# Orthogonal Striping and Mirroring in Distributed RAID for I/O-Centric Cluster Computing

Kai Hwang, *Fellow*, *IEEE*, Hai Jin, *Member*, *IEEE*, and Roy S.C. Ho

**Abstract**—This paper presents a new distributed disk-array architecture for achieving high I/O performance in scalable cluster computing. In a serverless cluster of computers, all distributed local disks can be integrated as a *distributed-software redundant array of independent disks* (ds-RAID) with a single I/O space. We report the new RAID-x design and its benchmark performance results. The advantage of RAID-x comes mainly from its *orthogonal striping* and *mirroring* (OSM) architecture. The bandwidth is enhanced with distributed striping across local and remote disks, while the reliability comes from orthogonal mirroring on local disks at the background. Our RAID-x design is experimentally compared with the RAID-5, RAID-10, and chained-declustering RAID through benchmarking on a research Linux cluster at USC. Andrew and Bonnie benchmark results are reported on all four disk-array architectures. Cooperative disk drivers and Linux extensions are developed to enable not only the single I/O space, but also the shared virtual memory and global file hierarchy. We reveal the effects of traffic rate and stripe unit size on I/O performance. Through scalability and overhead analysis, we find the strength of RAID-x in three areas: 1) improved aggregate I/O bandwidth especially for parallel writes, 2) orthogonal mirroring with low software overhead, and 3) enhanced scalability in cluster I/O processing. Architectural strengths and weakness of all four ds-RAID architectures are evaluated comparatively. The optimal choice among them depends on parallel read/write performance desired, the level of fault tolerance required, and the cost-effectiveness in specific I/O processing applications.

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

## 1 Introduction

Many cluster computing tasks are I/O-centric such as transaction processing, pervasive computing, and E-commerce applications [21]. We present a new distributed architecture for building *Redundant Array of Independent Disks* (RAID) [7] to solve the collective I/O problem in PC or workstation clusters. We concentrate on serverless cluster of computers in which no central server is used. The client-server architecture does not apply here. Instead, all cluster nodes must divide the storage and file management functions in a distributed fashion [1].

A serverless cluster demands a *distributed software RAID* (henceforth ds-RAID) architecture, which embodies dispersed disks physically attached to client hosts in the cluster [20]. A ds-RAID is also known as a *distributed* RAID [30] or a *software* RAID [9] 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.

In the past, the Princeton TickerTAIP project [5] offered a *parallel* RAID architecture for supporting parallel disk I/O with multiple controllers. However, the TickerTAIP was implemented as a centralized I/O subsystem. The HP AutoRAID [35] was built as a hierarchy of RAID-1 and RAID-5 subsystems. These two disk arrays are not really distributed in nature. The distributed RAID concept was originally explored by Stonebraker and Schloss [30]

Prototyping of ds-RAID started with the Petal project [25] at Digital Laboratories and with the Tertiary Disk project [3] at UC Berkeley. The Petal was built with a disk-mirroring scheme, called chained declustering [17]. The Tertiary Disk adapted a parity-checking RAID-5 architecture, using the serverless xFS file system [1]. Our RAID-x was built in the USC Trojans cluster project. The level x in RAID-x is yet to be assigned by the RAID Advisory Board.

Single I/O space for a cluster was first treated in [20]. Different aspects of this work have been presented in two IEEE Conferences: The 20th International Conference on Distributed Computing Systems paper [15] covers various single I/O issues and the International Symposium on High Performance Distributed Computing paper [22] shows some preliminary benchmark results of RAID-x. A grouped-sector approach was suggested in [23] for fast access of a ds-RAID subsystem. This journal paper provides a comprehensive treatment of the RAID-x architecture supported by extensive benchmark results. We compare RAID-x with three known ds-RAID architectures for cluster I/O processing.

Besides presenting architectural innovations, we report benchmark results on RAID-x, RAID-5, RAID-10, and chained-declustering RAID. To build a ds-RAID, one must establish three fundamental capabilities: 1) a single I/O space for all disks in the cluster, 2) high scalability,

- *K. Hwang is with the Department of Electrical Engineering and Computer Science, University of Southern California, Los Angeles, CA 90089. E-mail: kaihwang@usc.edu.*
- *H. Jin is with the Department of Computer Science, Huazhong University of Science and Technology, Wuhan, China. E-mail: hjin@hust.edu.cn.*
- *R. Ho is with the Department of Computer Science and Information Systems, University of Hong Kong, Hong Kong, China. E-mail: scho@csis.hku.hk.*

*Manuscript received 25 Sept. 2000; revised 21 Feb. 2001; accepted 13 July 2001.*
*For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 112907.*

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 | RAID 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 system |

availability, and compatibility with I/O-centric cluster applications, and 3) local and remote disk I/O operations with reduced 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 *system-area network* or a *local-area network*. It is up to the communication middleware, file management, and operating system extensions that glues the distributed disks together for collective I/O operations. The quality of such a ds-RAID is highly relevant to network bandwidth, disk characteristics, and software overhead involved. We will attack these issues in the paper.

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 $B$ is the maximum bandwidth per disk. $R$ and $W$ stand for the *average read* or *write times* (latencies) per block of 32KB. Normally, $W$ is slightly longer than $R$, both in the order of a few ms based on the disk technology. The *file size*, denoted by $m$, is the number of blocks in the file being read or written.

The rest of the paper is organized as follows: Section 2 introduces three approaches to building single I/O space in a ds-RAID embedded in a cluster. Several related parallel and distributed RAID projects are discussed. Section 3 introduces the orthogonal architecture and functions. Section 4 specifies the detailed RAID-x design and benchmark experiments performed in the USC Trojans cluster. Section 5 describes the architecture of the cooperative disk drivers, which enables the single I/O space. The performance results are presented in Sections 6-10.

Through Andrew benchmark in Section 6, we compare the relative performance of four ds-RAIDs against the central NFS approach. In Section 7, we report the traffic effects on the aggregate I/O bandwidth in clusters. The effects of distributed striping are discussed in Section 8. Section 9 analyzes the scaling effects on cluster I/O performance. Bonnie benchmark results are presented in Section 10 to reveal the software and network overheads experienced. Finally, we summarize the research contributions, identify some shortcomings, and suggest directions for further research.

## 2 Distributed Software RAID Architectures

First of all, we characterize the concept of single I/O space in distributed RAID and the corresponding file systems in network-based clusters. Then, we present the design objectives of ds-RAID. To assess state-of-the-art distributed storage subsystems, we compare five related parallel and distributed disk array projects.

### 2.1 Single I/O Space

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 device [20]. Such a collection of distributed disks forms 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) [28] 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. [20]. 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.

There are three advantages to have the SIOS in a 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 lightweight driver process. Second, the SIOS supports a persistent programming paradigm. It facilitates the development of SSI services such as *distributed shared memory* (DSM) and *global file hierarchy* [15]. All I/O devices and their physical locations are transparent to all users. Third, the SIOS allows striping across distributed disks. This will accelerate parallel I/O operations in clusters.

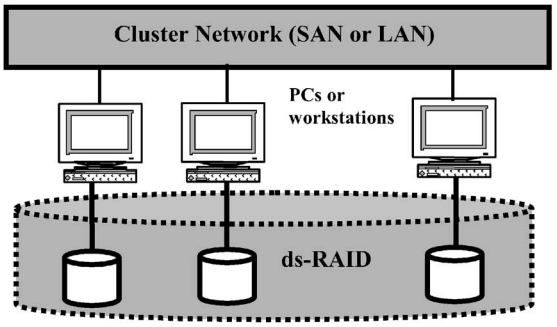


Fig. 1. A distributed software RAID with a single I/O space embedded in a serverless cluster of computers.

TABLE 2
Related Research Projects on Parallel or Distributed RAID

| System Attributes | USC RAID-x [22] | Princeton TickerTAIP[5] | Digital Petal Project [25] | Berkeley [3, 31] Tertiary Disks | HP AutoRAID [35] |
|---|---|---|---|---|---|
| **RAID Architecture environment** | Orthogonal striping/mirroring in a Linux cluster | RAID-5 with multiple controllers | Chained De-clustering in a 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 with its own I/O space | Petal device drivers at user level | xFS storage servers designed at the file level | Disk array with a single array controller |
| **Data Consistency Checking** | Locks at device driver level | Sequencing of user requests | Lamport's Paxos algorithm | Directory-based protocol 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 |

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], and the remote I/O project [11] are typical examples of user-level SIOS. This approach has two shortcomings: First, users have to apply specific APIs and identifiers to explore parallelism in I/O. Second, system calls are needed to perform network file accesses, which is rather slow to meet realtime or cluster computing requirements.

### 2.2 Distributed File Systems

A *distributed file system* achieves the SIOS across distributed storage servers [8], [13], [19]. 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 [32] 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 some well-established operating systems.

What we want to achieve is a SIOS supported by device drivers at the kernel level. The SIOS applies to both users and the file system used. Digital Petal [25] developed a device driver at the user space to enable remote I/O accesses. All physically distributed disks are viewed as a single large virtual disk. Each virtual disk is accessed as if it is a local disk. The Petal approach has the difficulty to optimize the file distribution. Many device drivers have their own custom-designed file system to optimize the performance. Petal project has later developed such a distributed file system, called Frangipani [32].

### 2.3 Design Objectives of ds-RAID

Four objectives are identified below for the SIOS design in a ds-RAID. These objectives are applicable to any ds-RAID architecture, not necessarily restricted to the new RAID-x being introduced.

1. ***A Single Address Space for All Disks in the Cluster.*** We build the SIOS by designing a special device driver for use in local disks. The drivers 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 Support.*** 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, have also deployed some specific features for availability enhancement.
3. ***Performance and Size Scalability.*** There is no central server in our cluster design and the drivers maintain a peer-to-peer relationship among themselves. The size scalability must be guaranteed in a serverless cluster design. Furthermore, the data striping running across distributed disks leads to some performance gains in benchmark experiments.
4. ***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.

### 2.4 Relevant Research Projects

Table 2 compares our RAID-x features with four parallel or distributed RAID projects. The TickerTAIP was jointly developed at HP laboratories and Princeton University [5]. It is a parallel RAID with multiple controllers. The TickerTAIP applied a single RAID-5 server to achieve fault tolerance. Data consistency was slowly handled with the sequencing of user requests. This was the first parallel disk array project exploiting parallelism at the controller level in a centralized RAID.

The Petal project [25] offers a distributed storage system consisting of network-connected storage servers. The 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.

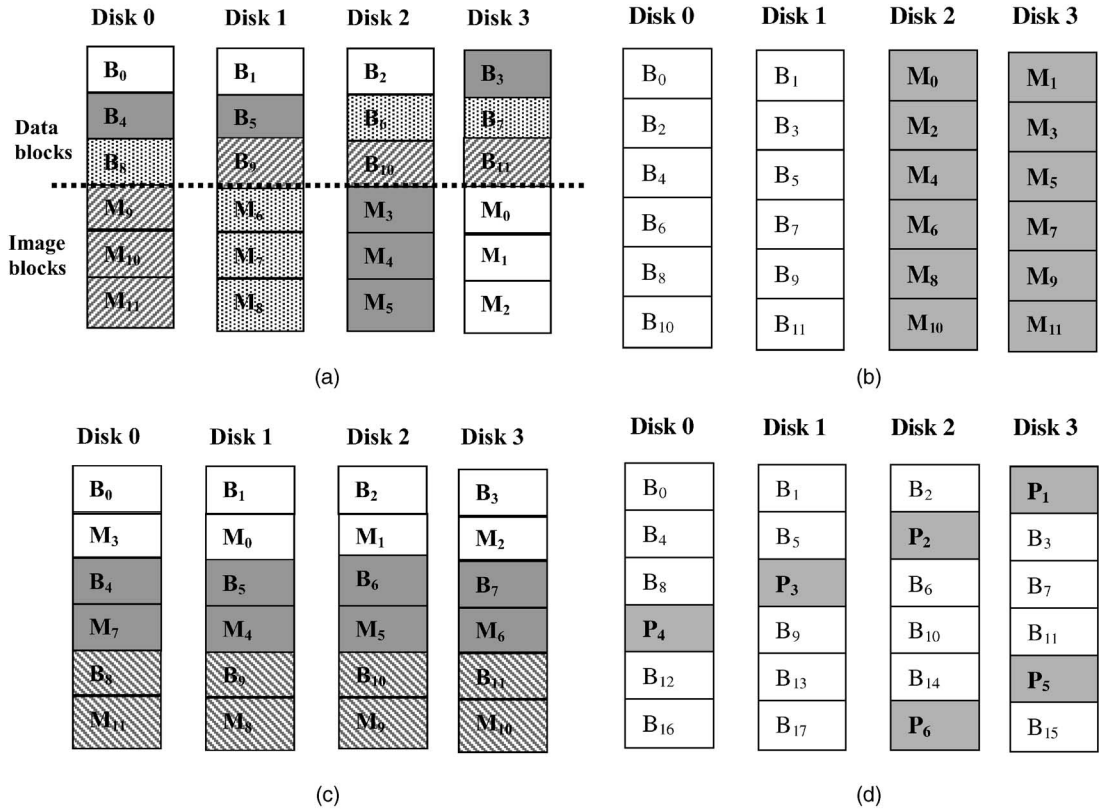


Fig. 2. The new RAID-x architecture (a) compared with three known RAID architectures (b, c, and d) for building distributed RAID subsystems. (a) Orthogonal striping and mirroring (OSM) in the RAID-x, (b) data mirroring in RAID-10, (c) skewed striping in a chained-declustering RAID, and (d) parity checking in RAID-5.

Petal developed device drivers to implement the SIOS at the user level, rather at the kernel level. Our disk driver was developed at the kernel level, which can apply most available file systems without modification. Petal applies the chained declustering architecture. An important contribution of the Petal disk was the development of a distributed file system, called Frangipani [32], to manage its distributed disks.

Tertiary Disk from UC Berkeley [3], [31] 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 higher reliability. Tertiary Disk applies the RAID-5 architecture for data distribution. From fault tolerance viewpoint, the TertiaryDisk offers higher fault tolerance in the controller area than any of the four ds-RAID configurations 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 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. The AutoRAID [35] was built as a hierarchical storage consisting of a RAID-1 on top of a redundant RAID-5 in two levels. Other related works appeared in [4], [12], [14], [26], [34].

## 3 Orthogonal Striping and Mirroring

The new RAID-x architecture is based on the *orthogonal striping and mirroring* (OSM) described in this section. We compare the RAID-x with three known RAID architectures. Then, we discuss the peak bandwidth and assess the fault tolerance of these disk arrays. All four ds-RAID architectures have been implemented on the Linux cluster at USC.

### 3.1 The Orthogonal Architecture

In Fig. 2, the *data blocks* are denoted as $B_i$ in the white boxes. The *mirrored images* are marked with $M_i$ in shaded boxes. The orthogonal mapping of data blocks and their images in RAID-x is illustrated in Fig. 2a. The data blocks are striped across the disks horizontally on the top half of the disk array. Their images are clustered together and copied in a single disk vertically. All image blocks occupy the lower half of the disk array.

This horizontal striping and vertical mirroring constitute the *orthogonality property*. Four data stripes and their images are illustrated in Fig. 2a by four different shading patterns. On a RAID-x, the images can be copied and updated at the background, thus reducing the access latency. Consider the top data stripe consisting of three blocks: $B_0$, $B_1$, and $B_2$. Their images, $M_0$, $M_1$, and $M_2$, are stored in Disk 3. Similarly, the images of the second stripe, $B_3$, $B_4$, and $B_5$, are mapped to Disk 2, etc.

The rule of thumb is that *no data block and its image should be mapped to the same disk*. Full bandwidth can be also achieved by reading or writing across all disks per each row

TABLE 3
Theoretical Peak Performance of Distributed 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*-1) *B* | *n B* | *n B* |
| | Small Write | *n B* | (*n*-1)*B*/2 | *n B* | *n B* |
| Parallel Read/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*-1) | 2 *mW / n* | *mW / n* + *mW / n*(*n*-1) |
| | Small Write | 2*W* | *R*+*W* | 2*W* | *W* |
| Max. Fault Coverage | | *n*/2 disk failures | Single disk failure | *n*/2 disk failures | Single disk failure |

in Fig. 2a. For example, all four blocks ($B_0$, $B_1$, $B_2$, $B_3$) in each row of Fig. 2a forms a data stripe. Their images ($M_0$, $M_1$, $M_2$, $M_3$) are written into disk 2 and disk 3 in a delayed time at the background.

### 3.2 Orthogonal Mapping of Data Blocks and Their Images

Formally, two mapping functions are given below to specify how to map the data blocks and their images to physical disk blocks orthogonally. 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 image block of $d$ is mapped to the $i$th disk and $j$th block, where

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

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

The orthogonality enables faster background copying of data images in the lower half of the disk array, while the striping is in progress. Like chained declustering (Fig. 2c), the mapping of data blocks and their images can be also interleaved. But we prefer to separate the data blocks and their images at the upper and lower halves of the array. This separation offers the advantage of sequential writes of all image blocks from the same stripe to a single disk. The striping and mirroring are thus done simultaneously.

### 3.3 Competing RAID Architectures

Fig. 2b, Fig. 2c, and Fig. 2d show three competing RAID architectures for building distributed RAID subsystems. The RAID-10 duplicates each data block on an extra disk without striping, making it easy to implement (Fig. 2b). The chained-declustering RAID [17] allows a skewed striping of image blocks (Fig. 2c). All mirroring schemes shown in Fig. 2a, Fig. 2b, and Fig. 2c have 50 percent redundancy. The RAID-5 does not apply mirroring at all. Instead, it uses a redundant disk block per each stripe (each row in Fig. 2d) as the parity check for that stripe.

In Table 3, parallel I/O 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. Parallel writes, RAID-10, and chained declustering require doubling the number of disk accesses. In RAID-x, the image blocks are clustered written into the same disk. That is, $m/n(n-1)$ images are written together to each disk. The large write latency is reduced to $mW/n + mW/n(n-1)$.

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 a ds-RAID-5 is $(n-1)B/2$ 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.

### 3.4 Peak I/O Bandwidth

Table 3 lists the theoretical peak I/O performance of four RAID architectures. The *maximum I/O bandwidth* reflects the ideal cases of parallel reads or parallel writes of all data blocks in the disk array.

The peak I/O bandwidth excludes the effects of caching, software overhead, parity or mirroring penalties, or network delays. In reality, the measured bandwidth of a ds-RAID would be much lower than the peak values presented in Table 3. In the best case, all four RAID architectures can deliver a maximum I/O bandwidth of $nB$. For RAID-5, each stripe-write is an atomic operation. The next stripe starts to write only after the previous stripe-write finishes. So, the maximum I/O bandwidth for large write on RAID-5 is $(n-1)B$. For small writes, the RAID-5 yields a peak bandwidth of $(n-1)B/2$ due to excessive parity update overhead. This shortcoming of ds-RAID-5 becomes more apparent in later sections.

### 3.5 Backgroud Image Writing

The advantage of RAID-x over other ds-RAID architectures comes mainly from this orthogonal mapping of data and image blocks to distributed disks. This overlapped foreground and background writes of data and their images are done simutaneously. Even other mirroring RAID can also take advantage of this property, but not as aggresively as RAID-x.

Our RAID-x takes only $W$ time to write all data blocks involved. The chained-declustering RAID needs to write a data stripe and its image in $2W$ time because interleaved data and images must be written, separately, without the property of orthogonality. The writing of the image blocks

can also be done later when all stripe images are loaded. This delayed writing is done at the background, overlapping with the regular data writes.

### 3.6 Fault Tolerance

The bottom row of Table 2 shows the maximum number of disk failures that each RAID architecture can tolerate. The RAID-x can tolerate all single-disk failures, same as that of RAID-5. Both RAID-10 and chained-declustering RAID are more robust than RAID-x and RAID-5. Thus, the RAID-x matches RAID-5 in terms of fault tolerance. The RAID-x matches the costs of RAID-10 and the chained declustering in terms of redundancy.

Among the four architectures, the RAID-5 achieves single fault tolerance with a much lower cost for using a fewer redundant disk space. The write bandwidth gain of RAID-x comes at some loss in *availability* from the chained-declustering RAID. The chained-declustering certainly has the advantage of higher reliability, which can tolerate up to $n/2$ disk failures, while the RAID-x tolerates only a single disk failure. However, the advantage of RAID-x lies in its much improved write performance, which is attributed to the property of orthogonality.

### 3.7 Aggressive Implementation

To implement a ds-RAID aggressively, one must conecrn about whether the image blocks are mapped contiguously in the same disk. If so, they can be written to the disk in a pipelined fashion. The mapping of image blocks in RAID-x follows this principle. This cannot be applied to other RAID architectures because successive image blocks are physically separate to take noncontiguous addresses on different disks. Therefore, they cannot be implemented as aggresively as the RAID-x.

The RAID-x architecture has no side effects on seek time and disk cache hit ratio. The results on the Random-Seek test in the Bonnie benchmark will verify this claim in Section 10. For delayed image writing, we maintain a buffer file to record all delayed blocks. When all delayed blocks are written to the disks, this buffer is cleared. If a disk failure occurs during delayed image writing, the lost data blocks are recovered from this buffer. By so doing, the reliability is preserved, even within a short delay period. Comparing with the longer disk writing time, the overhead to maintain this buffer can be neglected.

### 3.8 Advantages and Weakness

To sum up, the major strength of RAID-x lies in its much improved write performance than the competing RAID architectures. In terms of fault tolerance, the RAID-x matches RAID-5 in tolerating all single disk failures. However, the RAID-x performs much better than RAID-5 in small writes. Image copying in RAID-x is much faster than the competing RAIDs. The experimental results in Sections 6-10 will verify all the claimed advantages. The major weakness of RAID-x lies in its 50 percent redundancy, at the same level as the RAID-10 and the chained-declustering RAID. However, the latter have higher capability to tolerate some multiple faults.

## 4 The RAID-x Design and Experiments

The USC Linux cluster is introduced below. Then, we describe the detailed architecture designs in the RAID-x. Finally, we outline the experimental settings. All benchmark experiments were implemented with the integration of hardware, software, and middleware on the USC cluster. The overall purpose was to tune the Linux cluster system for improving collective I/O performance.

### 4.1 Linux Cluster at USC

All ds-RAID architectures, including the new RAID-x, were implemented in an experimental Linux PC cluster built at USC. The prototype cluster was built with 16 Pentium PCs running the Linux operating system. A switched Fast Ethernet connects these PC nodes in full duplex operation. This implies a peak bandwidth of 12.5 MB/s per switch port. At present, each node is attached with an IDE disk. All 16 disks form a single I/O space. All PC nodes in the cluster are homogeneous and supported by some middleware packages.

At different experimenting stages, we reconfigure thatthe hardware 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 implementation effort was exerted on the RAID-x and chained-declustering RAID configurations, which are not commercially available.

The centralized NFS was used as a baseline for comparison purpose. The mapping of the mirrored blocks in RAID-x is done by a special address translation subroutines built in disk driver. All ds-RAID configurations were made possible with the SIOS features built in the Trojans cluster. To study the aggregate I/O bandwidth, we have to lift the caching restriction on local disks. This was done in the Linux kernel by issuing a special *sync* command.

### 4.2 Layout of the RAID-x

Fig. 3 shows the orthogonal RAID-x architecture with three disks attached to each node. All disk blocks in the same horizontal stripe, ($B_0$, $B_1$, $B_2$, $B_3$) are accessed in parallel. Consecutive stripe groups, such as ($B_0$, $B_1$, $B_2$, $B_3$) and ($B_4$, $B_5$, $B_6$, $B_7$), etc., are accessed in a pipelined fashion because multiple disks are attached to each SCSI bus.

In general, an *n*-by-*k* RAID-x has a *stripe group* of ***n*** disk blocks from $n$ 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*. Trade-offs do exist between these two orthogonal concepts of parallelism.

The pipelining depth of successive stripe groups depends on the I/O bus used. We plan to extend the Trojans cluster with four disks per node. Using 40 GB SCSI

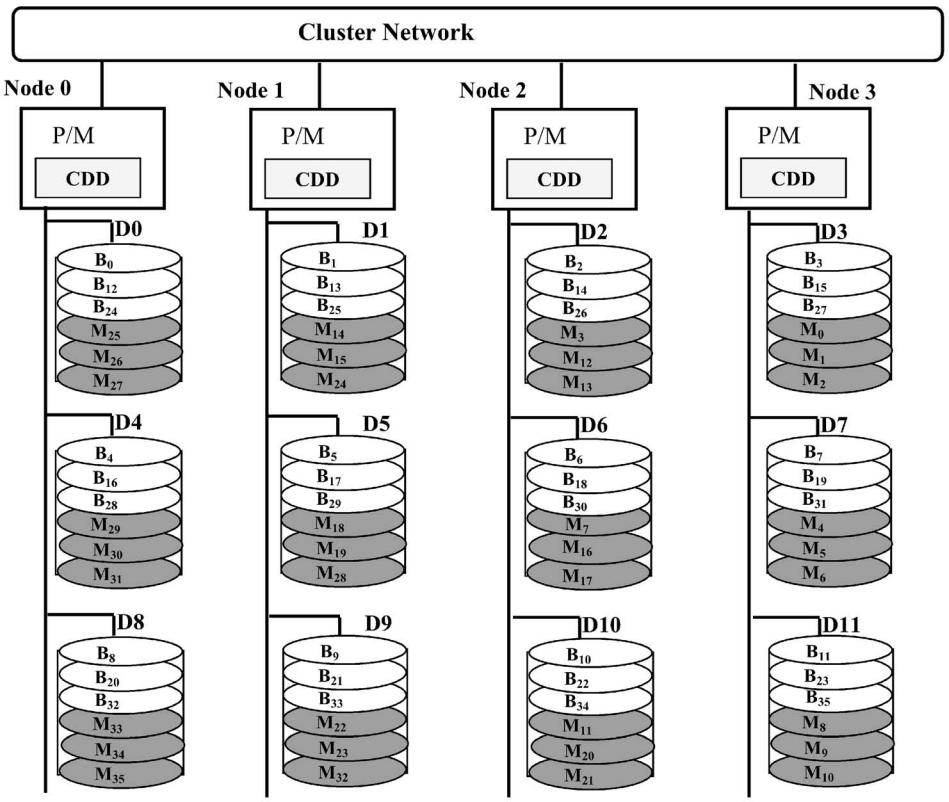


Fig. 3. A $4 \times 3$ RAID-x architecture with orthogonal striping and mirroring (P: processor, M: memory, Dj: the jth disk, $B_i$: the ith data block, $M_i$: the ith image block).

disks, the RAID-x array may have 2.56 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 eight disks per node, the disk array could be enlarged to have a total capacity exceeding 40 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.

### 4.3 Experimental Settings

In our experiments, each disk block (stripe unit) is set as 32 KB. 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 32 KB, retrieved from one block of a disk. We have performed five benchmark experiments. These experiments measure the parallel I/O performance in terms of the aggregate I/O bandwidth, elapsed time of Andrew benchmark, and the I/O rate or seek rate in Bonnie benchmark.

The first experiment is based on using the Andrew benchmark to reveal the relative performance of various ds-RAIDs against the use of a central NFS server. The second test reveals the effects of traffic rate or of the number of client requests in the disk array system. The third test reveals the bandwidth scalability of the four ds-RAID configurations. The fourth test checks the striping effects on cluster I/O performance. Finally the last test uses the Bonnie Benchmark to reveal the software overheads in various cluster I/O operations on four ds-RAID configurations.

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 Sections 6 through 10. 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 disks. No heavy cross-space system calls are needed to access remote files.

## 5 Cooperative Disk Drivers in Clusters

Generally speaking, local disk access is faster than remote disk access. This section describes how to reduce the latency of remote disk access. We have developed a *cooperative disk driver* (CDD) which can achieves this goal. The functional structure of the CDD design and its implementation requirements are in this section. In particular, we specify the unique mechanisms used to reduce the remote acess latency and to maintain data consistency in the ds-RAID.

### 5.1 Remote Disk Access

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. Fig. 4 illustrates the difference of using the NFS and CDDs for remote disk accesses.

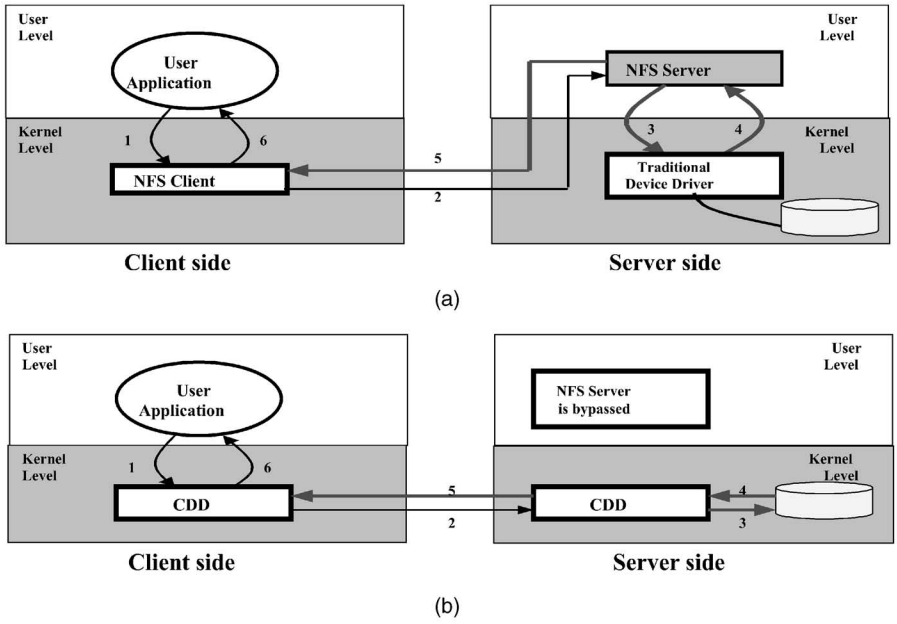


Fig. 4. Remote disk access steps using the NFS server versus the use of cooperative disk drivers in the RAID-x cluster. (a) Parallel disk I/O using the NFS in a server/client cluster. (b) Using CDDs to achieve a SIOS in a serverless cluster.

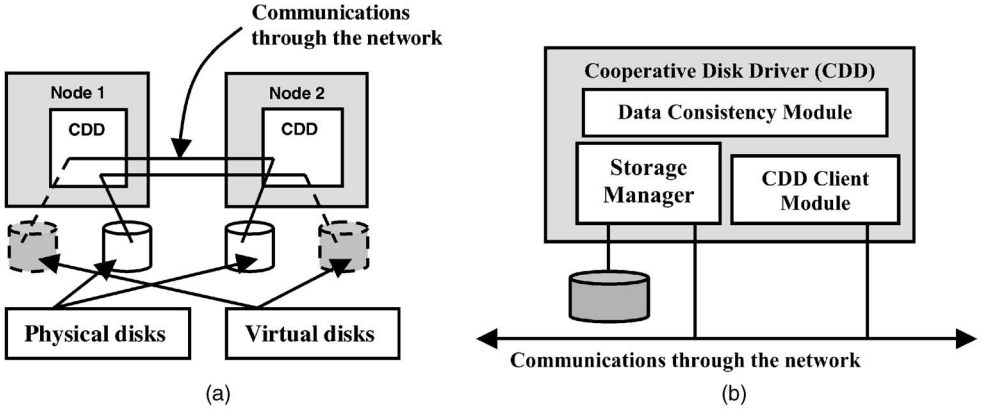


Fig. 5. Architecture design of the cooperative device drivers. (a) Device masquerading. (b) CDD architecture.

Fig. 4a shows six steps to access remote disks using a central NFS server. Unix or Linux system calls take place in Steps 3 and 5 crossing the user and kernel spaces. This may introduce long delays in transferring data files between 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.

Fig. 4b shows the modified six steps that are implemented with the CDDs. 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 used either as a client or as a server, or both. Therefore, removing disk access latency is significantly reduced. Thus, the scalability is improved to build larger ds-RAID for cluster I/O processing.

### 5.2 The CDD Architecture

The *device masquerading technique* [15] 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.

Fig. 5a 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. Fig. 5b shows the internal design of a CDD. Each CDD is essentially made from three working modules.

The *storage manager* receives and processes the I/O requests from remote *client modules*. The *client module* redirects local I/O requests to remote storage managers. The *data consistency module* is responsible for maintaining

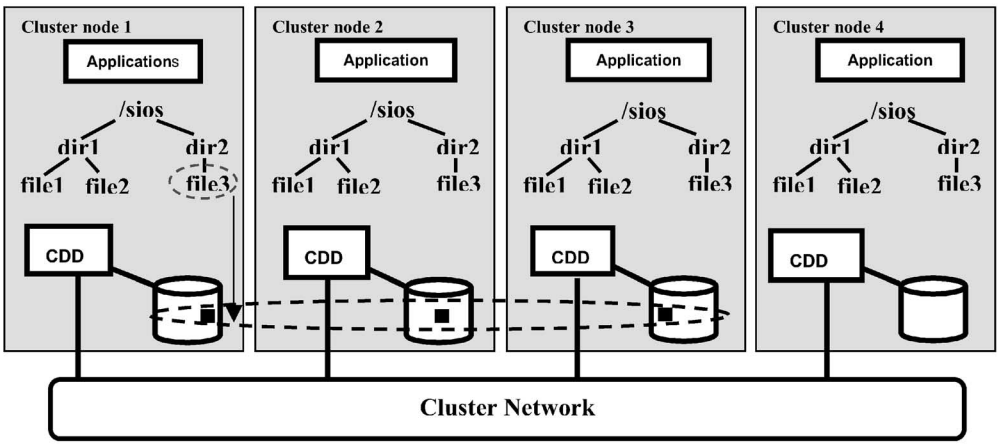


Fig. 6. Maintaining consistency of the global directory /sios by all CDDs in any of the ds-RAID configured on the Trojans cluster.

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. 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.

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 implement other RAID configurations. 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 special disk drivers. Our approach simplifies distributed file management. All CDDs in the same stripe maintain block-level data consistency. Fig. 6 shows a scenario how the 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 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.

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.

A *lock-group table* is used to facilitate distributed file management. 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. To maintain data consistency, the lock information is duplicated on every node. In case of a node crash, the lock information is still available from the remaining node. If the crashed node is the home node of the lock, the lock will be released from the lock table on the remaining nodes.

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) [29] and Andrew File System [16]. When the number of clients becomes very large, the SIOS provides a scalable performance through its serverless cluster design. The global file hierarchy is useful for process migration, where a process can access the opened files from any cluster node.

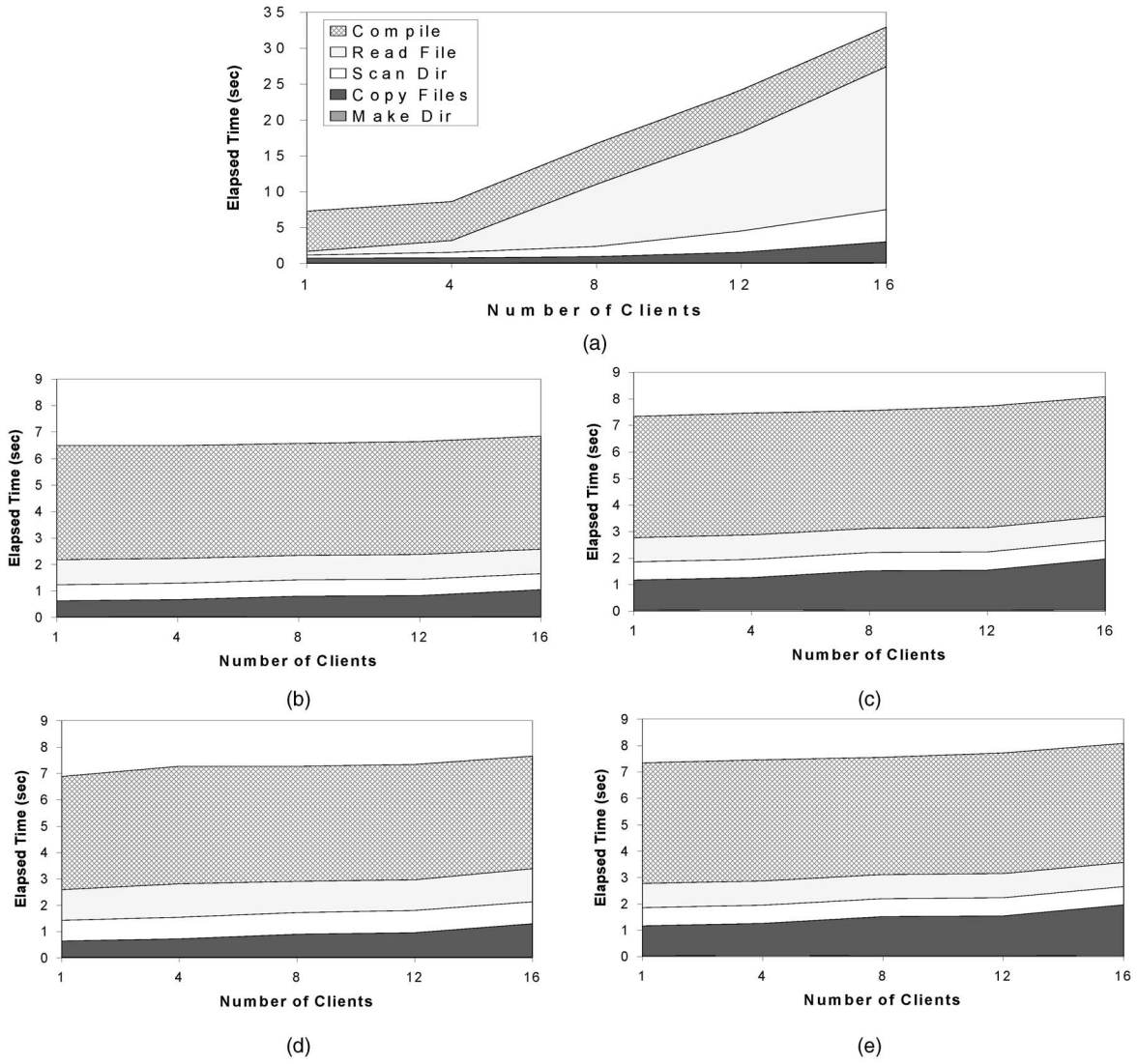


Fig. 7. Andrew benchmark results of four ds-RAID configurations compared with the NFS results on the Linux cluster at USC. (a) NFS results, (b) RAID-x results, (c) chained declustering results, (d) RAID-10 results, and (e) RAID-5 results.

## 6 Relative Performance of ds-RAID and NFS

In the next four sections, we report the performance results of four ds-RAID configurations running on the Trojans cluster. The performance results are obtained from raw I/O bandwidth measurements or from commonly accepted benchmark experiments. We discuss specific performance effects from the traffic rate, size scalability, and stripe unit size. In this section, we compare the relative performance of various ds-RAIDs with respect to a central NFS using the Andrew benchmark [17].

### 6.1 Andrew Benchmark Suite

The Andrew benchmark was developed at Carnegie Melon University. In the past, Andrew has been applied to test the performance of NFS, AFS, Digital Petal disk array [25], 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. Fig. 7 plots 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 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. Andrew benchmark is more effective for evaluating distributed file management systems. We apply the benchmark here because we want to reveal the relative performance of various ds-RAIDs with respect to the central NFS approach.

The performance is indicated by the *elapsed time* in executing the Andrew benchmark on the target RAID configuration. These tests demonstrate how the storage structure affects the performance of the file system. 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. 6) atomic.

### 6.2 Benchmark Results

Fig. 7a show the results of using the central NFS. Fig. 7b, Fig. 7c, Fig. 7d, and Fig. 7e show the elapsed times in executing the Andrew benchmark on four ds-RAID configurations. The NFS time increases sharply with the number of clients, while the ds-RAID configurations are all scalable with quite flat elapsed time. The bar height in Fig. 7b stays essentially constant. For 16 clients, the elapsed time of the Andrew benchmark on four ds-RAID architectures increases from 6.8 seconds to 7.9 seconds slowly.

The benchmark shows that the NFS requires 33 seconds to complete the execution. All four ds-RAIDs perform well here, approximately at the same level. Their differences in read performance are rather small. Among the four ds-RAIDs, majority of the time (approximately 56 percent to 62 percent) was spent on the Compile phase. The Read-File and Scan-Directory are both essentially read operations, each consuming about 12 percent to 17 percent and 9 percent to 11 percent of the time, respectively. Copy-File times increase slightly with the traffic due to parallel writes. The time to Make-Directory is very small and thus can be ignored.

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 a 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 is that all ds-RAIDs perform much better than using the centralized 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 relatively 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. This will be further studied in subsequent sections.

## 7 Effects of Traffic Rate on I/O Performance

In the following sections, we report the ds-RAID performance using some synthetic workloads, representing uniform distribution of parallel reads and writes. These experiments will reveal the raw aggregate I/O bandwidth. The cluster network is a dedicated switched Fast Ethernet with 16 ports. In theory, the aggregate I/O bandwidth is upper bounded by $12.5 \times 16 = 200$ MB/s. However, our experiments have never reached this peak network bandwidth.

In fact, each IDE disk can only deliver at most 4 MB/s due to network protocols and software overheads experienced. This implies a peak value of $4 \times 16 = 64$ MB/s in aggregate bandwidth, far below the above upper bound. With 16 disks, we achieved at most 28 percent of the peak bandwidth (18 MB/s) in all measurements. In this section, we reveal the effects on I/O bandwidth performance by increasing the total *I/O traffic rate*, which is the number of simultaneous client requests to the ds-RAID subsystem. In subsequent sections, we reveal other performance effects and provide scalability and overhead analyses of the benchmark results.

### 7.1 Access Granularity

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. A small access is set at block level of 32 KB. 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 reads or writes in file accesses.

We consider up to 16 client requests representing light to heavy I/O traffic rates. 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. Small read or write tests individual disk performance plus the redundancy overhead. Local or remote disk accesses are generated with a uniform distribution.

All files are uncached initially and each client accesses its private files from local or remote disks. All read or write operations are performed simultaneously, using the MPI_Barrier() command. For write operations, each client write the files to the buffer and issues the *sync*() call to stripe the data blocks to all 16 disks, repeatedly. The sync() call is issued to eliminate all buffering or caching effects.

### 7.2 NFS Performance

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. The fact that the NFS performance being lower than the Ethernet limit clearly demonstrates that the NFS is not scalable in cluster I/O processing.

### 7.3 Parallel Reads

For large read, in Fig. 8a, RAID-x performs only slightly lower than chained declustering, bur far above the RAID-10 performance. Both chained declustering and RAID-x show higher scalability that that of RAID-10. For 16 clients, RAID-x and chained declustering can scale to a bandwidth approximate to 16 MB/s. RAID-10 lags behind with a show of 10.7 MB/s bandwidth. The results for small read are shown in Fig. 8b, which are very close to that for large read, except that RAID-x occasionally has higher bandwidth than that of chained declustering.

### 7.4 Parallel Writes

Fig. 8c shows the large write situations. In this test, each client writes a 20MB file to the disk buffer to stripe the data blocks to all 16 disks, recursively. For timing purposes, all write operations among the clients are synchronized by

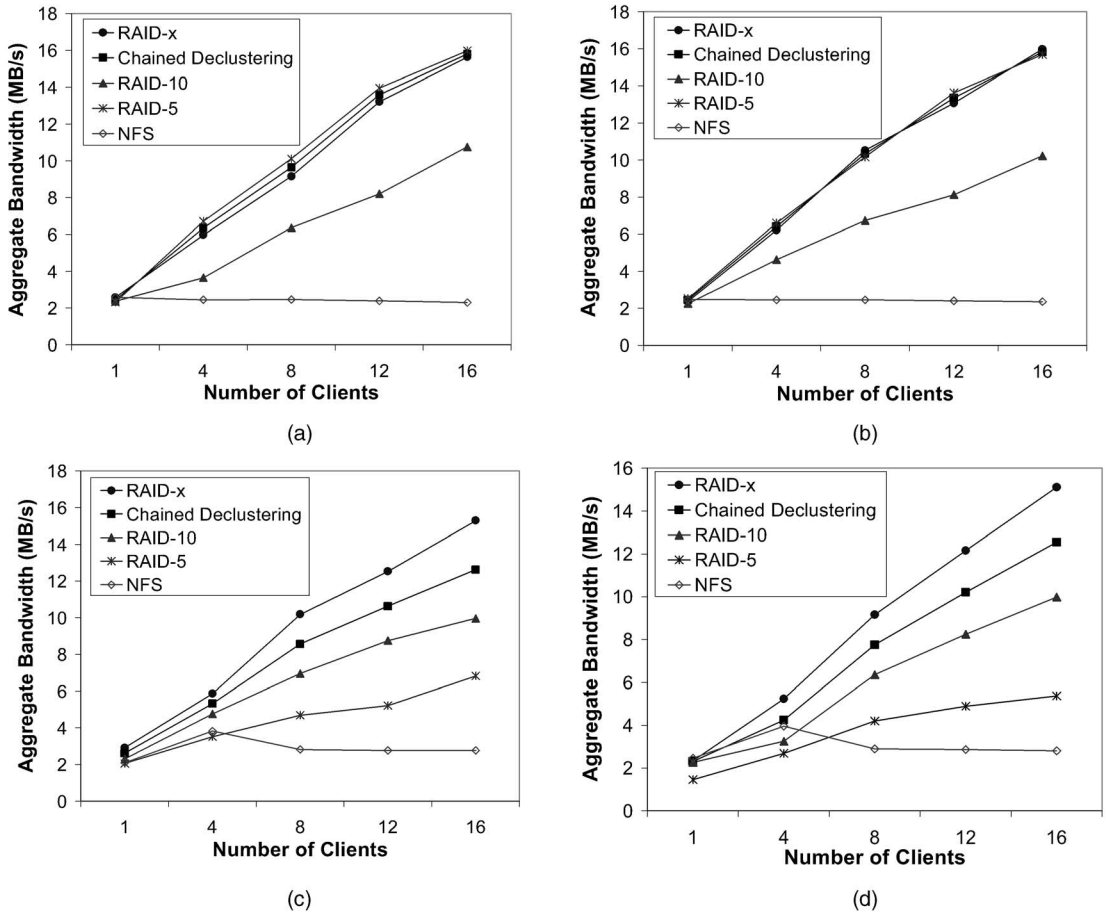


Fig. 8. Effects of traffic rate on the aggregate I/O bandwidth performance of four ds-RAID architectures on the Trojans cluster. (a) Large read (20 MB per client). (b) Small read (32 KB per client). (c) Large write (20 MB per client). (d) Small write (32 KB per client).

issuing a special *sync*() call. The NFS scales in performance up to four requests, even higher than that of chained declustering, due the caching effect at the NFS server. As the requests exceed 4, the NFS bandwidth drops to a low 2.8 MB/s. For parallel writes of either a large file (Fig. 8c) or a small block (Fig. 8d), the RAID-x achieves the best scalability among the four with a highest 15.3 MB/s for 16 clients.

In contrast, chained declustering scales slowly due to the heavy involved seek latency for the frequent change of disk head position in writing data and mirroring. RAID-10 scales slower than RAID-x, but faster than chained declustering. This is due to the fact that only half of the disks are actually accesses. To sum up, the RAID-x outperforms the others because full parallelism is exploited to access all data blocks in the upper half of the disk array (Fig. 2a). The parity update overhead is totally eliminated in either RAID-10 or RAID-x.

### 7.5 Summary on Traffic Effects

The aggregated I/O bandwidth is primarily limited by the effective network bandwidth rather by the traffic density. For parallel reads (Fig. 8a, Fig. 8b), the RAID-5, chained declustering, and RAID-x have almost equal performance. The RAID-10 scales slower than other mirroring RAIDs does. For parallel writes, the RAID-x clearly outperforms the others. The chained-declustering RAID ranks the second and the RAID-10 the third. The low write performance of RAID-5 makes it least scalable. This situation is especially true for small write operations. The parity update overhead in RAID-5 is totally eliminated in RAID-x based on orthogonal striping and mirroring.

## 8 Striping Effects on Cluster I/O Bandwidth

This section is dedicated to the effects of data striping across the distributed disks in a ds-RAID subsystem. Again, the issues are answered by analyzing the experimental results reported in Fig. 9.

### 8.1 Striping Issues

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). Disk array performance is sensitive to stripe unit size because different stripe unit size may result in different granularity of data to be accessed. In the past, Kim, et al. [24] has studied the effects of striping and caching effects.

In our study, assuming a sufficiently large data file eliminates the caching effect. The reason is that we are interested in the asymptotic behavior of the aggregate I/O bandwidth, limited only by the network bandwidth and by software overheads experienced. With a sufficiently large data file, we were able to isolate the effects of striping by changing the file and block sizes. Other factors affecting the

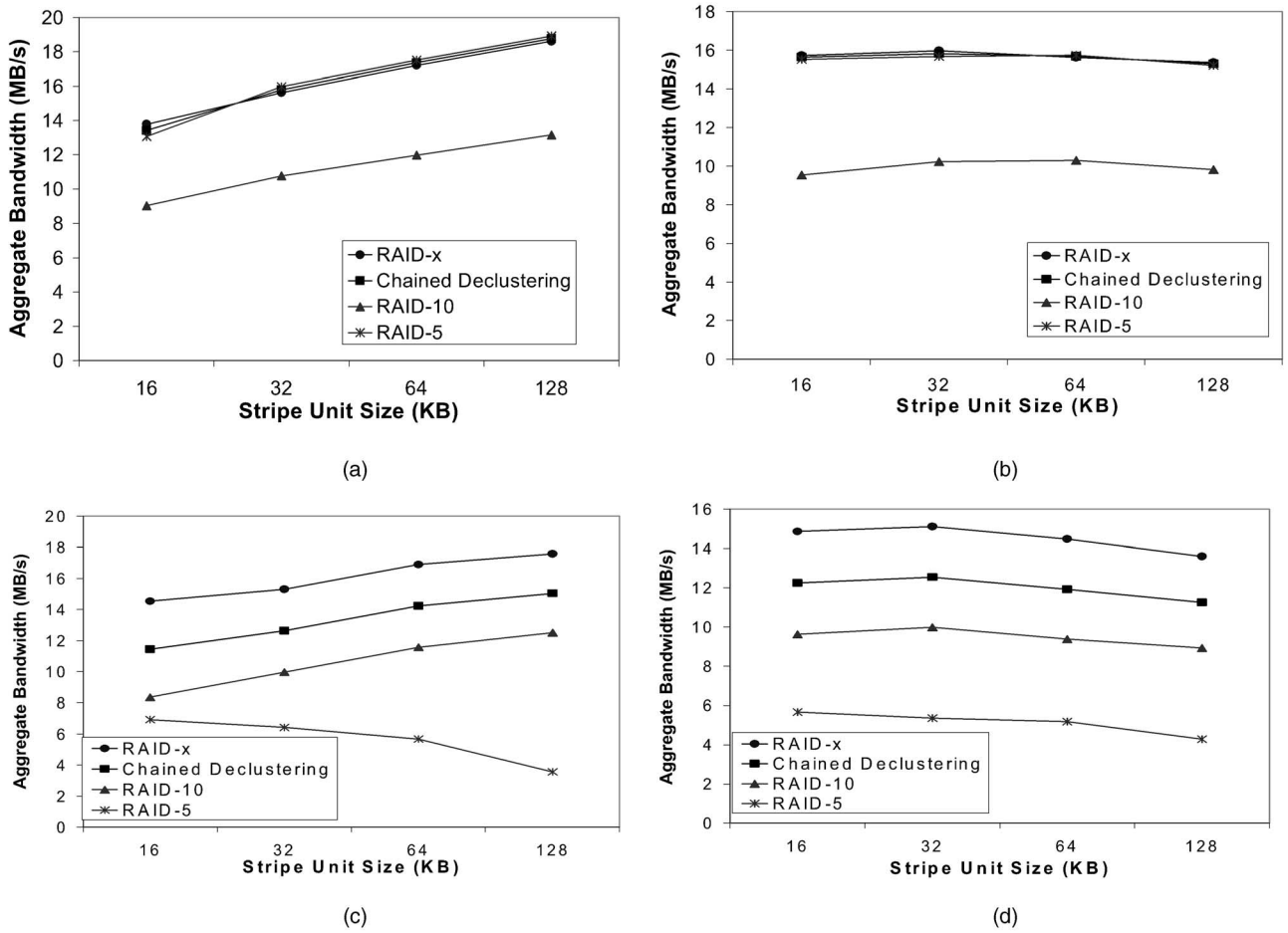


Fig. 9. Effects of stripe unit size on the aggregate I/O bandwidth of four ds-RAID architectures on the Trojans cluster. (a) Large read (320 MB for 16 clients). (b) Small read (512 KB for 16 clients). (c) Large write (320 MB for 16 clients). (d) Small write (512 KB for 16 clients).

performance include the disk characteristics, storage servers, and distributed file system used. We lump together all of these effects in the following experimental data collected.

### 8.2 Parallel Reads

Fig. 9 reveals the effects of stripe unit size on the aggregate I/O bandwidth performance of four ds-RAID architectures. We consider the stripe unit increasing from 16 KB to 128 KB under the condition that 16 client requests are posted on 16 disks simultaneously. Again, the RAID-5, chained-declustering and RAID-x have comparable read performance. The RAID-10 lags far behind the other three RAIDs in either large or small reads.

For small stripe unit sizes, the bandwidth gap is as wide as 6 MB/s between RAID-10 and other three RAIDs. For large read (Fig. 9a), 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. 9b), the optimal block size stays with 32 KB. The bandwidth gaps among the four RAIDs widens and none of them is scalable as seen by the flat curves in Fig. 9b.

### 8.3 Parallel Writes

For large writes (Fig. 9c), the RAID-x demonstrates higher speed than what can be achieved in a chained-declustering RAID or a ds-RAID-10. The RAID-5 shows a declining bandwidth with increasing stripe size. The top three RAIDs all perform well with larger data blocks. With small writes in Fig. 9d, the optimal block size is 32 KB.

However, all four ds-RAID show a trend of slowing down, as the block size increases. The optimal block size varies with applications, disk types, workload conditions, and even cluster platforms used. These results certainly reflect the site factors. The USC Linux PC cluster uses IDE disks and fast Ethernet connections. The results may vary to some extent, if SCSI disks or Gigabit Ethernet were used in the experimentation.

### 8.4 Optimal Stripe Unit

In an earlier paper [23], we have provided an analytical model to determine the optimal size of a stripe unit as $2dKC$ bytes, where $d$ is the number of disks per drive, $K$ is the number of sectors per track, and $C$ is the sector size in bytes. The above result implies that one should choose a stripe unit approximately equal to the size of one cylinder, if the file is manageable in size. Large stripe unit tends to cluster the file in a few disks, hence reducing the parallelism exploitable in an I/O request.

The optimal choice of stripe unit size lies in a trade-off between the two conflicting goals. In the past [24], Kim et al. has studied the effects of striping and caching. Caching effect is ignored here using a sufficiently large data file. We

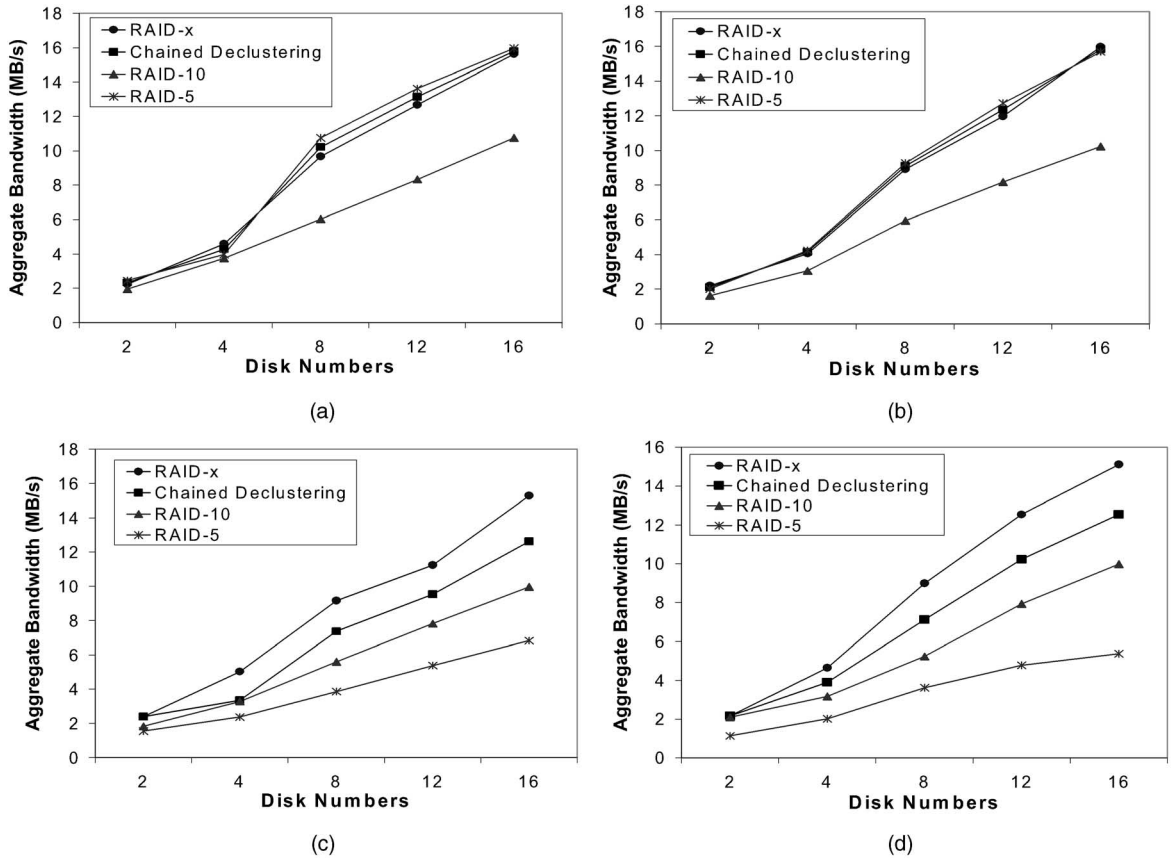


Fig. 10. Scalability of four ds-RAID architectures with increasing number of disks under a heavy I/O traffic conditions. (a) Large read (320 MB for 16 clients). (b) Small read (512 KB for 16 clients). (c) Large write (320 MB for 16 clients). (d) Small write 512 KB for 16 clients).

thus focus on the asymptotic behavior of I/O bandwidth, primarily limited by the protocol and software overhead experienced. With a sufficient large data file, we were able to isolate the effects of striping by changing the file size and stripe unit simultaneously.

## 9 Scalability Analysis of Distributed Raid

This section discusses the scalability of various distributed disk arrays in a cluster environment. We use the times for parallel reads and parallel writes to assess the scalability of ds-RAID architectures. Finally, we summarize the results with improvement factors achieved in the scaling experiments.

### 9.1 The Scaling Issues

Scaling of a disk array size may increase the aggregate I/O bandwidth. More disks in the array may satisfy more client requests. But it may also introduce more network contentions. The effects of disk array size are very similar to the effects of client request rate. In both cases, we consider the cases of accessing a large number of user files simultaneously. We plot in Fig. 10 the measured aggregate I/O bandwidth against the disk-array size, increasing from two to 16 disks.

The total number of client requests is fixed at 16. Since the NFS does not scale well with increasing number of disks, we do not include NFS in this experiment. The workload is fixed at 320 MB for large read or for large write, regardless of the array size. For small read or write, the fixed workload is set at 512 KB for 16 clients. In what follows, we realize that all four ds-RAIDs show some scalable performance in read and write operations. However, the scalability rating varies among the four ds-RAID architectures.

### 9.2 Parallel Reads

For parallel read, the RAID-5, chained-declustering RAID, and RAID-x perform almost the same. They all scale well with the increasing number of disks. Comparing Fig. 10a and Fig. 10b, we realize that there exists very little difference in bandwidth performance between large and small reads. The RAID-10 is much less scalable for either large or small reads. This is mostly attributed to the fact that RAID-10 can not be aggressively implemented to assume contiguous addresses as discussed in Section 3. For large reads, more stripe groups must be retrieved recursively. The RAID-5 has slightly higher performance, as shown in Fig. 10a.

### 9.3 Parallel Writes

For large or small writes, the ranking changes sharply as shown in Fig. 10c and Fig. 10d, respectively. The RAID-x far outperforms the remaining disk arrays. The RAID-10 moves ahead of the chained-declustering RAID. For 16 disks, the write bandwidths of RAID-x, RAID-10 and chained-declustering RAID are 15.3 MB/s, 9.9 MB/s, and 6.8 MB/s, respectively. These bandwidth differences are attributed

TABLE 4
Achievable I/O Bandwidth and Improvement Factor of Three ds-RAIDs Compared with 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 |
| **I/O 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 |

mainly to their architectural characteristics. The strength of the RAID-x lies mainly in its superior performance in performing parallel write operations.

### 9.4 Improvement Factors

Table 4 compares the bandwidth improvement of using the three RAID subsystems and the NFS. The *improvement factor* is defined as the ratio of read or writes bandwidth of 16 clients over that of 1 client request on the cluster [1]. For parallel reads, chained declustering and RAID-x have almost equal performance and they scale at the same pace. Therefore, we need only report the large read results. The difference is attributed mainly to the CDD protocol and TCP/IP overhead incurred. The gaps among them widen rapidly, showing the superiority of RAID-x in parallel write operations.

Comparing with Berkeley xFS results [1], our 1-client bandwidth is quite high due to well-exploited parallelism in 16-way striping across the disk array. For this reason, the improvement factor is lower than that achieved by the xFS system. However, the RAID-x demonstrates the highest improvement factor among the 3 RAID architectures. The comparison separates the large read from write situations. We ignored small reads because they are not relevant to the saturated aggregate performance.

## 10 Overhead Analysis in Cluster I/O Operations

Cluster I/O operations are slowed down essentially by limited network bandwidth, extensive software and communication overheads experienced on distributed disk accesses. This section focuses on all overhead issues. We use the Bonnie benchmark to reveal the overheads in six different cluster I/O operations. Cluster I/O is often slowed down by these overheads experienced.

The overhead imposes a limit on cluster I/O performance. The overheads are attributed to a number of factors, including the *network protocols*, *file size, access granularity, access pattern, disk block size, internode communications, OS system calls,* and even the delays due to *random seek time,* etc. The Bonnie benchmark offers various tests to answer some of these concerns.

### 10.1 Bonnie Benchmark Suite

We report below the Bonnie benchmark results which reveal the software overhead associated with various cluster I/O operations. 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.

The benchmark tests the *input rate*, *output rate*, and *seek rate* of parallel I/O operations. In our test, we used 500 MB data as the workload. The file size is large enough to avoid the buffer caching effects so that the results reported can truly reflect the performance of the I/O storage. The OS overheads include the times to use the file system to access the low-level storage. In Fig. 11, the input rate or output rate is expressed as MB/s, while the seek rate is measured as Seeks/s.

In Fig. 11, the I/O rates are plotted against the number of disks in the disk array. The faster is the data transfer rate, the less is the software overhead experienced. The client's work is to write or read the 500MB-long file. The server could be on a remote CDD. Thus, very little CPU time is spent locally in handling distributed I/O. This is very different from the situation in accessing only the local disks.

### 10.2 Character Write

Fig. 11a shows the results of this. 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 RAID-10. The write rate of the RAID-5 lags far below because more software overhead experienced with the parity update operation, while the other three mirroring RAID experience no such parity overhead.

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 gap among them increases with the disk array size. The RAID-5 performs the worst with a low peak bandwidth of 1.6 MB/s. In all ds-RAIDs, the output rate increases only slightly with the array size. These results agree with the small-write results reported in Fig. 10.

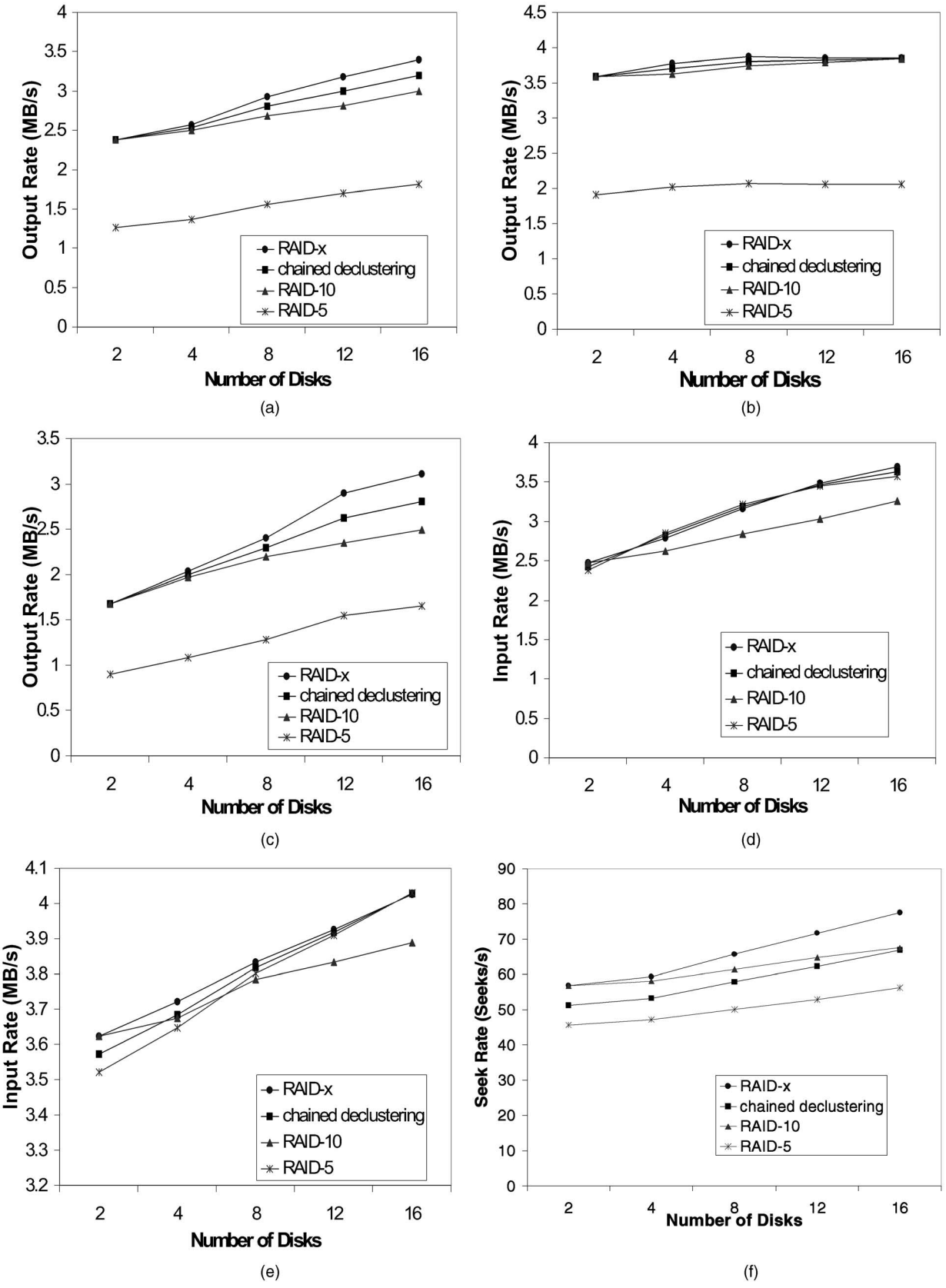


Fig. 11. Bonnie benchmark results of six I/O functions performed on four ds-RAID configurations of the USC Trojans cluster. (a) Character write, (b) block write, (c) file rewrite, (d) character read, (e) block read, and (f) random seek.

### 10.3 Block Writes

Fig. 11b shows the results of writing a 500-MB file to the ds-RAID in blocks. Again the top three ds-RAID configurations have the same output rate of 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 output rates are higher in absolute value than in the Character-Write test. This is because character-based operations introduce a considerable amount of software overhead in system calls, and in reconstructing the block-based I/O requests from the scattered characters. While lower overhead is experienced when the whole data file is lumped together for block write. Here, the size of disk array makes very little difference in the raw output rate.

### 10.4 File Rewrite

Fig. 11c shows the output rate of the File-Rewrite test, where a 500 MB file is read into the memory with each block modified and then written back to the disks. These files are much larger than those in character-write or block-write tests are. They correspond to the case of large reads and writes subsequently. Therefore, the output rate increases faster than those fine-grain disk accesses reported in Fig. 11a. The gaps among the four ds-RAID also becomes wider with more disks.

### 10.5 Character and Block Reads

Fig. 11d and Fig. 11e show the Character Read and Block Read results, respectively. In both tests, the RAID-5, RAID-x, and chained-declustering RAID perform better than RAID-10. As the array increases to 16 disks, all three RAIDs converge to an output rate of around 4 MB/s. But the RAID-10 speed lags far behind because it experienced more overheads. This overhead comes from a less efficient mirroring scheme on the RAID-10.

### 10.6 Random Seek

The benchmark program executes 4,000 seeks to random locations in the file. About 10 percent of these seeks, the blocks are rewritten to the disks. The seek rate in Fig. 11f increases sharply to as high as 77 seeks/s. The speed gaps among the four ds-RAIDs become closer, as the disk number increases. For 16 disks, the seek rates are 77 Seeks/s in RAID-x, 67 Seeks/s in RAID-10 and chained-declustering RAID, and 56 Seeks/s for RAID-5. Higher seek rate corresponds to a larger number of seeks performed simultaneously over a large number of stripe groups.

### 10.7 Some Observations

1. The total I/O processing time is attributed to I/O system calls, I/O transfer time, and delays in applying TCP/IP protocol in the cluster network. The Bonnie benchmark reveals the performance of the network storage as a whole. Although the actual disk access time is not a small portion of the total time, the overhead is even a greater portion. In fact, the benchmark results reveal more weight on protocol and software overhead than the time for raw I/O data transfer.
2. The Bonnie benchmark reveals the effects of access granularity on the RAID performance. The larger is the grain size in parallel writes, the more scalable will be the RAID. In Fig. 11, the access size increases from characters (Fig. 11a) to blocks (Fig. 11b) and to files (Fig. 11c and Fig. 11f). When the software overhead grows faster than the available bandwidth, the ds-RAID becomes less scalable.
3. The less overhead experienced in RAID-x is attributed to the time saved from background image writing as described in Section 3. Another reason why RAID-x has less overhead is due to its efficient mirroring scheme. The RAID 10 and chained-declustering RAID demand much more time to write the data images for lack of the orthogonality property.

## 11 Conclusions and Suggestions

Four distributed RAID architectures including the new orthogonal RAID-x have been evaluated for scalable cluster I/O processing. The RAID-x architecture demonstrates its strength in cluster I/O performance. Both experimental and analytical results are presented to back up the claims. To conclude, we summarize the contributions and shortcomings and make suggestions for further research effort.

### 11.1 Summary of Contributions

The main contributions of this work lie in the development of the orthogonal RAID-x architecture. We have developed cooperative disk drivers to enable the SIOS with lowered latency in remote disk access and presented extensive benchmark results on four ds-RAIDs, which can be applied to design optimization of future cluster I/O subsystems.

1. The new RAID-x shows its strength in building distributed storage for serverless clusters. The orthogonal architecture is unique with *orthogonal striping and mirroring*. In the reliability area, the RAID-x can tolerate all single disk failures like the RAID-5. The background image writing also contributes the performance gain, especially in large write operations. The RAID-x completely eliminates the small-write problem by using no parity checks.
2. We have developed new disk drivers at the Linux kernel level to support the SIOS in any ds-RAID configuration. These drivers enable faster remote disk access and parallel I/O without using a central file server. Much of the software overhead comes from heavy system calls in cluster I/O operations. The use of the CDDs with middleware reduces some overheads significantly. Benchmark performance results show scalable performance of RAID-x in cluster I/O operations.
3. The Andrew benchmark shows the scalable advantage of using any ds-RAID over the use of a central NFS for cluster I/O operations. For parallel reads with 16 active clients, the RAID-x achieved a 1.5 times higher performance than other three RAID architectures. For parallel writes, RAID-x shows up to 2.2 times higher performance. The

Bonnie benchmark shows that the RAID-x cuts 13 percent of the overhead in many I/O-centric operations.

4. We have extended the Linux kernel to support the SIOS. This helps also implement the shared virtual memory and global file management. Many I/O-centric applications can benefit by achieving scalable performance on a serverless cluster of computers. In particular, this will benefit data mining, transaction processing, and pervasive computing applications.

### 11.2 Relative Merits of Four ds-RAIDs

Even the RAID-x has clear advantages in write performance, we have to admit some of its weakness. Both RAID-10 and chained-declustering RAID have higher reliability and fault tolerance capability than our RAID-x. For I/O applications demanding higher bandwidth performance, RAID-x should be the choice. However, for highly fault-tolerant I/O applications, the RAID-10 and chained declustering approach are still better. In this sense, the RAID-5 is weak in both performance and fault tolerance, especial in small-write applications.

We applied the standard storage servers with local disks. These servers have not been optimized in the distributed cluster environment. The overhead in maintaining data consistency is still high. The use of simple locking mechanism may add extra time delay. A new data consistency model is thus very much needed to explore even higher performance in ds-RAID. These weaknesses triggered us to suggest the following further studies:

### 11.3 Suggestions for Further Research

To amend the above shortcomings, we suggest to attack the following issues towards the optimization, industrialization, and applications of ds-RAIDs.

1. To scale up to a much larger number of disks, the communication protocol used in the CDD may become a performance bottleneck. To solve the problem, the TCP/IP protocol could be replaced by a low-latency protocol, similar to that suggested by Martin, et al [27].
2. A new distributed file system is desired to improve the efficiency of the disk drivers in using even faster networks. A more efficient data consistency protocol will provide higher scalability to hundreds or even thousands of disks in a very large ds-RAID system.
3. An *adaptive sector grouping* method was suggested in [23] 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 very large ds-RAID configurations.
4. In this work, we did not apply any data prefetching [2], [33] or cooperative caching technique [10], [18], [24]. These techniques will hide some latency for parallel reads in using the ds-RAID. Even higher performance would be expected, if these acceleration techniques were applied.

## ACKNOWLEDGMENTS

The authors appreciate the critical comments made by the anonymous referees and the valuable suggestions by Associate Editor Alok Chaudary. This research was supported in part by the University of Southern California and in part by the research grant HKU 7022/97E from the Hong Kong Research Grant Council. The authors would also like to acknowledge the research facilities provided by the Internet and Cluster Computing Laboratory at University of Southern California. Roy Ho would like to thank the financial support from the Department of Computer Science and Information Systems, University of Hong Kong. Hai Jin would like to acknowledge the support from the Area-of-Excellence Program in Information Technology at the University of Hong Kong.

## REFERENCES

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

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

[3] S. Asami, N. Talagala, and D.A. Patterson, "Designing a Self-Maintaining Storage System," *Proc. 16th IEEE Symp. Mass Storage Systems,* pp. 222-233, Mar. 1999.

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

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

[6] P.H. Carnset et al., "PVFS: A Parallel File System for Linux Clusters," *Proc. Extreme Linux Track: Fourth Ann. Linux Showcase and Conf.,* Oct. 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, pp. 145-185, June 1994.

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

[9] T. Cortes, "Software RAID and Parallel Filesystems," *High Performance Cluster Computing,* R. Buyya, ed., Prentice Hall PTR, pp. 463-496, 1999.

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

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

[12] G. Gibson et al. "A Cost-effective, High-bandwidth Storage Architecture," *Proc. Eighth Conf. Architectural Support for Programming Languages and Operating Systems,* 1998.

[13] M. Harry, J.M. Del Rosario, and A. Choudhary, "VIP-FS: A VIrtual, Parallel File System for High Performance Parallel and Distributed Computing," *Proc. Ninth Int'l Parallel Processing Symp. (IPPS '95),* pp. 159-164, Apr. 1995.

[14] J.H. Hartman, I. Murdock, and T. Spalink, "The Swarm Scalable Storage System," *Proc. 19th IEEE Int'l Conf. Distributed Computer Systems (ICDCS '99),* June 1999.

[15] R. Ho, K. Hwang, and H. Jin, "Design and Analysis of Clusters with Single I/O Space," *Proc. 20th Int'l Conf. Distributed Computing Systems (ICDCS 2000),* pp. 120-127, Apr. 2000.

[16] J.H. Howard et al., "Scale and Performance in a Distributed File System," *ACM Trans. Computer System,* vol. 6, no. 1, pp. 51-81, Feb. 1988.

[17] H.I. Hsiao and D. DeWitt, "Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines," *Proc. Sixth Int'l Data Eng. Conf.,* pp. 456-465, 1990.

[18] Y. Hu, Q. Yang, and T. Nightingale, "RAPID-Cache—A Reliable and Inexpensive Write Cache for Disk I/O Systems," *Proc. Fifth Int'l Symp. High Performance Computer Architecture (HPCA-5),* pp. 204-213, Jan. 1999.
[19] J. Huber, C.L. Elford, D.A. Reed, A. Chien, and D. Blumenthal, "PPFS: A High Performance Portable Parallel File System," *Proc. Ninth ACM Int'l Conf. Supercomputing,* pp. 385-394, July 1995.
[20] 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,* Mar. 1999.
[21] K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, Programming.* New York: McGraw-Hill, 1998.
[22] K. Hwang, H. Jin, and R. Ho, "A New Distributed RAID-x Architecture for I/O-Centric Cluster Computing," *Proc. Ninth IEEE High-Performance Distributed Computing (HPDC-9) Symp.,* pp. 279-286, 2000.
[23] H. Jin and K. Hwang, "Adaptive Sector Grouping to Reduce False Sharing of Distributed RAID," *Cluster Computing J.,* vol. 4, no. 2, pp. 133-143, Apr. 2001.
[24] 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," *Proc. 19th IEEE Int'l Conf. Distributed Computing Systems,* pp. 544-551, 1999.
[25] E.K. Lee and C.A. Thekkath, "Petal: Distributed Virtual Disks," *Proc. Seventh Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* pp. 84-92, Oct. 1996.
[26] C.S. Li, M.-S. Chen, P.S. Yu, and H.I. Hsiao, "Combining Replication and Parity Approaches for Fault-Tolerant Disk Arrays," *Proc. IEEE Symp. Parallel and Distributed Processing,* pp. 360-367, Oct. 1994.
[27] R.P. Martin, A.M. Vahdat, D.E. Culler, T.E. Anderson, "Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture," *Proc. 24th Ann. Int'l Symp. Computer Architecture,* pp. 85-97, June 1997.
[28] G.F. Pfister, "The Varieties of Single System Image," *Proc. IEEE Workshop Advances in Parallel and Distributed System,* pp. 59-63, 1993.
[29] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network Filesystem," *Proc. USENIX Conf.,* pp. 119-130, June 1985.
[30] M. Stonebraker and G.A. Schloss, "Distributed RAID—A New Multiple Copy Algorithm," *Proc. Sixth Int'l Conf. Data Eng.,* pp. 430-437, Feb. 1990.
[31] N. Talagala, S. Asami, D. Patterson, and K. Lutz, "Tertiary Disk: Large Scale Distributed Storage," Technical Report UCB//CSD-98-989, Univ. of California, Berkeley, 1998.
[32] C.A. Thekkath, T. Mann, and E.K. Lee, "Frangipani: A Scalable Distributed File System," *Proc. ACM Symp. Operating Systems Principles,* pp. 224-237, Oct. 1997.
[33] P.J. Varman and R.M. Verma, "Tight Bounds for Prefetching and Buffer Management Algorithms for Parallel I/O Systems," *IEEE Trans. Parallel and Distributed Systems,* pp. 1262-1275, Dec. 1999.
[34] R.W. Watson and R.A. Coyne, "Parallel I/O Architecture of the High Performance Storage System," *Proc. 14th IEEE Symp. Mass Storage Systems,* pp. 27-44, Sept. 1995.
[35] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID Hierarchical Storage System," *ACM Trans. Computer Systems,* vol. 14, no. 1, pp. 108-136, Feb. 1996.

**Kai Hwang** received the PhD degree from the University of California at Berkeley. He is a professor of electrical engineering and computer science at the University of Southern California. An IEEE fellow, he specializes in computer architecture, digital arithmetic, parallel processing, and distributed computing. He is the founding editor-in-chief of the *Journal of Parallel and Distributed Computing.* He has published six books and more than 165 scientific papers. His latest book deals with *Scalable Parallel Computing*, published by McGraw-Hill, 1998. Dr. Hwang has lectured worldwide on computers, Internet, and information technology and has performed consulting and advisory work for IBM, MIT Lincoln Lab., ETL in Japan, ITRI in Taiwan, and GMD in Germany. He has chaired numerous IEEE/ACM conferences including the International Symposium on High Performance Computer Architecture '97, International Parallel Processing Symposium '96, Internatioanl Conference on Algorithms and Parallel Processing '96, Iinternational Conference on Parallel Processing '86, and the IEEE Symposium on Computer Arithmetic (ARITH-7). Presently, he leads a research group at University of Southern California developing new technology and architecture for Intranet security, reliable cluster computing, distributed firewalls, and pervasive computing.

**Hai Jin** received the PhD degree in computer engineering from Huazhong University of Science and Technology (HUST) in 1994. He is a professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. In 1996, he has visited the Technical University of Chemnitz, Germany. He has participated in the cluster computing research projects at the University of Hong Kong and at the University of Southern California since 1998. Dr. Jin is a member of IEEE and ACM. He chairs the 2001 International Workshop on Internet Computing and E-Commerce. He served as Program vice-chair of the 2001 International Symposium on Cluster Computing and the Grid. He has coauthored four books and published more than 50 scientific papers. His research interests cover parallel I/O, RAID architecture, fault tolerance, and cluster computing.

**Roy S.C. Ho** received the MPhil degree from the Department of Computer Science and Information Systems at the University of Hong Kong in 2000, where he received the BEng degree in Computer Engineering in 1998. His work on this project started with the University of Hong Kong and was completed during his visit to the University of Southern California in fall semester of 1999. Presently, he works as a software engineer in a startup cluster computing company in Hong Kong. His technical interest lies mainly in scalable computer systems and software support for distributed computing.