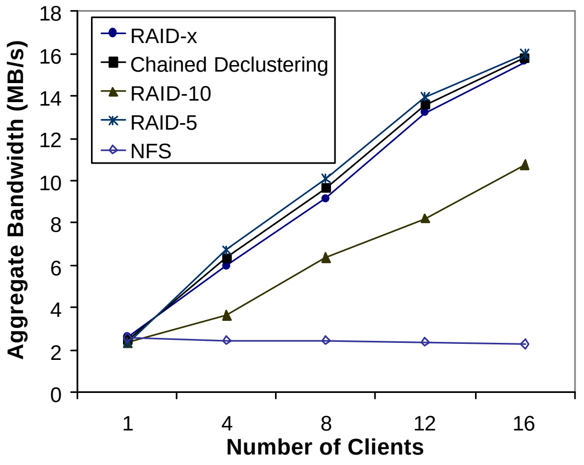
We report the performance results obtained in parallel disk I/O experiments on four ds-RAID configurations running on the Trojans cluster. The I/O performance of a ds-RAID is measured by the *aggregate I/O bandwidth* (or *throughput*) across all disks in the array. In theory, the maximum bandwidth of an *n*-disk array is *nB*, where *B* is the maximum bandwidth of a single disk. In reality, the measured or bandwidth of a ds-RAID would be much lower than the peak value due to the software overhead, network contentions, caching effects, and parity or mirroring penalties incurred.

Our experiments were designed to test the performance of distributed RAID-5, RAID-10, chained declustering, and RAID-x against the use of a central NFS system in the same cluster environment. Through these experiments, we reveal the scalability of each ds-RAID configuration against the I/O traffic rate (client request number), disk array size, and stripe unit size, etc. In what follows, we first study the effects of client requests on the throughput. Then we study the performance effects of disk array size and of the stripe unit size. These effects are useful to determine the scalability and make wise design choices. Our results are useful to both designers and users of the ds-RAID subsystems.

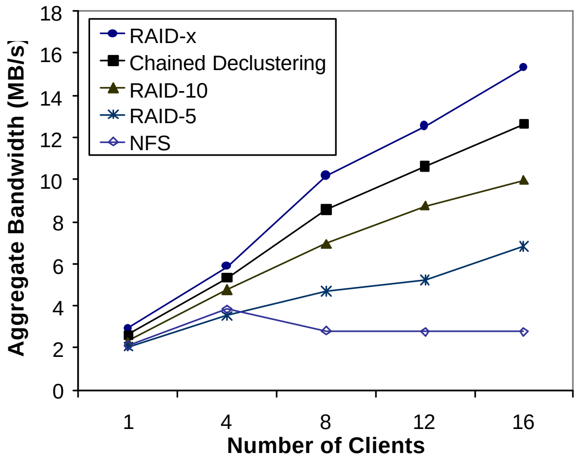
6.1. Effects of I/O Traffic Rate

With striping, a *client request* may access a data file spreading across many disks in the ds-RAID. The number of disks accessed depends on the file size and the block size (stripe unit). The number of simultaneous client requests is the *I/O traffic rate* to or from a RAID subsystem. We consider parallel reads and parallel writes, separately. Furthermore, we discuss the effects of having a large number of such requests taking place at the same observation period. A large file access for either read or write is set at 20 MB. With a block size of 32 KB, this implies a high degree of striping and parallelism. Local or remote disk accesses are generated with a uniform distribution.

In Fig. 8, we plot the measured aggregate I/O bandwidth as a function of the number of client requests for parallel accesses of large files of 20 MB each. We consider from 1 to 16 client requests representing light to heavy I/O traffic conditions. We compare the relative performance of four RAID architectures, all of which are implemented with *n* = 16 disks in the Linux cluster at USC. Large read and large write reveal the parallel I/O capability of the disk arrays. All files are uncached initially and each client only accesses its private files from either local or remote disks. All read or write operations are performed simultaneously, using the MPI_Barrier() command. Each client reads or writes a 20MB file to the buffer and issues the *sync*() call to stripe the data blocks to all 16 disks, recursively.



(a) Parallel reads (20MB per client)



(b) Parallel writes (20MB per client)

Figure 8 Aggregate I/O bandwidth of four ds-RAID architectures, compared with using the NFS on the Linux cluster of 16 nodes at USC

In both read and write accesses, the centralized NFS shows a flat or even declining performance. The NFS server performance is limited between 2.1 and 3.9 MB/s regardless of the number of client requests. This is due to the fact that sequential I/O must be performed on a central NFS server. As the request number increases, the NFS bottleneck effect become worse and thus shows a declining performance. Our cluster network is a 100 Mbps Fast Ethernet. This means that each disk port may have a peak speed of 12.5 MB/s. The fact that the NFS performance being lower than the Ethernet bandwidth clearly demonstrates that the NFS is not scalable in cluster I/O processing applications.

For parallel reads (Fig.8a), the RAID-5, chained declustering, and RAID-x have similar scalable performance. For 16 clients, all top three RAID configurations scale to a bandwidth of 16 MB/s. For a fewer requests, the RAID-5 performs slightly better than chained declustering, which is again slightly higher than the RAID-x. As the number of read requests increases, the performance gaps among the three diminish quickly. The performance of the RAID-10 lags far behind the top three RAID configurations. The gap between them widens sharply as the client number increases. With 16 clients, RAID-10 has a 10.8 MB/s throughput. This slow down is due to the fact that only half of the accessed disks in the RAID-10 carry the desired data blocks. The other half provides mirrored images of the desired data blocks.

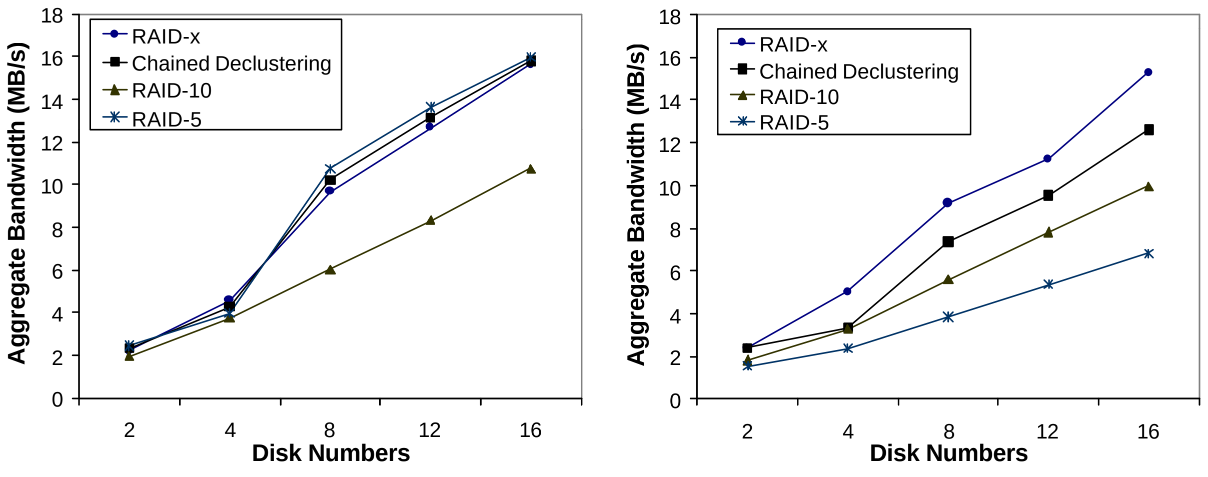
It is the write operation (Fig.8b) that stands out the RAID-x architecture over the rest. Chained declustering ranks the second in parallel writes. The RAID-10 and RAID-5 rank the third and the fourth, respectively. For timing purpose, all write operations among the clients are synchronized. The NFS scales only up to 4 requests, even higher than that of RAID-5, due the caching effect of the NFS server. As the requests exceed four, the NFS bandwidth drops to a low 2.7 MB/s. For parallel writes, the RAID-x achieves the best scalability among the four RAID configurations with a 15.1 MB/s for 16 clients.

In Fig.8, we did not show the performance of small reads or small writes. In these cases, the accesses are made to single data blocks of 32 KB from a single disk in the array. A small read or a small write affects essentially individual disk performance plus the redundancy overhead experienced. With 16 such requests taking place at the same time, we could have a data demand of 32 KB × 16 = 512 KB on the cluster network, simultaneously, the performance curves are quite similar to those shown in Fig.8. The RAID-5 does perform badly for small writes, as shown by its low scalability situation in Fig.8b.

In summary, the RAID-5, chained declustering, and RAID-x have almost equal parallel read performance. RAID-10 scales slower than the top 3 in read operations. For parallel writes, RAID-x clearly outperforms the competing RAID architectures. The chained declustering ranks second and the RAID-10 moves to the third. The write performance of the RAID-5 is the least scalable one among the four RAID architectures being evaluated. This situation is especially true for small write operations. In RAID-x, parallelism is exploited fully to yield the full bandwidth. The parity update overhead in RAID-5 is totally eliminated in other RAID architectures based on mirroring.

6.2. Effects of Disk Array Size

The disk array size is indicated by the total number of disks in the array. In Fig.8, we have a fixed array size of 16 disks. Now, we demonstrate the results with a fixed I/O rate of 320 MB for parallel read or parallel write, regardless of the disk array size. In Fig.9, we plot the measured aggregate I/O bandwidth against the disk array size, which varying from 2 to 16 disks. Again, all disk caches are bypassed in the experiment. Since the NFS server does not scale well with the increase of disks, it is not included in this performance experiment.



(a) Parallel read **(b) Parallel write**

Figure 9 Aggregate I/O bandwidths of four ds-RAIDs plotted against increasing disk array size under a fixed I/O request of 320 MB from 16 clients.

For parallel reads, the RAID-5, chained declustering, and RAID-x perform about the same and all scale well with increasing number of disks in the array. The RAID-10 lags far based for the same redundancy reasoning. For parallel writes, the ranking of the four changes sharply as shown in Fig. 9b. The RAID-x outperforms the other three disk arrays. For 16 disks, we observe the write bandwidths of 15.3 MB/s, 12.6 MB/s, 9.9 MB/s, and 6.8 MB/s for the RAID-x, chained declustering, RAID-10, and RAID-5, are respectively. The effects of the disk array size is very similar those of the I/O requests. In both cases, we consider the saturated case of accessing large files simultaneously. Essentially, we measure the asymptotic I/O bandwidth when the cluster network is utilized to its limit.

Table 3 shows the improvement factor of 16 clients over 1 client in using three 16-disk RAID architectures based on mirroring. We compare the RAID-x, chained declustering, and RAID-10 with the NFS. Now, the comparison separates the large read from small and large write situations. We ignore the case of small read, because it is uninteresting to the saturated conditions.

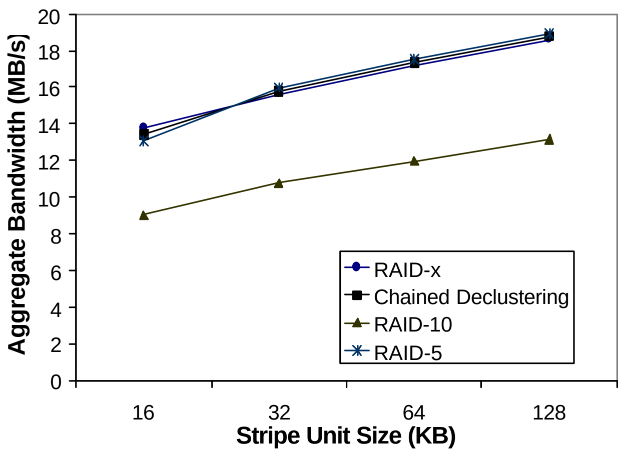
Table 3 Achievable I/O Bandwidth and Improvement Factor of Three Distributed Software RAIDs Against the NFS on the Trojans Cluster

I/O Operations	NFS			RAID-x		
	1 Client	16 Clients	Improve	1 Client	16 Clients	Improve
Large Read	2.58 MB/s	2.3 MB/s	0.89	2.59 MB/s	15.63 MB/s	6.03
Large Write	2.11 MB/s	2.77 MB/s	1.31	2.92 MB/s	15.29 MB/s	5.24
Small Write	2.47 MB/s	2.81 MB/s	1.34	2.35 MB/s	15.1 MB/s	6.43
Operations	**Chained Declustering**			**RAID-10**		
	1 Client	**16 Clients**	**Improve**	**1 Client**	**16 Clients**	**Improve**
Large Read	2.46 MB/s	15.8 MB/s	6.42	2.37 MB/s	10.76 MB/s	4.54
Large Write	2.62 MB/s	12.63 MB/s	4.82	2.31 MB/s	9.96 MB/s	4.31
Small Write	2.31 MB/s	12.54 MB/s	5.43	2.27 MB/s	9.98 MB/s	4.39

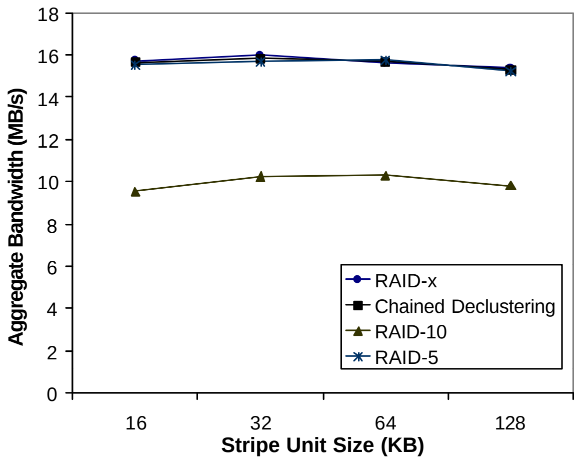
For parallel reads, chained declustering and RAID-x have almost equal performance and the RAID-10 lags far behind. The difference in improvement factor of larger or small writes become very noticeable among the four I/O subsystems. These differences are attributed to the sensitivity to architectural differences in ds-RAIDs being evaluated. Comparing with Berkeley xFS results, our 1-client bandwidth is quite high due to 16-way striping across the disk array. For this reason, the improvement factor is lower than that achieved by the xFS system. Again, the table entries demonstrate the highest improvement factors of the RAID-x over the RAID-10, chained declustering, and the NFS, especially in parallel write operations.

6.3 Effects of Stripe Unit Size

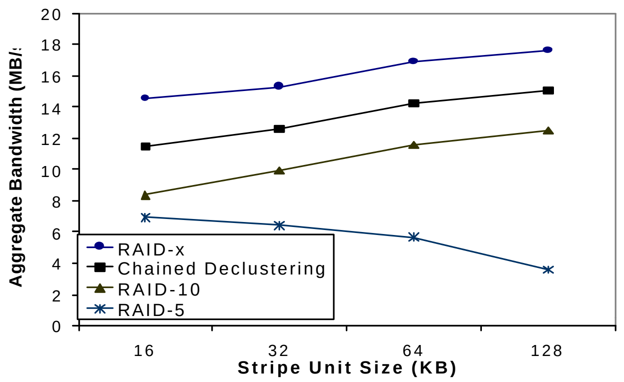
Figure 10 reveals the effects of stripe unit size on the aggregate I/O bandwidth performance of four ds-RAID architectures. We consider the stripe unit from 16 KB to 128 KB, assuming 16 client requests on 16 disks, simultaneously. Again the chained declustering and RAID-x have comparable read performance with RAID-10 lagging far behind. The bandwidth gap between RAID-10 and other RAIDs stays quite large for all stripe unit sizes. For large read (Fig.10a), all RAID architectures scale slowly with increasing block size. In the best case, 18.6 MB/s throughput was observed for RAID-x using 128-KB block size. For small read (Fig.10b), the optimal block size stays with 32 KB. The bandwidth gaps among the 4 RAIDs widens and none of them is scalable as seen by the flat curves.



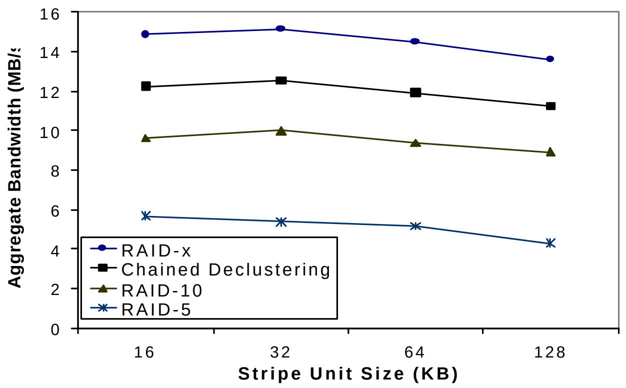
(a) Large read (320MB for 16 clients)



(b) Small read (512KB for 16 clients)



(c) Large write (320MB for 16 clients)



(d) Small write (512KB for 16 clients)

Figure 10 Effects of stripe unit size on the aggregate I/O bandwidth of three ds-RAID architectures

For large write (Fig.10c), RAID-x demonstrates much higher speed than those for chained declustering and RAID-10. The RAID-5 shows a declining speed with increasing block size. The top three all enjoy larger block size. With small write in Fig.10d, the optimal block size stays at 32 KB and the ranking order stays the same as large writes. However, the all show a slow-down trend as the block size increases. Note that the optimal block size varies with applications, disk types, workload conditions, and even cluster platforms used. These results certainly reflect the site factors associated with the USC Linux PC cluster using IDE disks and fast Ethernet connections.

7. Andrew and Bonnie Benchmark Results

What we have reported in the previous section are based on synthetic workloads, representing uniform distribution of parallel reads and parallel writes. The raw I/O bandwidth results depend on the workload characteristics. In this section, we consider commercially available benchmarks that are typical in file server and disk I/O applications. The Andrew benchmark tests the performance of network file systems such as NFS, AFS, xFS, etc. [23]. The Bonnie benchmark tests the software overhead for parallel I/O subsystems, including various RAID configurations.

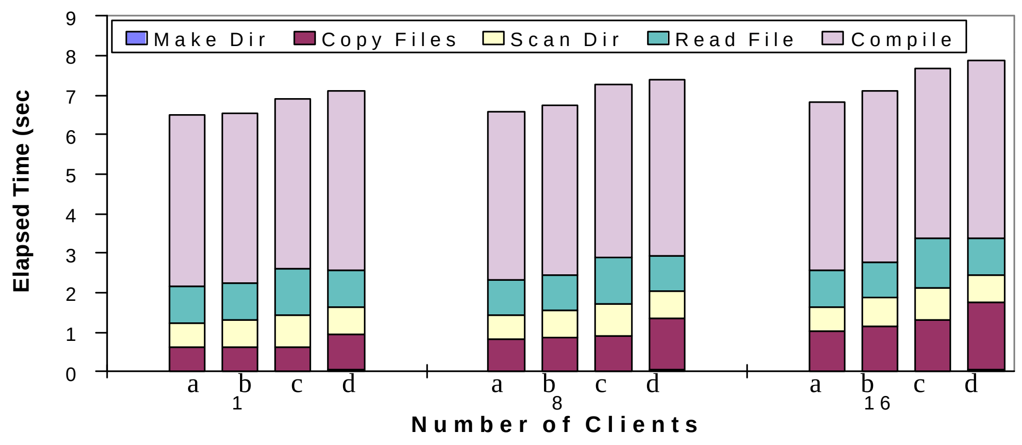
7.1. Andrew Benchmark Results

Andrew benchmark [18] is a popular file-testing program suite, often used to test the performance of file systems in a network environment. It was developed at Carnegie Melon University and has been applied to evaluate the Digital Petal virtual disk array [30] and the Berkeley LFS and xFS [1]. We execute the Andrew benchmark on four ds-RAID subsystems running on the Trojans cluster. We consider the effects of increasing number of client requests. Figure 11 measures the Andrew benchmark results on the Trojans cluster.

There are five phases in the Andrew benchmark. The first phase recursively creates subdirectories. The second phase measures the data transfer capabilities by copying files. The third phase recursively examines the status of every file. The fourth phase scans every byte of every file. The final phase compiles the files and links them together. The performance is indicated by the *elapsed time* in executing the Andrew benchmark on the target RAID configuration. These tests demonstrate how the underlying storage structures can affect the performance of the file system being supported. Each local file system mounts the “virtual” storage device provided by the CDD.

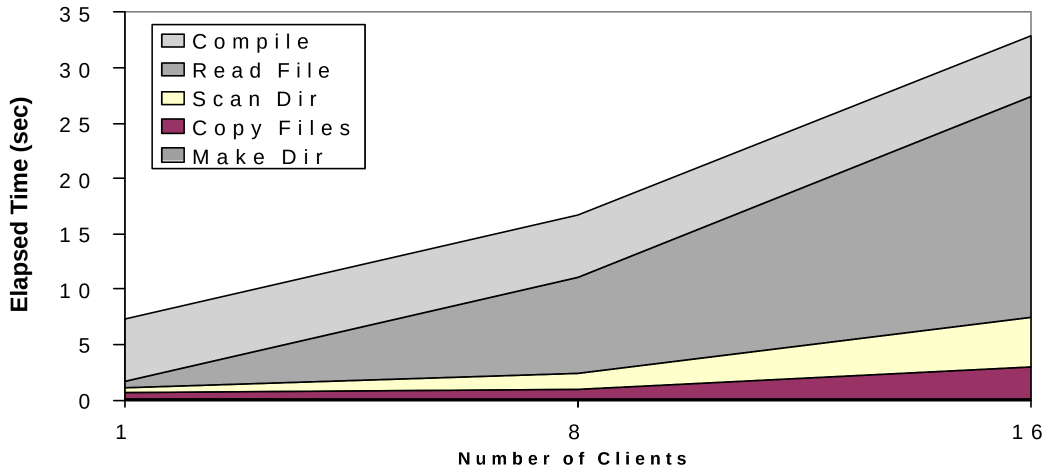
The number of disk I/O nodes is fixed at 16. Each client executes its private copy of Andrew benchmark. We use the CDD on each node to keep the metadata (*inode* in Fig.7) atomic. Figure 11a shows the elapsed times in executing the Andrew benchmark on four ds-RAID configurations. Figure 11b shows the corresponding time in using the NFS. The NFS time increases sharply with the number of clients, while the ds-RAID configurations are all scalable with almost constant elapsed time. The bar height in Fig.11a stays essentially constant, independent of the client number. For 16 clients, the total elapsed time of the Andrew benchmark on the RAID-x, chained declustering, RAID-10, and RAID-5 increases slowly from 6.8 sec., to 7.1 sec., 7.7 sec., and 7.9 sec., respectively. The same benchmark shows that the NFS requires 33 sec to complete the execution.

Examining the bar diagrams in Fig.11a, we realized that all four ds-RAIDs perform well, approximately at the same level. Their differences in performance are rather small. However, they rank in increasing elapsed times as RAID-x, chained declustering, RAID-10, and RAID-5. Among the four ds-RAIDs, majority of the time (approximately 56% - 62%) was spent on the Compile phase. The Read-File and Scan-Directory are both read operations, each consuming about 12% - 17% and 9% - 11% of the time, respectively. Theses time components stay essentially constant, when the client number increases. The time spent in Copy-Files does increase slightly with the client number, because parallel writes are performed here. The time to Make-Directory is very small and can be ignored.



a: RAID-x b: Chianed Declustering c: RAID-10 d: RAID-5

(a) Results of four ds-RAID architecture



(b) NFS results

Figure 11. Elapsed times of four ds-RAIDs to execute the Andrew benchmark on the Trojans cluster against the use of a central NFS.

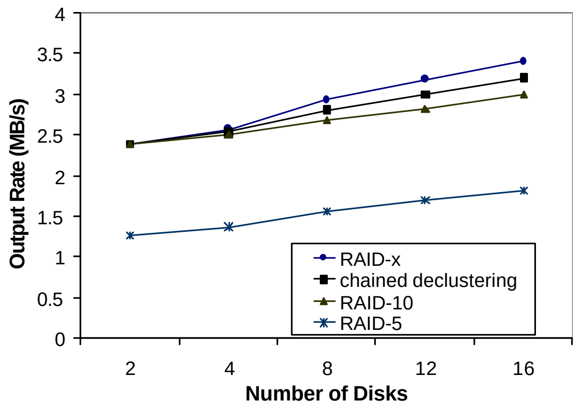
The NFS performance lags far behind, especially over a larger number of clients. The NFS shows a linear increase in times for Read-Files, Scan-Directory, and Copy-Files. The Compile time is still constant, because only CPU time is involved in the compile process. The time to Make-Directory is again too small to notice. As expected, the elapsed time of NFS increases sharply with increasing clients. The message being conveyed is that all ds-RAIDs perform much better than using the NFS.

All four ds-RAIDs show a slow increase in elapsed time with client traffic. This suggests that the performance of a ds-RAID in a serverless cluster is insensitive to the traffic intensity, especially when the workload is light. In other words, the Andrew benchmark implies that all four ds-RAIDs are scalable with the increase of I/O requests. Similar to those plotted in Fig.9, with fixed traffic density, all four ds-RAID configurations also scale well with the increase of disk array size. The ranking order of the four configurations is also similar to that shown in Fig.11a.

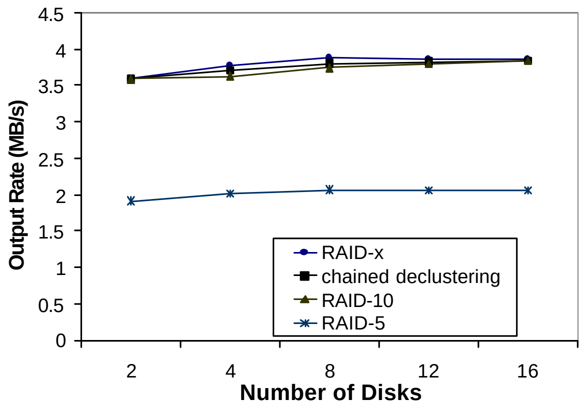
7.2. Bonnie Benchmark Results

The Bonnie benchmark performs a series of experiments, which intends to reveal the weight of software overhead incurred with parallel I/O operations. (Visit the Bonnie Benchmark Web Site: http://www.textuality.com/bonnie for details). These overheads include the times to use the file system and to access the low-level storage. The benchmark is composed of six parts. It writes data to the storage using character and block based I/O, reads, modifies, and rewrites the contents of the whole file, reads the file using character and block based I/O, and seeks into the file.

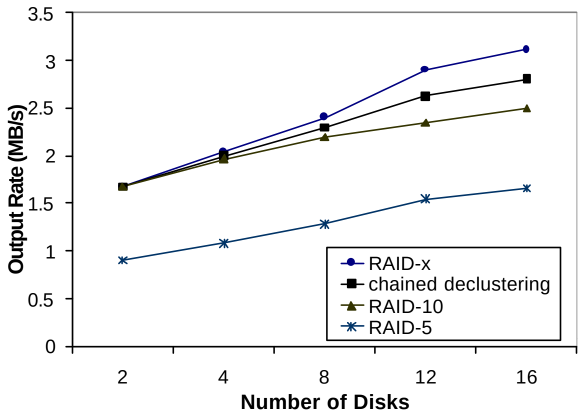
Bonnie reads and writes large disk files and produces a dense report. In our test, we used 500 MB data as the workload. The file size is large enough to avoid the buffer caching effects of the operating system so that the results reported could truly reflect the performance of the I/O storage. Bonnie benchmark reports either the *input rate*, the *output rate*, or the *seek rate* as shown in Fig.12.



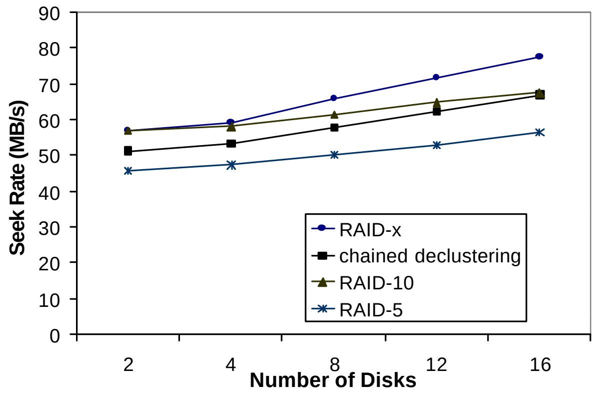
(a) Character write



(b) Block write



(c) File rewrite



(d) Random seek

Figure 12 Bonnie benchmark results of four I/O functions performed on four ds-RAID configurations in the USC Trojans cluster

These speed rates are expressed as MB/s. These speed results are plotted against increasing number of disks in the ds-RAID. The faster the data transfer rate, the less the software overhead experienced. The main work of the client is to write, or to read the 500MB-long file. It was the server's responsibility to do the real I/O. The server could be on a remote CDD. Thus very little CPU time is spent locally in handling distributed I/O operations. This is very different from the situation of accessing only local disks.

Figure 12a shows the results of a Character-Write test. In this test, a 500 MB long file is written to the storage byte-by-byte using the *putc*() function. This is a typical small-write operation. The RAID-x performs slightly better than the chained declustering RAID and the RAID-10. The write rate of the RAID-5 lags behind, because more software overhead due to parity update. The RAID-x was measured at 3.4 MB/s with 16 disks. RAID-10 achieved 2.9 MB/s due to the fact that only half of the disks providing useful bandwidth. The RAID-5 is the worst one, with a peak bandwidth of 1.8 MB/s. In all ds-RAIDs, the output rate increases only slightly with the disk array size.

Figure 12b shows the results of Block-Write test, where a 500MB-long file is written to the ds-RAID in blocks. Again the top three ds-RAID configurations have the same output rate with a peak 3.8 MB/s. This rate is almost independent of the disk array size. The RAID-5 has a rather flat low output rate (about 2 MB/s) due to excessive parity update overhead experienced. Block-Write (Fig.12b) results in better performance than the Character-Write in Fig.12a. This is because character-based operations introduce a considerable amount of overhead in system calls, and in reconstructing the block-based I/O requests from the scattered characters.

Figure 12c shows the output rate of the File-Rewrite test. In this test, the 500 MB-long file just created is read into the memory again with each block modified and then written back to the disks. The granularity of the files is much larger than those in character-write or block-write. Therefor, the output rate increases faster than those fine-grain disk accesses. The RAID-x still outperforms the others, chained declustering ranks the second, RAID-10 the third, and RAID-5 lags far behind.

Figure 12d shows the parallel seek rate of the Random-Seek operations. The benchmark program executes 4,000 seeks to random locations in the file. Of 10% of these seeks, the blocks that have been read are modified and rewritten to the disks. The seek rate measured increases sharply to as high as 77 MB/s. The speed gaps among the four ds-RAIDs are also getting closer, as the disk number increases. The ranking order changes slightly with 77 MB/s in RAID-x, RAID-10 and chained declustering RAID converging to the same speed of 67 MB/s, and RAID-5 at 56 MB/s for the 16-disk array configurations.

To summarize, the messages being conveyed in Bonnie benchmark experiments are twofold:

1. The measured I/O data rates in Figs. 12a, b, c are rather low, because of large software overhead experienced. These large overheads are attributed mainly to a large number of I/O system calls and the delays caused by the use of the slow TCP/IP protocol in the Trojans cluster. The actual disk access time is only a small fraction of the total execution time of the Bonnie benchmark. Therefore, Bonnie benchmark indicates primarily the weight of software overhead, rather the raw I/O data transfer rate measured in Section 6. The fact that the seek rate

is much higher in Fig.12d is attributed to a large number of parallel seeks performed simultaneously over a large number of stripe groups.

2. The Bonnie benchmark reveals the effects of disk access granularity on the RAID performance. The larger is the granularity in parallel write operations, the more scalable will be the RAID configuration applied. In Fig. 12, the access granularity increases from characters (Fig.12a) to blocks (Fig.12b) and to files (Figs.12c, d). When the software overhead grows faster than the increase of I/O bandwidth, the ds-RAID becomes less scalable as seen in Figs. 12a and 12b. For this reason. the scalability in large file accesses does improve in Figs. 12c, d.

8. Related Projects and Assessments

Our distributed software RAID-x project was influenced by the pioneering work at Berkeley [1][7][11] and at CMU [13]. We were also inspired by the AFRAID [39] and AutoRAID [47] projects at HP laboratories, by the TickerTAIP project [5] at Princeton, the Berkeley Tertiary Disk project [3][41], the chained declustering developed at the University of Wisconsin [19][32], and the Petal project [30] at the Digital Laboratories. These related projects are assessed below.

Our RAID-x design appeals especially for construction in a serverless cluster of computers. The major innovation lies in the orthogonal striping and mirroring architecture over distributed disks. Distributed disks are coordinated at the Linux kernel level, rather in the user space. Table 4 compares our RAID-x features with four parallel or distributed RAID projects. Key references in these projects are also identified in the table.

Table 4 Research Projects on Parallel or Distributed RAIDs

System Attributes	**USC Trojans RAID-x [24]**	**Princeton TickerTAIP [5]**	**Digital Petal [30]**	**Berkeley Tertiary Disk [3][41]**	**HP AutoRAID [47]**
RAID Architecture environment	Orthogonal striping and mirroring in a Linux cluster	RAID-5 with multiple controllers	Chained Declustering in Unix cluster	RAID-5 built with a PC cluster	Hierarchical with RAID-1 and RAID-5
Enabling Mechanism for SIOS	Cooperative device drivers in Linux kernel	Single RAID server implementation	Petal device drivers at user level	xFS storage servers at file level	Disk array within single controller
Data Consistency Checking	Locks at device driver level	Sequencing of user requests	Lamport's Paxos algorithm [28]	Modified DASH protocol [31] in the xFS file system	Use mark to update the parity disk
Reliability and Fault Tolerance	Orthogonal striping and mirroring	Parity checks in RAID-5	Chained Declustering	SCSI disks with parity in RAID-5	Mirroring and parity checks

8.1 TickerTAIP Project at Princeton

The TickerTAIP was jointly developed at HP laboratories and Princeton University [5]. It is a parallel RAID that distributes the controller functions across several loosely coupled processors. There is no central controller in this disk array: Instead, a set of cooperating array controllers was used to provide all the functions needed. The TickerTAIP applied a single RAID-5 server to achieve fault tolerance.

On the TickerTAIP, data consistency was slowly handled with the sequencing of user requests. In our opinion. The TickerTAIP was the very first project exploiting parallelism to alleviate the controller bottleneck problem in a centralized RAID. It is not a true ds-RAID design, which spread the disks across a cluster network. However, the TickerTAIP truly opened up the interest to develop distributed RAID using the software integration approach.

8.2 Petal Project in Digital Laboratory

The Petal project [30] offers a distributed storage system consisting of network-connected storage servers. This project was truly the very first ds-RAID implementing the concept of a shared virtual disk array. It was done with a global name space in a cluster environment. Petal developed device drivers to implement the SIOS at the user level, rather at the kernel level. Our CDD was developed at the kernel level, which make it possible to apply most available file systems without modification.

Lamport's Paxos algorithm [28] was used to implement the global state. Petal applies the chained declustering architecture. Another important contribution of Petal group was the development of a distributed file system, called Frangipani [43]. The purpose was to cooperatively manage a virtual disk using tokens to synchronize the disk accesses from all physically distributed disks. We feel that the adaptive sector grouping method reported in [25] will reduce the false effects and enhance the effective I/O bandwidth not only in Petal but also in any ds-RAID.

8.3 Tertiary Disk at Berkeley

Tertiary Disk from UC Berkeley [3][41] is a highly reliable storage system built with a PC cluster. So far this is the largest ds-RAID project ever built with 380 disks. Each cluster node is composed of two PCs with up to 14 shared disks. The shared disks are connected by doubly-ended SCSI disks to yield the high reliability. Tertiary Disk applies the RAID-5 architecture for data distribution. From fault-tolerance viewpoint, the Tertiary Disk offers higher fault-tolerance in the controller area than any of the four ds-RAID configurations being evaluated.

A log-structured serverless network file system, called the xFS [1], runs on top of Tertiary Disk. The xFS distributes storage, cache, and control over cluster nodes. It uses a modified directory-based cache coherence protocol [31] to maintain data consistency. The xFS offers the very first distributed file system for serverless Unix clusters. The project demonstrated high scalability in very large-scale database applications. It would be interesting to see that some of our RAID-x and CDD features (Sections 3-5) be integrated into the Tertiary Disk architecture.

8.4 AFRAID and Auto-RAID Projects

The AFRAID [39] assumes a RAID-5 architecture with relaxed coherency between data and parity blocks. Parity is made consistent in the idle periods between bursts of client writes. The stored data is frequently held redundant, rather all the time. This will alleviate the small-update penalty of RAID-at the expense of slightly increased data loss from disk failure. Inspired by AFRAID, the AutoRAID [47] group built a two-level hierarchy storage system under a single array controller. In the upper level, two copies of active data are stored in RAID-1. In the lower level, RAID-5 parity protection is applied. The AutoRAID can automatically manage the migration of data blocks between these two levels.

The AutoRAID eliminates the data loss risk by using two redundancy schemes jointly. All of the above projects on parallel RAID or on ds-RAIDs can be applied by both designers and users in parallel I/O applications on serverless clusters. From the benchmarking results reported in Sections 6 and 7, it is clear that the RAID-x provides much better solution to the small write problem associated with the RAID-5 architecture. For this reason, it would be interesting to adopt the RAID-x architecture in future upgrades of the existing ds-RAID projects.

9. Conclusions and Suggestions

In this final section, we summarize our research contributions. Then we discuss some of the outstanding issues and comment on meaningful directions for future research.

9.1 Summary of Contributions

The major contribution of this work lies in providing functional specifications, enabling mechanisms, and benchmark results of ds-RAID. The intention is to advance state of the art in ds-RAID architectures. Specifically, the contributions are summarized in four areas: a new distributed software RAID-x architecture, newly developed cooperative disk drivers to achieve the SIOS in Linux kernel, demonstrated I/O performance in benchmark results, and enabling new cluster applications using various ds-RAID configurations.

(1) The new RAID-x shows its strength in building large and distributed I/O storage for serverless PC or workstation clusters. The RAID-x is unique with its orthogonal striping and mirroring architecture, which exploits full-stripe bandwidth similar to what a RAID-10 can provide. Like RAID-5, RAID-x was designed to tolerate all single disk failures. These features are based on clustered mirroring, while orthogonal striping across the distributed disks in the cluster.

(2) We have developed new disk drivers at Linux kernel level to support efficient implementation of the SIOS in any ds-RAID configuration. These device drivers enable fast remote disk accesses and parallel disk I/O without using a central file server. Benchmark performance results show that our ds-RAID can achieve scalability, performance, and availability in general-purpose cluster computing applications.

(3) The raw I/O performance of the RAID-x is experimentally proven superior to the competing RAID architectures evaluated. For parallel reads with 16 active clients, the RAID-x achieved a 15.97 MB/s throughput, which is up to 1.5 times higher than using a chained-declustering RAID, RAID-10, and RAID-5. For parallel writes, RAID-x achieved 15.29 MB/s, which is 1.2 to 2.2 times higher than using other three RAIDs. The Andrew benchmark ranks RAID-x first with a 13% cut in

elapsed time, compared with those experienced on a RAID-10 or on a chained-declustering RAID.

(4) Some Linux extensions and middleware were developed to support the SIOS, shared virtual memory, global file management, and distributed checkpointing in clusters. Lots of I/O-centric applications can be generated on scalable and reliable clusters of computers. The scalable I/O bandwidth makes the RAID-x and other ds-RAIDs appealing to biological sequence analysis, collaborative engineering design, secure E-commerce, data mining, specialized digital libraries, and distributed multimedia processing [23] in any Unix cluster or Linux cluster environments.

9.2. Suggestions for Further Research

For furthers research, we suggest to attack the following outstanding issues towards the optimization, industrialization, and applications of ds-RAIDs in reliable cluster I/O processing.

(1) In our experiments, we have proved the scalability of the ds-RAID up to 16 disks. To scale up to a much larger number of disks, the communication protocol used in the CDD may become the performance bottleneck. The current hand shaking TCP/IP protocol could be replaced by a low-latency protocol, similar to that suggested by Martin, et al [33].

(2) Based on today's technology, a ds-RAID can be easily scaled to hundreds of disks. With 40 GB per disk, one can easily build a 20 TB disk array consisting of 512 disks. The underlying cluster network must be upgraded to provide the needed bandwidth. A distributed file system is desired to improve the efficiency of the CDD built in the RAID-x. A more efficient data consistency protocol will provide higher scalability to hundreds or even thousands of disks.

(3) An *adaptive sector grouping* (ASG) method was suggested in [25] for faster access of ds-RAID in a cluster environment. Grouped sector access of disks in a ds-RAID will reduce the false sharing effects in shared virtual disks. This topic is worthy of further experimentation to prove its effectiveness in accessing large ds-RAID configurations.

(4) In the work being reported, we did not apply any data prefetching [2][9][27][36][45] or cooperative caching techniques [9][11][20][26][35][44] to improve the performance of ds-RAID. Prefecting will further improve parallel read performance. Cooperative caching will further improve parallel write performance. In future efforts, these directions are highly recommended. Even higher parallel I/O performance would be expected, if these techniques were applied.

References:

[1] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. "Serverless Network File Systems", *ACM Trans. on Computer Systems*, Jan. 1996, pp.41-79.

[2] M. Arunachalam, A. Choudhary, and B. Rullman, "Implementation and Evaluation of Prefetching in the Intel Paragon Parallel File System", *Proceedings of the 10th International Parallel Processing Symposium (IPPS'96)*, April 1996, pp.554-559.

[3] S. Asami, N. Talagala, and D. A. Patterson, "Designing a self-maintaining storage system", *Proceedings of 16th IEEE Symposium on Mass Storage Systems*, March 1999, pp. 222-233.

[4] L. F. Cabrera, and D. E. Long, "Swift: Using Distributed Disk Striping to Provide High I/O Data Rates", *Proceedings of USENIX Computing Systems*, Fall 1991, pp.405-433.

[5] P. Cao, S. B. Lim, S. Venkataraman, and J. Wilkes, "The TickerTAIP Parallel RAID Architecture", *ACM Trans. on Computer System*, Vol.12, No.3, August 1994, pp.236-269.

[6] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters", *Proc. of the Extreme Linux Track: 4th Annual Linux Showcase and Conf.*, October 2000.

[7] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz and D. A. Patterson; "RAID: High-Performance, Reliable Secondary Storage", *ACM Computing Surveys*, Vol.26, No.2, June 1994, pp.145-185.

[8] P. F. Corbett, D. G. Feitelson, J.-P. Prost, and S. J. Baylor. "Parallel Access to Files in the Vesta File System". *Proceedings of Supercomputing'93*, 1993.

[9] T. Cortes, "Software RAID and Parallel Filesystems", in *High Performance Cluster Computing–Architectures and Systems*, Rajkumar Buyya (ed.), Prentice Hall PTR, 1999, pp.463-496.

[10] T. H. Cormen and D. Kotz, "Integrating Theory and Practice in Parallel File Systems", *Proceedings of DAGS '93 Symposium*, June 1993, pp. 64-74.

[11] M. Dahlin, R. Wang, T. Anderson, D. Patterson, "Cooperative Caching: Using Remote Client Memory to Improve File System Performance". *Proceedings of Operating System Design and Implementation*, 1994.

[12] I. Foster, D. Kohr, Jr., R. Krishnaiyer, and J. Mogill, "Remote I/O: Fast Access to Distant Storage". *Proc. of the Fifth Workshop on I/O in Parallel and Distributed Systems*, November 1997, pp.14-25.

[13] G. Gibson, D. Nagle, K. Amiri, F. Chang, H. Gobioff, E. Riedel, D. Rochberg and J. Zelenka, "A Cost-effective, High-bandwidth Storage Architecture", *Proc. of the 8th Conf. on Architectural Support for Programming Languages and Operating Systems*, 1998.

[14] M. Harry, J. M. del Rosario, and A. Choudhary, "VIP-FS: a VIrtual, Parallel File System for High Performance Parallel and Distributed Computing", *Proceedings of the 9th International Parallel Processing Symposium (IPPS'95)*, April 1995, pp. 159-164.

[15] J. H. Hartman, I. Murdock, and T. Spalink, "The Swarm Scalable Storage System", *Proceedings of the 19th IEEE International Conf. on Distributed Computing Systems (ICDCS '99)*, June 1999.

[16] J. H. Hartman, and J. K. Ousterhout, "The Zebra Striped Network File System", *ACM Transactions on Computer System*, Vol.13, No.3, Aug. 1995, pp.274-310.

[17] R. S. Ho, K. Hwang, and H. Jin, "Design and Analysis of Clusters with Single I/O Space", *Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, April 2000, Taiwan, pp.120-127.

[18] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West, "Scale and Performance in a Distributed File System". *ACM Trans. on Computer System*, Vol.6, No.1, pp.51-81, February 1988.

[19] H. I. Hsiao and D. DeWitt, "Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines", *Proc. of 6th International Data Engineering Conf.*, 1990, pp.456-465.

[20] Y. Hu, Q. Yang and T. Nightingale. "RAPID-Cache --- A Reliable and Inexpensive Write Cache for Disk I/O Systems", *Proceedings of the 5th International Symposium on High Performance Computer Architecture (HPCA-5)*, Orlando, Florida, Jan. 1999, pp. 204 – 213.

[21] J. Huber, C. L. Elford, D. A. Reed, A. A. Chien, and D. S. Blumenthal, "PPFS: A High Performance Portable Parallel File System", *Proceedings of the 9th ACM International Conference on Supercomputing*, Barcelona, July 1995, pp.385-394.

[22] K. Hwang, H. Jin, E. Chow, C. L. Wang, and Z. Xu. "Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space". *IEEE Concurrency Magazine*, March 1999, pp.60-69.

[23] K. Hwang and Z. Xu. *Scalable Parallel Computing: Technology, Architecture, Programming*. McGraw-Hill, New York, 1998.

[24] K. Hwang, H. Jin, and R. Ho, "A New Distributed RAID-x Architecture for I/O-Centric Cluster

Computing”, *Proc. of 9th IEEE High-Performance Distributed Computing (HPDC-9) Symposium*, Pittsburgh, August 1-4, 2000, pp.279-286.

[25] H. Jin and K. Hwang, “ Adaptive Sector Grouping to Reduce False Sharing of Distributed RAID in PC/Workstation Clusters”, to appear in *Cluster Computing Journal*, 2001.

[26] J. H. Kim, S. W. Eom, S. H. Noh, and Y. H. Won, “Striping and Buffer Caching for Software RAID File Systems in Workstation Clusters”, *Proceedings of 19th IEEE International Conference on Distributed Computing Systems*, 31 May-4 June 1999, pp. 544 - 551.

[27] D. Kotz and C. S. Ellis, “Practical Prefetching Techniques for Parallel File Systems”, *Proceedings of the First Int’l Conf. on Parallel and Distributed Information Systems*, Dec. 1991, pp.182-189.

[28] L. Lamport, “The Part-Time Parliament”, *Technical Report 49*, Digital Corp., Sept. 1989.

[29] E. K. Lee, and R. H. Katz, “The Performance of Parity Placements in Disk Arrays”, *IEEE Transactions on Computers*, Vol.42, No.6, June 1993, pp.651-664

[30] E. K. Lee and C. A. Thekkath. “Petal: Distributed Virtual Disks”. *Proceedings of the Seventh International Conf. on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, October 1996, pp.84-92.

[31] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy, “The DASH prototype: Logic overhead and performance”, *IEEE Transactions on Parallel and Distributed Systems*, Vol.4, No.1, Jan. 1993, pp.41-61.

[32] C.-S. Li, M.-S. Chen, P. S. Yu and H.-I. Hsiao, “Combining Replication and Parity Approaches for Fault-Tolerant Disk Arrays”, *Proceedings of Sixth IEEE Symposium on Parallel and Distributed Processing*, Oct. 1994, pp. 360 – 367.

[33] R. P. Martin, A. M. Vahdat, D. E. Culler, T. E. Anderson, “Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture”, *Proc. of the 24th Annual International Symp. on Computer Architecture*, June 1997, pp.85-97.

[34] N. Nieuwejaar and D. Kotz, “Performance of the Galley Parallel File System”. *Proceedings of the Fourth Workshop on I/O in Parallel and Distributed Systems*, pp.83-94, Philadelphia, May 1996.

[35] B. Nitzberg and V. Lo, “Collective Buffering: Improving Parallel I/O Performance”, *Proceedings of the Sixth IEEE International Symposium on High Performance Distributed Computing* (HPDC-6), Portland, OR, August 1997, pp.148-157.

[36] R. H. Patterson, G. Gibson, E. Ginting, D. Stodolsky, J. Zelenka, “Informed Prefetching and Caching”, *Proc. of the 15th Symposium of Operating Systems Principles*, 1995, pp.79-95

[37] G. F. Pfister. “The Varieties of Single System Image”, *Proceedings of IEEE Workshop on Advances in Parallel and Distributed System*, IEEE CS Press, 1993, pp.59-63.

[38] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, “Design and Implementation of the Sun Network Filesystem”, *Proc. of the USENIX Conference*, June 1985, pp.119-130.

[39] S. Savage and J. Wilkes, “AFRAID -- A Frequently Redundant Array of Independent Disks”, *Proc. of 1996 USENIX Technical Conference*, January 1996, pp. 27-39.

[40] M. Stonebraker and G. A. Schloss, “Distributed RAID – a New Multiple Copy Algorithm”, *Proc. of the Sixth International Conf. on Data Engineering*, Feb. 1990, pp.430-437.

[41] N. Talagala, S. Asami, D. Patterson, and K. Lutz, “Tertiary Disk: Large Scale Distributed Storage”, *UCB Technical Report No. UCB//CSD-98-989.*

[42] R. Thakur, W. Gropp, and E. Lusk. “An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces”. *Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation*, October 1996, pp.180-187.

[43] C. A. Thekkath, T. Mann, and E. K. Lee. “Frangipani: A Scalable Distributed File System”. *Proceedings of ACM Symp. of Operating Systems Principles*, Oct. 1997, pp.224-237.

[44] R. Treiber, and J. Menon, “Simulation Study of Cached RAID5 Designs”, *Proceedings of First IEEE Symposium on High-Performance Computer Architecture*, Jan. 1995, pp. 186–197.

[45] P. J. Varman, and R. M. Verma, "Tight Bounds for Prefetching and Buffer Management Algorithms for Parallel I/O Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol.10, No.12, Dec. 1999, pp. 1262 – 1275.

[46] R. W. Watson, and R. A. Coyne, "The Parallel I/O Architecture of the High Performance Storage System (HPSS)", *Proc. of the 14th IEEE Symp. on Mass Storage Systems*, Sept. 1995, pp.27-44.

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

Biographical Sketches

Kai Hwang is a Professor of Electrical Engineering and Computer Science at the University of Southern California. He has engaged in higher education and computer research for 28 years, after earning the Ph.D. degree from the University of California at Berkeley. An IEEE Fellow, he specializes in computer architecture, digital arithmetic, and parallel processing. He is a founding Editor of the *Journal of Parallel and Distributed Computing.* In 1996, he received the Outstanding Achievement Award from the Association of Parallel and Distributed Processing Technology and Applications.

Dr. Hwang has published six books and over 170 technical papers in computer science and engineering, lectured worldwide on computer and information technology, and performed consulting and advisory work for IBM, MIT Lincoln Lab., ETL in Japan, CERN Computing School in the Netherlands, ITRI in Taiwan, and GMD in Germany. His current research interest lies in network-based PC or workstation clusters, Internet security architectures, and middleware support for security, availability, and single-system-image services in clusters. He can be reached by Email: kaihwang@usc.edu.

Hai Jin is a Professor of Computer Science at Huazhong University of Science and Technology (HUST) in China. He received his Ph.D. in computer engineering from HUST in 1994. In 1996, he was awarded the scholarship by German Academic Exchange Service (DAAD) at Technical University of Chemnitz in Germany. He has worked at the University of Hong Kong, where he participated in the HKU Cluster project. Presently, he works as a visiting scholar at the Internet and Cluster Computing Laboratory at the University of Southern California.

Dr. Jin is a member of IEEE and ACM. He chairs the *2001 International Workshop on Cluster Infrastructure on Web Server and E-Commerce Applications (CISCA'01)* and *First International Workshop on Internet Computing and E-Commerce (ICEC'01).* He served as program vice-chair of *2001 International Symposium on Cluster Computing and Grid (CCGrid'01)*. He has co-authored four books and published more than 30 papers. His research interests cover parallel I/O, RAID architecture, fault tolerance, and cluster computing. Contact him at hjin@ceng.usc.edu.

Roy S. Ho received the M.Phil. degree in May 2000 from the Computer Science and Information Systems Department at the University of Hong Kong (HKU), China. He received the B.S. degree in Computer Engineering from the HKU in 1988. His work on this project started his Master thesis work at the HKU and completed at the University of Southern California in early 2000. His current research interest lies in computer architecture and scalable cluster computing. He can be reached by Email: scho@csis.hku.hk.