Reliable Cluster Computing with a New Checkpointing RAID-x Architecture

Kai Hwang, Hai Jin, Roy Ho, and Wonwoo Ro

Internet and Cluster Computing Laboratory
University of Southern California

Email : {kaihwang, hjin, wro}@usc.edu, and scho@csis.hku.hk

Abstract

In a serverless cluster of PCs or workstations, the cluster must allow remote file accesses or parallel I/O directly performed over disks distributed to all client nodes. We introduce a new distributed disk array, called the RAID-x, for use in serverless clusters. The RAID-x architecture is based on an *orthogonal striping and mirroring* (OSM) scheme, which exploits full-bandwidth and protects the system from all single disk failures.

The performance of the RAID-x is experimentally proven superior to RAID-1 and NFS in the Linux cluster environment. We propose a new *striped checkpointing* scheme, leveraging on striped parallelism and pipelined writing of successive disk stripes. This RAID-x architecture greatly enhances the throughput, reliability, and availability of scalable clusters. It appeals especially to I/O-centric cluster applications.

Keywords: Scalable computing, RAID architectures, parallel I/O, Linux clusters, disk mirroring, single system image, checkpointing, staggered writing, and fault tolerance

1. Introduction

Many *redundant arrays of inexpensive disks* (RAID) [6] use independent disks under the control of a single or multiple controllers. The TickerTAIP [3] pioneered the *Parallel RAID* architecture for supporting parallel disk I/O with multiple controllers. Still, these parallel disk arrays are implemented as a centralized I/O subsystem. These RAID subsystems are often attached to a storage server or used as network-attached disks [10].

For this reason, we consider the classic disk arrays as a *centralized* RAID. In contrast, this paper deals only with *distributed* RAID architectures. This concept was investigated by Stonebraker and Schloss [25]. The actual prototyping of distributed RAIDs did not start until the Petal [17] and the Tertiary Disk project [26].

A distributed RAID is constructed out of dispersed disks, which are physically attached to different computer hosts through the network connections. The Petal was built with a chained declustering [12]. The Tertiary Disk was built with a RAID-5 architecture using software support by the serverless xFS file system [2].

The architecture and performance of a new distributed RAID architecture, namely the RAID-x, are reported here. The level x is yet to be rectified with an appropriate code assignment by the RAID Advisory Board [22]. Our RAID-x differs from existing distributed RAID architectures in many aspects.

First, the RAID-x is built with a new disk mirroring technique, called *orthogonal striping and mirroring* (OSM). The small write problem associated with RAID-5 is completely eliminated in this OSM approach. Second, we use cooperative disks instead of independent disks.

To enable true cooperation among dispersed disks, we have developed *cooperative disk drivers* (CDD) at the kernel level. Data consistency is maintained inside the CDD, instead of using a central network file system. Therefore, unmodified file system interface is available to users. Third, the RAID-x was specially designed over distributed disks for I/O-centric cluster computing.

The rest of the paper is organized as follows: Section 2 describes the Trojans cluster architecture and also presents an overview of distributed RAID architectures. Our RAID-x approach is compared with the architectural designs in Berkeley Tertiary Disks running the xFS, Digital Petal system and Princeton TickerTAIP parallel RAID system. Section 3 introduces the OSM scheme and the RAID-x architecture. We also compare RAID-x with RAID-1 for designing distributed disk arrays. Section 4 describes the architecture of the cooperative disk drivers and data consistency checking mechanisms.

Section 5 presents the benchmark performance results obtained on the Trojans cluster. Section 6 explains the striped staggering checkpointing scheme we developed on top of RAID-x. Section 7 gives out the preliminary experimental results on striped checkpointing overhead and the analysis of reliability issue of proposed checkpointing scheme. Section 8 summaries the contributions and identifies extended research work.

2. USC Trojans Cluster Architecture

The prototype Trojans cluster was built with 16 Pentium PCs (Pentium II 400MHz) running the Linux operating system (Redhat Linux 6.0 with kernel 2.2.5). These PC nodes are connected by a 100 Mbps Fast Ethernet switch.

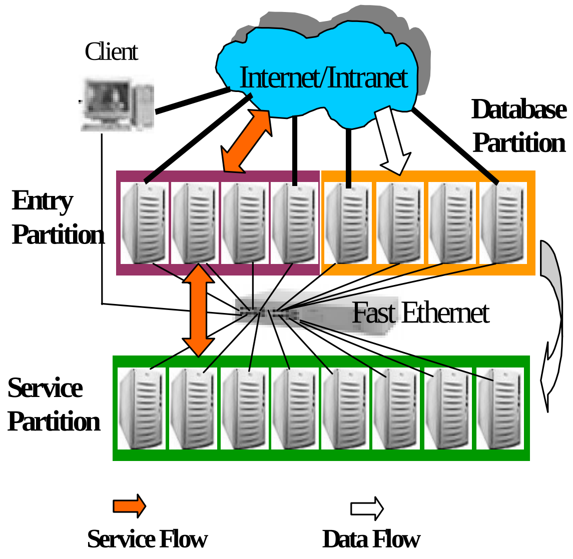
At present, each node is attached with a 10-GB disk. With 16 nodes, the total capacity of the disk array is 160 GB. All 16 disks form a single I/O space. Figure 1a shows the front view of the prototype Trojans cluster. This cluster is connected to Internet over fiber links.

As illustrated in Fig.1b, we subdivide the cluster nodes into three functional classes. The *entry partition* is for the user to access the cluster through Internet/Intranet. Nodes in the *service partition* provide the services requested by users. The *database partition* supports database or information accesses operations. Nodes in the three partitions can be dynamically reconfigured to suit special application demands.

To build a distributed RAID with a SIOS, our research objectives are identified in three aspects: (i) A single address space for all data blocks in the cluster. This means that the users can utilize all disk storage in a cluster without knowing the physical locations of the data blocks referenced or of the files used. (ii) High scalability, availability, and compatibility with current cluster architectures and applications must be maintained. (iii) Remote disk I/O operations should have performance at least comparable to that of local disk I/O operations.

Previous approaches to achieve SIOS were attempted at the user level, file-system level, and device-driver level. The *user-level approach* has the lowest cost and higher portability across different platforms. The Parallel Virtual File System (PVFS) [18] and the Remote I/O project [9] are two examples. However, this approach does introduce two problems: First, users still have to use specific APIs and identifiers to exploit full functionality of the packages. Second, using system calls to perform network and file I/O are too expensive to meet real-time or cluster computing requirements.

(a) A front-view of the Trojans Cluster



(b) I/O-centric cluster architecture

Fig. 1. Trojans cluster built at USC Internet and Cluster Computing Laboratory

Distributed file systems provide another approach to achieving SIOS to the users. Users can access remote data as if it is accessed locally. The serverless xFS system developed at Berkeley [2] and the Solaris MC project are good examples. However, this approach has its own shortcomings.

Changing the file system does not guarantee high compatibility with current applications. This will discourage the deployment of the distributed file systems in clusters. What we want to achieve is a SIOS with an unmodified file system to achieve high portability with a low cost/performance ratio.

Device-driver level designs provide SIOS not only to the users, but also to the file system. We choose this approach, because it solves most of the above problems and shortcomings. Digital Petal project [17] uses user level device driver design to enable remote I/O access.

All physically distributed disks can be viewed as a collection of virtual disks. Each virtual disk can be accessed as if it is a local disk. Petal developed a distributed file system, called Frangipani [27]. In Petal, the actual data transfer is handled at the user level.

We have developed *Cooperative Device Drivers* (CDD). These drivers work cooperatively at the kernel level. Data consistency is maintained by the CDD. Unmodified file system is used to achieve high portability and compatibility.

The development of the RAID-x architecture was inspired by previous projects. The pioneering RAID work at Berkeley [2][6][8] and at CMU [10], the TickerTAIP project [3], the Tertiary Disk project [25], chained declustering [12], and Petal project [17] all have influenced our design philosophy.

Our RAID-x design appeals especially to serverless clusters. The major innovation in our design lies in the cooperation of distributed disks in a serverless cluster environment. The cooperation is established at the Linux kernel level, rather in the user space.

Petal and Tertiary Disk achieve the SIOS at the levels of user level device drivers and xFS file system, respectively. The Digital Petal virtual disks was built in 1996, the Berkeley Tertiary Disk project was reported in 1998, the Princeton TickerTAIP parallel RAID was designed at 1993, and our RAID-x built at USC Trojans project in 1999.

The entries in Table 1 distinguish the four parallel and distributed RAID architectures in four aspects. All four I/O subsystems support SIOS, however by quite different mechanisms. All four parallel and distributed RAIDs support parallel disk I/O at the block level.

The first distinction among the four distributed RAIDs lies in their architectures. The Petal virtual disk array uses chained declustering, Tertiary Disk applies the RAID-5, TickerTAIP uses parallel disk array controllers within single RAID server to implement parallel RAID-5, and we use the new RAID-x architecture.

Our major contributions lie in the creation of the OSM and CDD mechanisms. The enabling mechanisms for SIOS are also quite different among the four architectures. TickerTAIP achieves SIOS by event-driven simulation among all the worker nodes. We realize the SIOS with cooperative device driver at the Linux kernel level.

Table 1 Parallel and Distributed RAID Projects at USC, Princeton, Digital and Berkeley

System Attributes	**USC Trojans RAID-x**	**Princeton TickerTAIP [3]**	**Digital Petal [17]**	**Berkeley Tertiary Disk [26]**
RAID Architecture Environment	Orthogonal striping and mirroring over The RAID-x in a Linux cluster	RAID-5 with multiple controllers in a single server	Chained declustering in an Unix cluster	RAID-5 built with a Solaris PC cluster
Enabling Mechanism for SIOS	Cooperative device drivers in Linux kernels	Single server implements the SIOS directly	Petal device drivers at user level	xFS storage servers at file system level
Data Consistency Checking	Locks at device driver level	Sequencing of user requests	Supported by Frangipani file system	Locks in the xFS file system
Communication Mechanism	TCP/IP Sockets	Not Available	UDP/IP Sockets	RPC at user level

Even both Petal and RAID-x choose the device driver approach, their implementations are very different under UNIX user level and Linux kernel level. Petal does provide a global name space for logical disks in the cluster. We want to extend the global name space to each data block in the cluster.

The four RAID architectures differ in their handling of the data consistency problem in establishing a distributed file management system. We implemented the lock mechanisms within the device drivers. Our performance results are generated in Linux cluster environment.

For inter-node communications, we use the TCP/IP sockets. Regardless of their differences, we believe that hardware and software experiences learned from distributed RAID projects will be complementary to each other in many aspects.

For parallel writes, the RAID-x has lower access times than RAID-1. These claims are based on benchmark results to be presented in section 5. To sum up, the RAID-x scheme demonstrates scalable I/O bandwidth with much reduced latency in a 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, because all distributed disks form a SIOS. No heavy cross-space system calls are needed to perform remote file accesses.

3. Orthogonal Striping and Mirroring

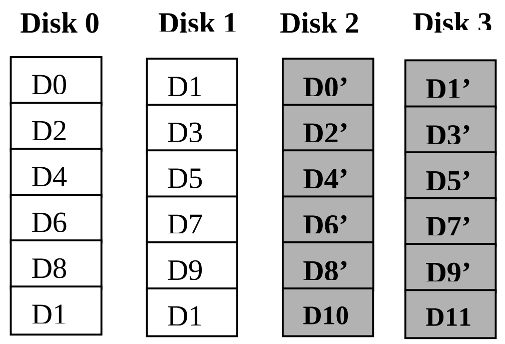
Over the years, many techniques have been developed to overcome the small-write problem [6][22], such as parity logging [24], floating parity and data [20], parity striping [7], disk caching disk [13], log-structured disk subsystem [19] and chained declustering [12]. The concept of OSM started with our earlier work [16].

In this paper, we present the design details of RAID-x and prove its effectiveness through experimentation. Figure 2 shows the architecture of RAID-x (Fig.2b) along with RAID-1 (Fig.2a) architectures. The original *data blocks* are denoted as Di in the unshaded boxes. The corresponding *image blocks* are distinguished with primes, such as Di' in the shaded boxes. The RAID-x completely avoids the small write problem.

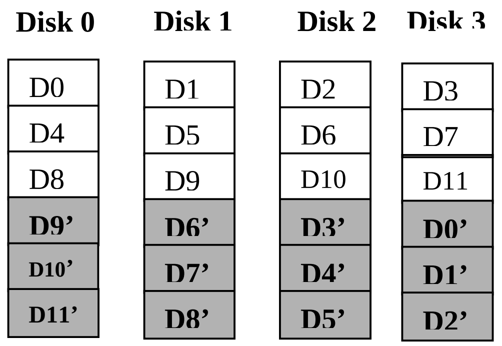
As shown in Fig.2b, data blocks in RAID-x are striped across the disks on the top half of the disk array. Low latency and high bandwidth of RAID-0 are preserved in RAID-x architecture. The image blocks of other data blocks in the same stripe are clustered in the same disk vertically. All image blocks occupy the lower half of the disk array. On a RAID-x, the images are copied and updated at the background, thus saving the overhead time.

Consider the top stripe of data blocks D0, D1, D2, and D3 in Fig.2b. Their image blocks D0', D1', and D2' are stored in Disk 3, while the image block D3' in disk 2. The rule is that no data block and its image should be mapped in the same disk. Full bandwidth is 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 the small write of a single block, the writing is directed to the data block, while the image block is postponed to write to the disk until all the clustered image blocks are ready.



(a) Duplicated striping in RAID-1



(b) Orthogonal striping and mirroring in RAID-x

Fig. 2 The mirroring schemes in RAID-1 and RAID-x

We define a pair of functions for the logical data block to physical RAID mapping: *data-mapping-function* and *mirror-mapping-function*. The data-mapping-function is a one-to-one function which maps a logical RAID block address A to a physical disk address (DiskNo, StripeNo). Mirror-mapping-function maps the corresponding image block of a logical block address to a physical disk address. The A, DiskNo, and StripeNo count from 0.

We define n as the number of disks in the array, k as the number of blocks per disk, and A as the logical RAID block address. Table 2 gives out the data-mapping function and mirror-mapping function for RAID-1 and RAID-x. The notation *mod* stands for arithmetic modulo operation. Table 2 also lists the expected peak performance of two RAID architectures.

The *maximum bandwidth* of a disk array reflects the ideal case of parallel accesses of all useful data blocks. B stands for the bandwidth per disk. In the best case, a full bandwidth of nB can be delivered by RAID-x. The RAID-1 can only deliver half of the full bandwidth. The parallel read or parallel write time of a file of m blocks depends on the read or write latencies (R and W) per block, the array size n, and the file size m.

The entries given in Table 2 are expected peak performance of parallel disk I/O operations, excluding all software overhead or network delays. In case of large reads, mR/n latency is expected to perform m/n reads simultaneously for RAID-x, while RAID-1 needs to double the latency. For small read of a single block, both require R time to finish the read.

For parallel writes, as in RAID-x, the image blocks are clustered in one disk, written to the disk at the same time. That is, $m/n(n\text{-}1)$ image blocks are written together to each disk. Therefore, the large write latency is reduced to $mW/n + m/n(n\text{-}1)$.

For small writes, our RAID-x takes only W time to write the data block. The writing of the image blocks will be done later when all the stripe images are clustered at the same disk. This clustered writing can be done at the background, overlapping with the regular data writes.

Table 2 also shows the maximum number of disk failures that each disk array can tolerate. The RAID-x can tolerate single-disk failures, RAID-1 is more robust than RAID-x. The experimental results in section 6 will verify the accuracy of the expected performance.

Figure 3 illustrates an example of the two-dimensional RAID-x architecture with 3 disks attached to each node. The maximum number of disks attached to each SCSI controller is determined by the SCSI controller used. For Wide/Fast SCSI-II, 15 disks can be connected to one single SCSI controller.

In order to implement SIOS, addresses of all the data blocks are linearly continuous among all the member disks. Only the disks with same position corresponding to each node belong to one stripe group. All the disks within stripe group can be accessed in parallel.

Different stripe groups are independent. As all the disks within one node are connected through SCSI bus, different stripe group can be accessed in pipeline. The overlap degree for the different stripe group is depends on the property of SCSI bus used.

The Trojans cluster is presently being upgraded to 4 disks per node. Using 20 GB SCSI disks, the next RAID-x array will have 1.28 TB on 64 disks. In the future, the Trojans cluster will scale to hundreds of PC nodes or more, using next generation of microprocessors and Gigabit switched connections.

Using the Fast Ethernet, the aggregate I/O bandwidth is at most 12.5 MB/s. As reported in section 5, we have achieved 9.7 MB/s bandwidth for large parallel reads. This represents 78% efficiency in the cluster utilization.

Table 2 Architectural Characteristics of RAID-1 and RAID-x

Performance Indicators		**RAID-1**	**RAID-x**
Data Block mapping	**DiskNo.**	$A \bmod n/2$	$A \bmod n$
	StripeNo.	$(2A/n) \bmod k$	$(2\,A/n) \bmod k$
Mirror-mapping function	**DiskNo.**	$n/2 + A \bmod n/2$	$(-(A/(n-1)) \bmod k/2 - 1) \bmod n$
	StripeNo.	$(2A/n) \bmod k$	$k/2 + (A/(n-1)\,n)\,(n-1) + A \bmod (n-1)$
Max. Bandwidth	**Read/Write**	$nB / 2$	$n\,B$
Estimates of Parallel Read/Write Time	**Large Read**	$2mR / n$	mR / n
	Small Read	R	R
	Large Write	$2mW / n$	$mW / n + m / n(n\text{-}1)$
	Small Write	W	$\approx W$
Max. Fault Coverage		$n/2$ disk failures	Single disk failure

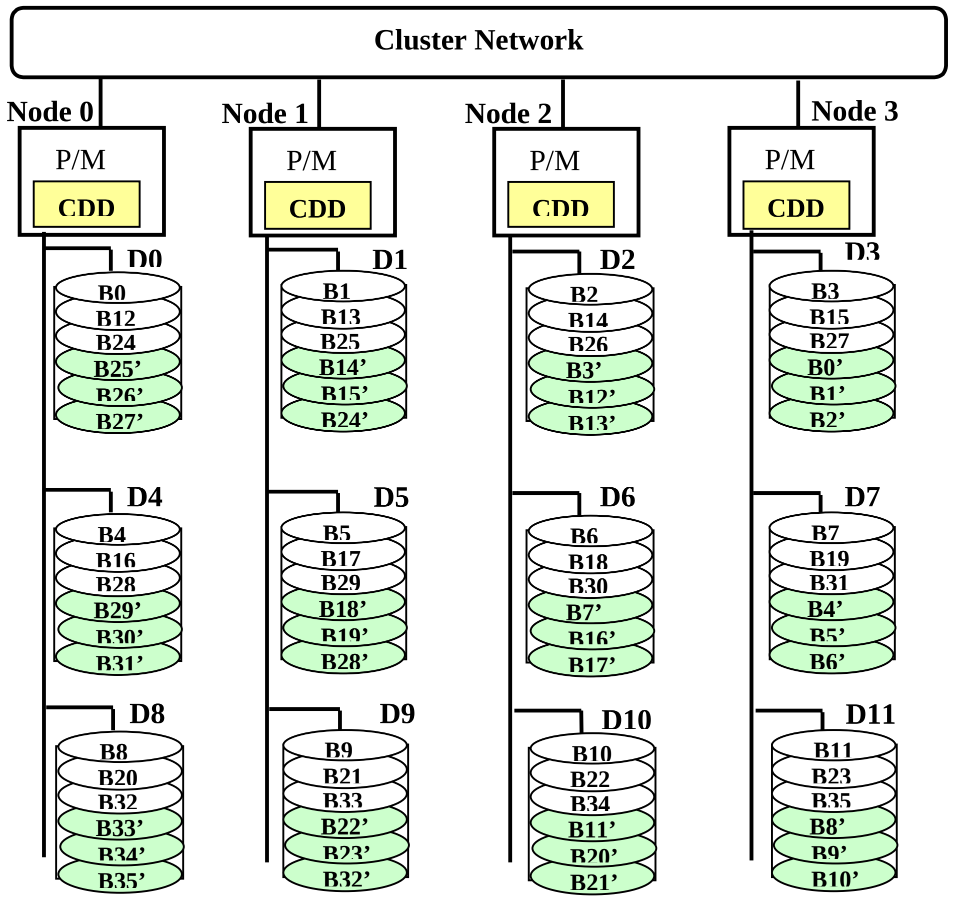


Figure 3. Distributed RAID-x architecture, shown with a 4 x 3 configuration

(P: processor, M: memory, CDD: cooperative disk drivers. All shaded blocks are mirrored images of the corresponding unshaded data blocks)

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 or multimedia applications. With an enlarged array of 128 disks, the cluster must be upgraded to a Gigabit switched connection. Based on the growing I/O bandwidth, the Trojans cluster and its RAID-x architecture show a very promising future in term of scalability and availability.

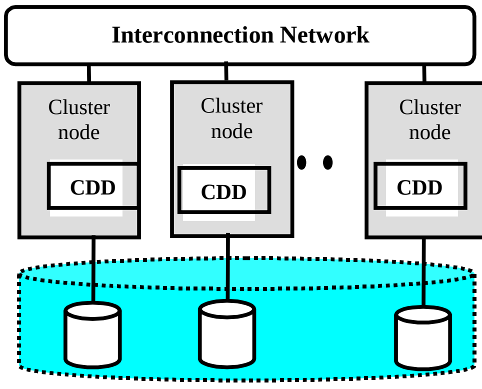
4. Cooperative Disk Drivers

The *Single I/O space* (SIOS) is crucial to building scalable cluster of computers. A loosely coupled cluster use distributed disks driven by different hosts independently. The *independent* disk drivers handle distinct I/O address spaces. Without the SIOS, remote disk I/O must be done by a sequence of time-consuming system calls through a centralized file server (such as the use of NFS) across the cluster network.

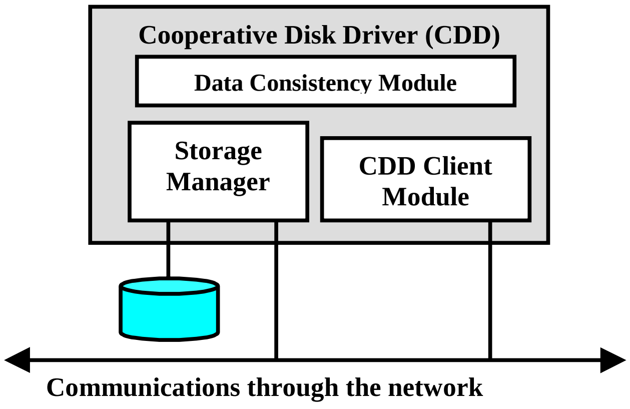
On the other hand, the CDDs work together to establish the SIOS across all physically distributed disks. Once the SIOS is established, all disks are used collectively as a single *global virtual disk* shown in Fig.4a.

Each node perceives the illusion that it has several physical disks attached locally. Figure 4b 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 disk managers.

The *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 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.



(a) A global virtual disk with a SIOS formed by cooperative disks


Cooperative Disk Driver (CDD)
Data Consistency Module
Storage Manager
CDD Client Module
Communications through the network

(b) The CDD architecture

Figure 4 Single I/O space in RAID-x built at Linux kernel level.

The Petal virtual disk array uses chained declustering, Tertiary Disk applies the RAID-5, and we use the new RAID-x architecture. The major innovations in RAID-x architecture lie in the creation of the orthogonal striping and mirroring in mapping the data blocks and their images on the distributed disks.

The OSM scheme outperforms the chained declustering scheme mainly in parallel write operations. The RAID-x scheme demonstrates scalable I/O bandwidth with much reduced latency in a cluster environment. Both Petal and Tertiary Disk achieve the SIOS at the user level. We achieved the SIOS at the Linux kernel level. Using the CDDs, the 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, because all distributed disks form SIOS. No heavy cross-space system calls are needed to perform remote file access. A device masquerading technique is adopted here. Multiple CDDs run cooperatively to redirect I/O requests to remote disks.

Data consistency problems arise when multiple cluster nodes have cached copies of the same set of data blocks. The xFS approach and the Frangipani approach maintain the data consistency at the file system level. In our design, data consistency checking is maintained at the disk driver level.

Our approach simplifies the design and implementation of distributed file management services. Data consistency is maintained by all CDDs with higher speed and efficiency at the data block level. We introduced a special lock-group table for developing distributed file management services.

Each record 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 record are granted and released atomically. This lock-group table is replicated among the data consistency modules in the CDDs. Which guarantee that file management operations are performed atomically.

5. Benchmark Performance Results

To test the cooperative operations among the CDDs residing on individual PCs, we use all 16 PCs as I/O storage servers. We use the same hardware platform to compare the relative performance of two disk array architectures: RAID-1 and RAID-x, all supported by CDDs. The NFS is used as a baseline for comparison purposes. Presently, Linux kernel version 2.2.5 supports the RAID-0, RAID-1, and RAID-5 configurations.

We implemented the RAID-x based on the RAID-0 implementation supported in the Linux kernel. This poses no difficulty in mapping the data blocks onto the top half of each disk. The mapping of the image blocks in the

RAID-x configuration is done by a special address translation subroutine residing in each CDD. To study the maximum I/O bandwidth of the disk array, the caches in the storage servers are bypassed by issuing a special *sync* command in the Linux kernel.

For reads or writes, the file size chosen was 10MB. Each block (stripe unit) in the disk is 4 KB. This means that a 10-MB file is striped uniformly across all 16 disks in consecutive stripe groups. We have performed three benchmark experiments.

The first two experiments measure the parallel I/O performance in terms of the *throughput* or the *aggregate I/O bandwidth.* The first experiment tests the throughput of RAID-x, RAID-1 and the NFS against the number of *client requests*. The second test checks the bandwidth against the disk array size for RAID-1 and RAID-x.

The distributed file system is evaluated in the third experiment using the standard Andrew Benchmark [11] consisting of a sequence of basic file system testing programs. 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 directories and the associated files. The fourth phase scans the contents of each file. The final phase compiles the files and links them together.

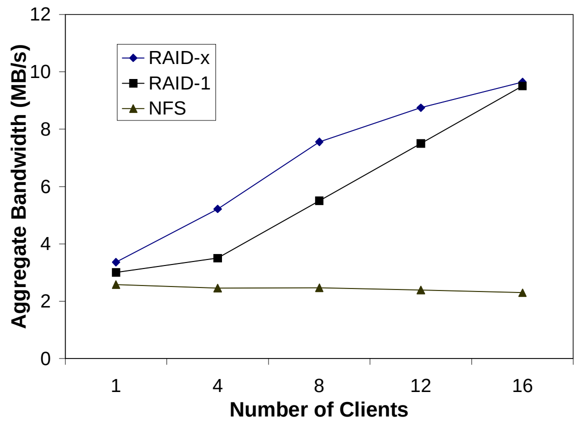
5.1. Bandwidth Results and Analysis

Figure 5 shows the performance of RAID-x, RAID-1 and NFS architectures. The results on parallel read are given in Fig. 5a. In this test, each client reads a 10MB-long file from all the disks. Therefore, the test is truly focused on the parallel I/O capability of the disk array. All the files are set to be uncached and each client only reads its own private file. All read operations are performed simultaneously, with the help of an MPI_Barrier() call.

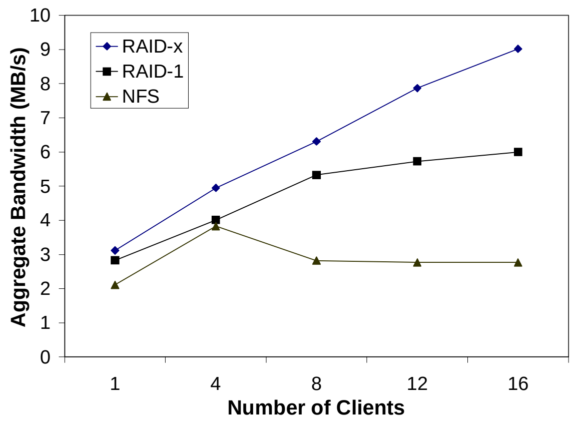
The NFS throughput is limited at 2.6 MB/s regardless of the number of clients, due to the fact that sequential I/O is performed by the NFS on a central server. As the request number increases, the NFS becoming the bottleneck shows a declining performance. RAID-x architectures scale up to a bandwidth of 9.7 MB/s for 16 clients. RAID-1 lags behind with a show of 6.33 MB/s for 16 clients.

Fig. 5b shows the write bandwidths of the RAID-x, RAID-1 and NFS subsystems. In this test, each client writes a 10MB-long file to the cache and issues a special *sync*() call to flush the data blocks to the disks. All write operations among the clients are also synchronized in these experiments.

The NFS scales in performance up to 4 requests. As the requests exceed 4, the NFS bandwidth drops to a low 2.77MB/s. For writes of a large file, RAID-x achieves the better scalability with a 9.02MB/s for 16 clients. RAID-1 saturates early to a 5.95MB/s, due to the fact that only half of the disks are used for data storage.



(a) Parallel read



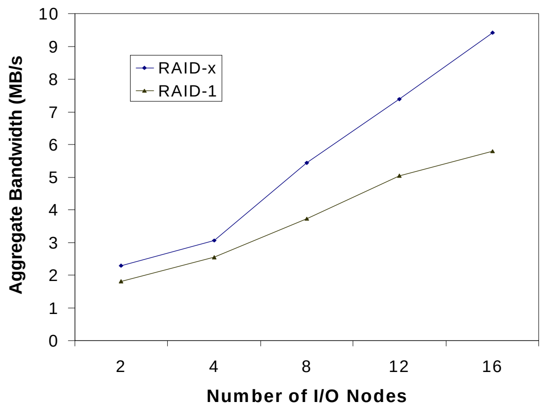
(c) Parallel write

Fig. 5 Aggregate I/O bandwidth of RAID-x, RAID-1 and NFS with increasing clients

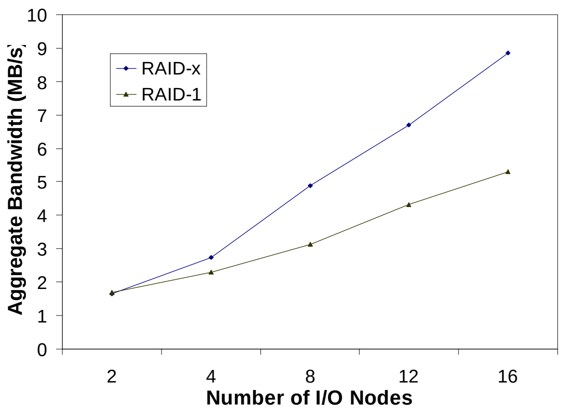
5.2. Raw I/O Performance of RAID-x

Raw I/O performance is plotted in Fig.6 against the disk array size. The results are shown for two RAID architectures. Again, all caches are bypassed in the experiments and the number of client processes is fixed at 16. The read ranking differs from the write ranking sharply in these plots.

For parallel reads (Fig. 6a), the data size has very little effects on the relative standings of two RAIDs. It is important to note that the read bandwidth of RAID-x approaches 9.7 MB/s, about 78% of 12.5 MB/s, the limit of a 100 Mbps Fast Ethernet. The difference is attributed mainly to the CDD protocol and TCP/IP overheads incurred.



(a) Parallel read



(b) Parallel write

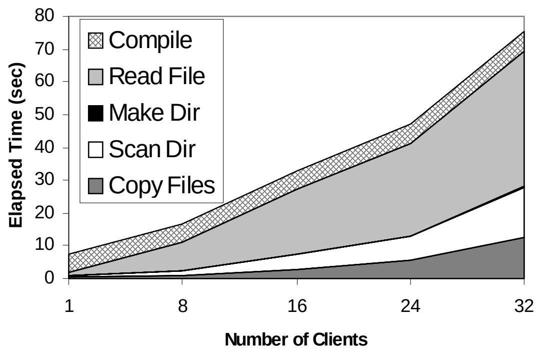
Fig. 6 Aggregate I/O bandwidth of RAID-x and RAID-1 with increasing disk numbers

For parallel writes, the large write bandwidths of RAID-x and RAID-1are 9.02MB/s and 5.72MB/s, respectively. Table 3 shows the improvement factor of 16 clients over 1 client in using the 16-node Trojans cluster. Comparing with Berkeley xFS results, our 1-client bandwidth is quite high due to well-exploited parallelism in 16-way striping across the disk array.

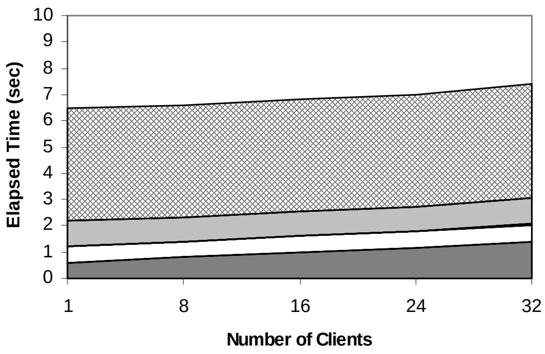
For this reason, the improvement factor is lower than that achieved by the xFS system. Again, the RAID-x demonstrated the highest improvement factor among the three distributed RAID architectures and the NFS.

5.3. Andrew Benchmark Results

Andrew benchmark tests the performance of a network file system. In this experiment, the Andrew benchmark was executed on four I/O subsystems with respect to increasing number of client requests up to 32. The performance is indicated by the elapsed time in executing Andrew benchmark on the target I/O subsystem. Figure 7 shows the benchmark results for RAID-x and NFS.



(a) NFS performance



(b) RAID-x performance

Fig. 7 Elapsed time to execute the Andrew benchmark on the Trojans cluster

These tests demonstrate how the underlying storage structures can affect the performance of the file system being supported. Each local file system on the I/O nodes mounts the "virtual" storage device provided by the CDD. The number of I/O nodes is fixed at 16. Each client only executes its own private copy of Andrew benchmark. We use the Linux ext2 local file system to keep the operations on metadata atomic.

Table 3 Achievable I/O Bandwidth and Improvement Factor on 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	3.36 MB/s	9.65 MB/s	2.87
Large Write	2.11 MB/s	2.77 MB/s	1.31	3.12 MB/s	9.02 MB/s	2.89
Small Write	2.47 MB/s	2.81 MB/s	1.34	3.22 MB/s	9.13 MB/s	2.84

Figure 7a shows the benchmark result of NFS, while Figures 7b shows the results of RAID-x. It is obvious that the elapsed time in using NFS increases sharply with the number of clients, while the RAID-x scheme can sustain the same workload. For 16 clients, the elapsed times for RAID-x and NFS are 6.8 and 33 seconds, respectively.

For 32 clients, these numbers increase to 7.41 and 75.5, respectively. From Fig. 7a, NFS shows a worsening performance especially in reading the files, scanning directories, and copying files operations. The RAID-x architectures, in contrast, do not share this weakness.

6. Striped and Staggered Checkpointing

The parallel I/O characteristic of distributed RAID-x architecture can be applied to achieve fast checkpointing in the cluster system. Striped checkpointing method is storing checkpointing file over distributed RAID-x system. To alleviate the network contention, the staggered writing skill is combined to striped checkpointing.

Simultaneous writing of multiple processes in coordinated checkpointing may cause a network contention and I/O bottleneck problem to a central stable storage. As suggested by Vaidya [28], staggered writing of the checkpoints taken by different nodes reduces the above contentions. The time lag between staggered checkpointers can alleviate the bottleneck problem associated with the central stable storage.

The basic concept of staggered checkpointing allows only one process to store the checkpoint at a time. A token is passed around to determine the timing. When a node receives the token, the node starts to store the checkpoint. After finishing checkpointing, the node passes the token to the next node.

Our work on coordinated checkpointing was inspired by the previous works by Cao [4] and associates, Chandy and Lamport [5], and Vaidya [28]. In our scheme, several nodes within the cluster form a striped group. Only the nodes within the same striped group checkpoint simultaneously and each of the groups checkpoints in a staggered way.

Figure 8 shows the concept of striped staggering in coordinated checkpointing on the RAID-x disk array. The drawing shows a 12-disk RAID-x array configured as a 2-dimensional structure, i.e. a 4 x 3 configuration. Each *stripe* corresponds to the *degree of parallelism* (DOP) in concurrent accesses of four disks in the 4 x 3 disk array.

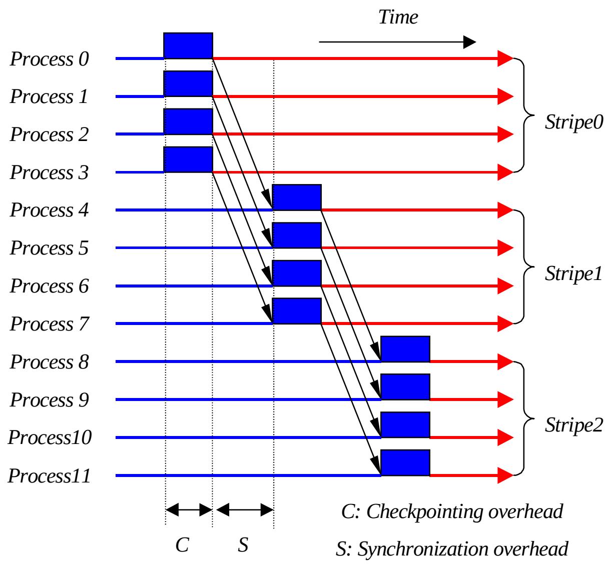


Fig. 8. Striped checkpointing with staggering on a distributed RAID-x

Successive stripes are accessed in a staggered manner from different stripes on successive 4-disk groups, as demonstrated in Fig.3. Staggering implies pipelined accesses of the disk array. We first proposed the idea of striped checkpointing in [23]. There exists trade-off between stripe parallelism and staggering depth.

For example, the layout in Fig.8 can be reconfigured from 4 x 3 to a 6 x 2 configuration, if needed. Higher DOP leads to higher aggregate disk bandwidth. Higher staggering degree can cope better the network contention problem. The staggered writing way can reduce the average checkpointing overhead. However, in the case of blocking algorithm, the staggered writing method also introduces the synchronization time.

Although blocking algorithm is the simpler than non-blocking algorithm to achieve coordinated checkpointing in parallel processing, it suffers from large amount of overhead. Every node should be blocked during the checkpointing procedure. The basic idea is to shut down all processes temporary to define consistent state. After all the processes are blocked and all the messages are clearly delivered, the global checkpoints are stored. In the staggered writing case, the blocked time increases according to the number of node.

7. Overhead and Reliability Analysis

Figure 9 shows the advantage of striped staggering on distributed disk array, as compared with staggering in Vaidya scheme [28] on a centralized disk and the conventional approach using the NFS server. These preliminary results were measured on the small prototype Trojans cluster.

Our striped checkpointing scheme has the lowest overhead, especially when the checkpoint files becomes very large. Through continued experiments on the enlarged 64-disk RAID-x cluster, we will reveal more experimental results on the checkpointing overhead and rollback recovery latency.

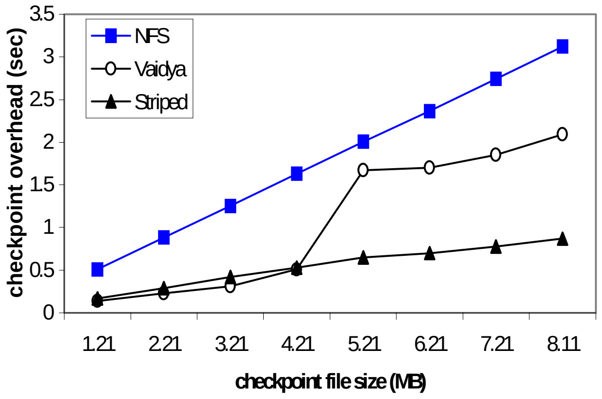


Fig. 9 Checkpointing overhead of staggered writing on distributed RAIDs

Table 4 summarizes three checkpointing schemes we have compared in this paper. Their advantages and shortcomings are identified. Suitable applications for each checkpointing scheme are also elaborated.

Using the OSM, each striped checkpointing file has its mirrored image in its local disk. For each node, transit failure can be recovered from its mirrored image in local disk. Permanent failure of a disk can be recovered from the striped checkpointing among the distributed disks.

The I/O performance in a degraded mode of OSM is the same as the RAID-0 performance in a normal mode. The striped checkpointing can be read in parallel from RAID-x. The checkpointing recovery latency can be shortened greatly.

Table 4 Summary of Three Coordinated Checkpointing Schemes

Checkpointing Scheme	Advantages	Shortcomings	Suitable applications
Simultaneous writing to a central storage (The NFS scheme)	Simple, no inconsistent state	Has network and I/O contentions, NFS is single point of failure	Small size of checkpoint, small number of nodes, low I/O operation
Staggered writing to a central storage (Vaidya scheme)	Eliminate the network and I/O contention	Network bandwidth is wasted, NFS is a single point of failure	Small size of checkpointers, small number of nodes, low I/O operations
Striped staggering checkpointing on any distributed RAID (Our scheme)	Eliminate network and I/O contentions, low checkpoint overhead, fully utilize network bandwidth, tolerate multiple failures among stripe groups	Can not tolerate more node failures within each stripe group	Large size of checkpointers, large number of nodes, low communication, I/O intensive applications

According to the mirror mapping of the OSM, the proposed RAID-x architecture can recover from any single disk failure in each stripe group. The total number of disk failure depends on the number of stripe groups to be accessed. For the 4 x 3 configuration in Fig.3, three disk failures in three stripe groups can be tolerated. An indepth analysis of the reliability of the proposed checkpointing RAID-x architecture is given in [23].

8. Conclusions

The development of the new RAID-x architecture was inspired by several research projects. The xFS and the Tertiary Disk projects at Berkeley [26], and the Petal project at Compaq Digital [17], all have influenced our design philosophy. The main difference between our approach and these projects is that we use the orthogonal striping and mirroring (OSM) to preserve both parallel disk accesses and staggered (pipelined) checkpointing of successive stripes.

We built data consistency checking in the device driver level. The CDDs work cooperatively to perform data transferring and consistency checking. With the support of CDDs, the design of a distributed file system can be focused on the concurrent file access policies and the related performance issues. In this case, the complexity of the distributed file system can be greatly reduced Our SIOS disk array separates the I/O subsystem into a distributed file system and a set of distributed CDDs.

All SSI services are provided by the CDDs while the file system modification is reduced to a minimum. Furthermore, some desired SSI services for cluster computing can be built on top of the SIOS. In this aspect, the SIOS is a very powerful middleware infrastructure to achieve single-system image. Benchmark performance results show that our distributed RAID can achieve scalability, performance, and availability in cluster computing.

The RAID-x outperforms the RAID-1 in the Linux cluster environment. For parallel reads with 16 active clients, the RAID-x achieved 9.7 MB/s throughput, 1.5 and 3.7 times higher than using RAID-1 and NFS, respectively. Running the Andrew benchmark, RAID-x results in a 17% cut in elapsed time, compared with that experienced on a RAID-1. The achieved throughput corresponds to 78% of the peak bandwidth deliverable by the Fast Ethernet. Scalable I/O bandwidth makes the RAID-x especially appealing to I/O-centric cluster applications.

The OSM mechanisms can be built not only on Linux PC clusters, but also on any Unix workstation clusters. These architectural features differ from the user-level designs in Berkeley Tertiary Disk and Digital Petal virtual disks. The new mechanisms support not only *single I/O space*, but also *distributed shared memory*, *checkpointing*, and *distributed file management* at the kernel level without using cross-space system calls.

The prototype RAID-x has the following open issues yet to be solved in future R/D efforts. These extended works are among the tasks planned in the next phase of our Trojans cluster project.

(1). We expect even higher performance as we continue improving the CDD protocol. The current hand shaking protocol could be improved with prefetching techniques. The TCP/IP used in our prototype is known for its high overhead. Plan is underway to port the whole cluster system with a low-latency protocol, expecting to further reduce the communication overhead.

(2). We plan to design a distributed file system with I/O load balancing capabilities along with an enlarged distributed disk array onto our Trojans cluster in the future. In addition to consider the RAID-1, RAID-5, and RAID-x configurations, we will also consider other configurations, such as RAID-10 and chained declustering.

(3). Our PC nodes in the Trojans cluster act as clients as well as storage servers at the same time. These dual roles affect the performance of the I/O nodes. We believe that the I/O performance can be further improved with an enlarged cluster size.

(4). We plan to develop a suite of middleware with striped staggering checkpointing to support process migration. Based on future Trojans cluster configuration, more detailed analysis of the DOP and depth of staggering will be conducted.

(5). New message logging algorithms for non-blocking striped checkpointing will be developed to reduce checkpointing overhead furthermore. We also plan to design an application dependent checkpointing scheme to elaborate the efficiency of striped checkpointing.

Lots of interesting research work can be generated out of a very large disk array in real-life applications. Potential applications are encouraged in biological sequence analysis, collaborative engineering design, clusters or grids for E-commerce, specialized digital libraries, and distributed multimedia processing.

References

[1] C. Amza, A. L. Cox, S. D. Wakadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, "TreadMarks: Shared Memory Computing on Networks of Workstations", *IEEE Computer*, Vol.29, No.2, 1996, pp.18-28.

[2] 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.

[3] 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.

[4] G. Cao and M. Singhal, "On Coordinated Checkpointing in Distributed Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 12, pp. 1213-1225, Dec. 1998.

[5] K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Trans. Computer Systems*, pp. 63-75, Feb. 1985.

[6] 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.

[7] S. Chen and D. Towsley, "The Design and Evaluation of RAID 5 and Parity Stripping Disk Array Architecture", *Journal of Parallel and Distributed Computing*, Vol.17, 1993, pp.58-74.

[8] 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.

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

[10] 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 Langagues and Operating Systems*, 1998.

[11] 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.

[12] 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.

[13] Y. Hu and Q. Yang, "DCD - Disk Caching Disk: A New Approach for Boosting I/O Performance", *Proc. of the 23rd International Symp. On Computer Architecture*, 1996, pp.169-177.

[14] 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.

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

[16] H. Jin and K. Hwang, "Striped Mirroring Disk Array", *Journal of Systems Architecture*, Elsevier Science, The Netherlands, March 2000.

[17] 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.

[18] W. B. Ligon and R. B. Ross. "An Overview of the Parallel Virtual File System". *Proceedings of the 1999 Extreme Linux Workshop*, June 1999.

[19] B. McNutt, "Background Data Movement in a Log-Structured Disk Subsystem", *IBM Journal of Research and Development*, Vol.38, No.1, January 1994, pp.47-58.

[20] J. Menon, J. Roche and J. Kason, "Floating Parity and Data Disk Arrays", *Journals of Parallel and Distributed Computing*, January 1993.

[21] 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.

[22] RAID Advisory Board, *The RAIDbook*, Seventh Edition, December 1998.

[23] W. Ro and K. Hwang, "Striped Staggering for Coordinated Checkpointing on Distributed RAIDs", *Technical Report*, Department of EE-Systems, University of Southern California, Feb.10, 2000.

[24] D. Stodolsky, M. Holland, W. V. Courtright II and G. A. Gibson, "Parity-Logging Disk arrays", *ACM Trans. on Computer Systems*, Vol.12, No.3, August 1994, pp.206-235.

[25] 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.

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

[27] 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.

[28] N. H. Vaidya, "Staggered Consistent Checkpointing", *IEEE Transactions on Parallel and Distributed Systems*, 1999, Vol. 10, No. 7, pp. 694- 702.

Biographical Sketches:

Kai Hwang 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, and parallel processing. He is a founding Editor of the *Journal of Parallel and Distributed Computing.* He received the Outstanding Achievement Award from the PDPTA in 1996.

He has published six books and over 160 technical papers in computer science and engineering. His current research interest lies in network-based PC or workstation clusters and the middleware support for security, availability, and single-system-image services. He can be reached by Email: kaihwang@usc.edu.

Hai Jin is an Associate 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. 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 USC.

A member of IEEE and ACM, he served as program committee chair of APSCC'2000. He has co-authored three books and published more than 30 papers in international journals and conferences. His research interests cover parallel I/O, RAID architecture design, fault tolerance, and cluster benchmark experiments. Contact him at hjin@ceng.usc.edu.

Roy Ho is currently working on his M. Phil. degree in the Computer Science and Information Systems Department, University of Hong Kong. He receives the B.S. degree in Computer Engineering from the University of Hong Kong in 1998. He has participated in this work, while he was visiting USC in Fall 1999. His research interest lies in computer architecture and scalable cluster computing. He can be reached by Email: scho@csis.hku.hk.

Wonwoo Ro presently works as a Ph.D. research assistant in the Department of Electrical Engineering at USC. He received the B.S. degree in Electrical Engineering from Yonsei University, Seoul, Korea, in 1996. He received the M.S. degree in Electrical Engineering from USC in 1999. His current research interest includes scalable cluster computing, checkpointing and fault tolerance. He can be reached by Email: wro@usc.edu.