Adaptive Sector Grouping to Reduce False Sharing in Distributed RAID

Hai Jin and **Kai Hwang**
Internet and Cluster Computing Laboratory
University of Southern California
Los Angeles, CA 90089 USA
Email: {hjin, kaihwang}@usc.edu

Abstract:

Distributed *redundant array of inexpensive disks* (RAID) is often embedded in a cluster architecture. In a centralized RAID subsystem, the false sharing problem does not exist, because the disk array allows only mutually exclusive access by one user at a time. However, the problem does exist in a distributed RAID architecture, because multiple accesses may occur simultaneously in a distributed environment. This problem will seriously limit the effectiveness of collective I/O operations in network-based, cluster computing. Traditional accesses to disks in a RAID are done at block level. The block granularity is large, say 32KB, often resulting in false sharing among fragments in the block.

The false sharing problem becomes worse when the block size or the stripe unit becomes too large. To solve this problem, we propose an adaptive sector grouping approach to accessing a distributed RAID. Each sector has a fine grain of 512 B. Multiple sectors are grouped together to match with the data block size. The grouped sector has a variable size that can be adaptively adjusted by software. Benchmark experiments reveal the positive effects of this adaptive access scheme on the performance of a RAID. Our scheme can reduce the collective I/O access time without increasing the buffer size. Both theoretical analysis and experimental results demonstrate the performance gain in using grouped sectors for fast access of distributed RAID.

Keywords: **Cluster computing, single I/O space, distributed RAID, false sharing, parallel I/O, performance evaluation, and benchmark experiments**

Table of Contents

1. Introduction

The speed gap between processors and disk accesses has resulted in the development of the I/O subsystem known as *redundant arrays of inexpensive disks* (RAID) [8][12][27][39]. The disk arrays are capable of providing higher bandwidth, reliability, and availability over a single large disk. Traditional RAIDs are mostly constructed as a centralized I/O subsystem, which only allows only mutually exclusive access by one user at a time. A disk array protects users from loss of data in one or more disk failures by having redundant or parity information in the array. The redundancy is used to reconstruct data lost on crashed disks after power outage, hardware or software problems.

With the rapid growth of PC and workstation clusters, more research is focused on creating a *single system image* (SSI) in cluster I/O operations. Distributed RAID are often built as an embedded subsystem with a single address space in a *cluster* of computers [5][18]. Distributed RAIDs offer the following advantages over the centralized RAID: First, it can provide higher availability, especially for the site disaster tolerance [37]. Second, it supports parallel I/O processing in clusters [4]. Third, PC or workstation clusters offer a single I/O space on top of the distributed-memory architecture. Distributed RAID provides better understanding and implementation of SIOS [16][17], which is one of the essential SSI characteristics of clusters [17][32].

In a centralized RAID subsystem, the false sharing problem [1][3][10][19][25][26][40] does not exist, because the disk array only allows mutually exclusive access by one user at a time. However, the problem does exist in a distributed RAID, because multiple accesses may occur simultaneously in a distributed environment. The problem will seriously limit the effectiveness of collective I/O operations in network-based cluster computing. Traditional accesses to disks in a RAID are done at 32 KB block level. This block granularity is too large, often resulting in false sharing among different segments in the block. The problem becomes even worse when the block size or the stripe unit becomes too large.

To solve this false sharing problem, we propose an adaptive sector grouping approach to accessing a distributed RAID. Each sector has a fine grain of 512 B. Multiple sectors are grouped together to match with data block size. The grouped sector has a variable size that can be adaptively adjusted by software. Our access scheme appeals to I/O intensive applications in clusters, such as distributed multimedia processing applications [31][36] and multi-agent network security applications in E-commerce [11].

In this paper, we concentrate on solving the false sharing problem in a distributed RAID. False sharing reduces performance in any shared memory or shared I/O subsystem, especially when the subsystem assumes a distributed architecture. We have studied the false sharing problem in a distributed RAID in earlier conference presentation [21][22]. This paper offers a comprehensive coverage of the *grouped sector access* approach [20] to solving the false-sharing problem on a distributed RAID in a cluster environment.

The rest of the paper is organized as follows: Section 2 introduces different architectures of distributed RAID in a cluster environment. Section 3 identifies the sources of false sharing in different RAID architectures. Section 4 present's performance metric to evaluate the effect of false sharing in distributed RAID. Section 5 presents the new sector grouping method to access a distributed RAID. The performance of the GSA scheme is analyzed in Section 6. This method reduces the false sharing effects and increases the collective I/O performance. Section 7 presents simulation results and provides some analysis. Finally, we conclude with some remarks and identify further research directions.

2. Distributed RAID in Cluster Architecture

Centralized RAID is built with independent disks under the control of a single controller. Such a RAID is often attached to a central storage server or used as network-attached disks. In a serverless cluster of computers [2], no central storage server is used. The conventional client-server architecture does not apply in a serverless cluster. Instead, all clients in the cluster must divide the server functions in a distributed manner. Thus, a serverless cluster often demands a *distributed* RAID architecture, which is built over dispersed disks physically attached to the client hosts in the PC or workstation cluster.

To build a distributed RAID, one must establish: (i) a *single I/O space* (SIOS) for all disks in the cluster, (ii) high scalability, availability, and compatibility with I/O-centric cluster applications, and (iii) local and remote disk I/O operations performed with comparable latencies [16]. These requirements imply a total transparency to the users, who can utilize all disks without knowing the physical locations of the data blocks. Three architectures for building a distributed RAID are characterized below:

2.1 Disks Distributed over Cluster Nodes

This disk-array architecture is built with independent disks attached to PC or workstation hosts, *called cluster nodes*, in a cluster of computers. The RADD (redundant arrays of distributed disks) [37], tertiary disk built at Berkeley [38], Digital's Petal project [28], Swarm scalable storage system [15], and the RAID-x at USC [16] are all belonging to this category. Each cluster node accesses its attached disk on behalf of its client. Figure 1 shows a typical configuration of such a distributed RAID-5 architecture. The data layout and parity blocks are also shown in the cluster. In each row, there is a single parity-check block denoted as *P*. The remaining are data blocks denoted as Di's.

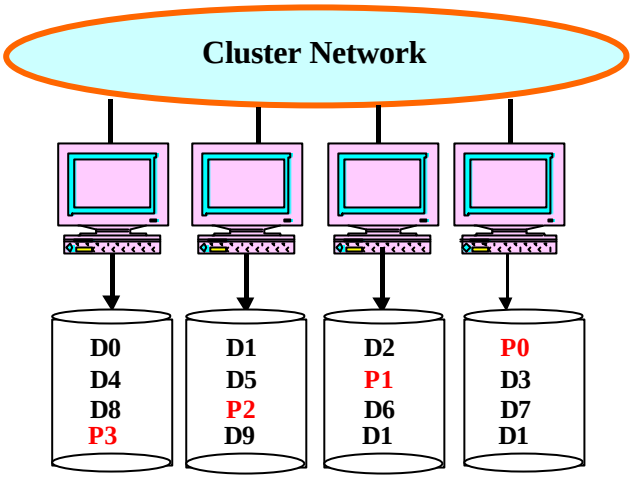


Figure 1 Distributed RAID built with local disks attached to cluster Nodes

The major advantage of the above architecture is its simplicity and low-cost to implement. Because all data and parity blocks are striped across the cluster nodes, a distributed RAID supports parallel I/O operations. By properly grouping the cluster nodes, a distributed RAID can scale from small to large sizes and leads to a scalable performance. Different implementations of such a distributed RAID provide a single I/O space in the cluster.

2.2 Network Attached Disk Array

This is the case when the disks are directly attached to a dedicated *storage area network* [9] as single parallel RAID, which is shared by all cluster nodes. Each computer or workstation in the cluster may or may not have a local disk. Even if some nodes have attached with local disks, they are accessed locally. The local disk cane be used to buffer data retrieved from the network attached RAID.

The TickerTAIP [7], Network-attached Secure Disks [13][14], active disks [34], and Storage Area Network [39] are good examples of network-attached storage system [24][30][35]. This architecture provides direct data transfer between central storage and clients in a networking environment. This architecture is more scalable than the use a central storage server by direct network striping [6]. Figure 2 shows the system architecture of a network attached RAID subsystem. Network attached disk array provides better scalability than the traditional secondary storage attached to local channels. This architecture matches better with the trend of using an open system architecture for building clusters of computers [5][18].

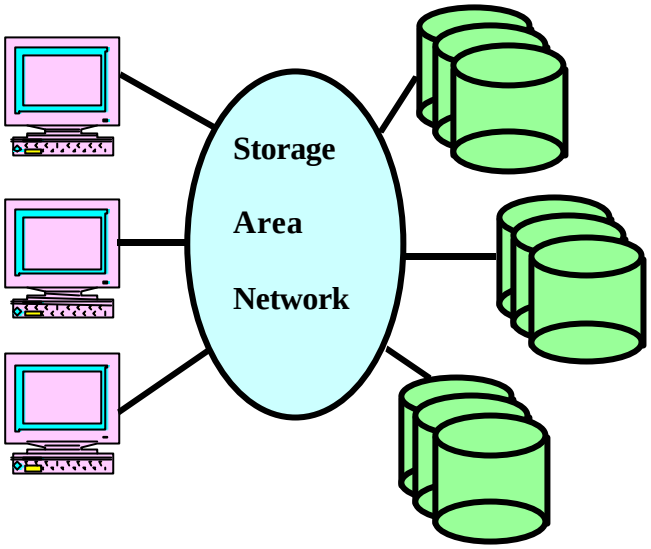


Figure 2 Distributed RAID built with network-attached disks

2.3 Hybrid Architecture for Distributed RAID

This architecture is conceptually illustrated in Fig.3, which appear in many cluster sites [17]. The cluster nodes are built with workstations or PCs. All nodes are connected by a *local area network* (LAN) or by a *system-area network* (SAN). Local disks are attached to each workstation node. Each local disk is only accessible from its own host. The network-attached RAIDs form a n*etwork-attached storage* (NAS) to be used as the stable storage. To have a SIOS, all local disks in the distributed RAID, and NAS form a single address space.

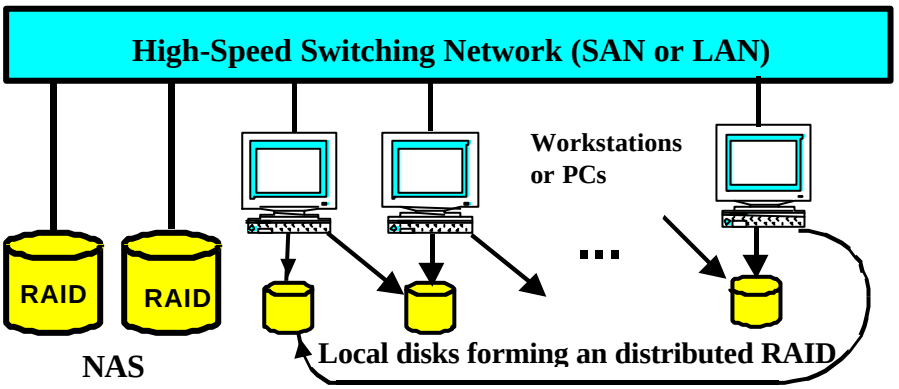


Figure 3 Hybrid architecture of a distributed RAID in a cluster

3. False Sharing in A Distributed RAID

In this section, we characterize the false sharing problem associated with distributed RAID. Any shared memory system may face the false sharing problem. The problem was first encountered in a cached multiprocessor system [26][40]. It becomes worse in NUMA (*non-uniform memory-access*) machines with *distributed shared memory* (DSM) [1][29]. Different data objects within the same coherent blocks (cache line or memory page) may be falsely shared, if the cache or page size becomes too large. False sharing has thus been blamed to contribute additional memory consistency overhead in multiprocessors.

In the case of clusters of computers, the distributed RAID desires data sharing within a single I/O space, which creates the chances of false sharing at disk block level. This problem becomes even worse than that for a NUMA machine at the DSM level. False sharing in distributed RAID is still a wide-open problem. Next we discuss the different false sharing situations for distributed RAIDs with different architectures as introduced in Section 2.

3.1 False Sharing in Parity Blocks

Parity blocks are also shared in a distributed RAID. For each write operation, the parity must be updated to protect current data blocks. Sometimes, the parity blocks are falsely shared as illustrated in Fig. 4. In each row, the parity block is denoted as P. The data blocks are marked with D. All blocks in the same row constitute a parity group. Consider the case of two software processes updating a shared blocks 2 in Fig.4, which is highlighted by an arrow.

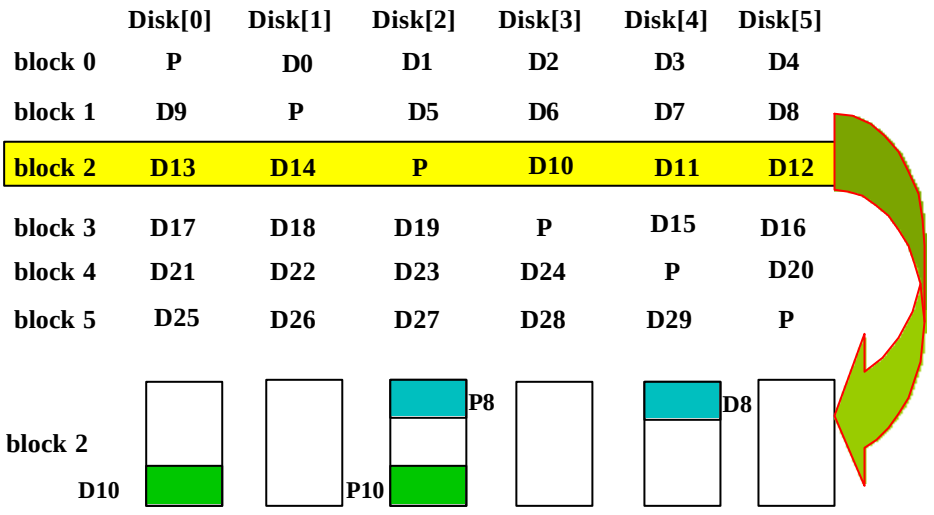


Figure 4 Parity False Sharing Distributed RAIDs

One process writes data block D8, which ends in the first several lines of block 2 in disk 4. The corresponding parity block is P8. Another process writes data block D10 that begins with

the last several lines of block 2 in disk 0. The corresponding parity block is P10. Thus parity block 2 of disk 2 is shared by both processes. This parity block is falsely shared, because each process writes to a different fragment of the data block. This triggers the modification of different fragments of the parity block.

From the above example, data access of different fragments of the same block is the main source of false sharing. This occurs quite often in distributed transaction processing applications, where small write is often performed. As the block size increases, the false sharing problem becomes worse.

We denote the write of a data segment by $W_{i,j}$, where i is the disk number and j is the segment address. The segment address j is represented by a 3-tuple, (b, o, e), where b is the block number within disk i, o is the offset of the segment, and e is ending address of the data segment.

Suppose there are two different write operation from different process, $W_{d1,\,a1}$ and $W_{d2,\,a2}$, respectively. Parity false sharing occurs when each process contains a valid copy of parity information, they both write to different disk to the same block of data, but the address of each data is different, that is both data region are not overlap. Parity false sharing occurs, if and only if the following condition is met.

$$W_{d1,a1} \wedge W_{d2,a2} \wedge (d_1 \neq d_2) \wedge (b_1 = b_2) \wedge (o_1 \neq o_2) \wedge [(e_1 < o_2) \vee (e_2 < o_1)] \qquad (1)$$

3.2 False Sharing in Data Blocks

Figure 5 shows a scenario of false sharing in data blocks in a distributed RAID. The layout of two data files is shown. File 1 occupies the entire block i of disk 0 and disk 1 and the first several lines of block i in disk 2. File 2 occupies the last several lines of block i in disk 2 and the entire block i of disk 3 and disk 4. The figure illustrates the situation that two software processes P1 and P2 access the File 1 and File 2, respectively, at the same time.

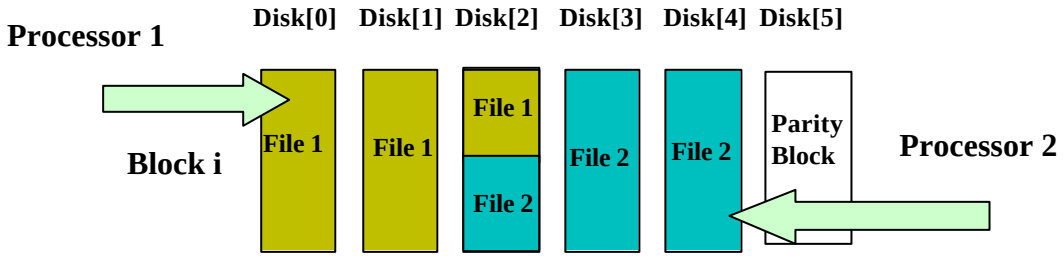


Figure 5 Data False Sharing in Distributed RAID

Each process has a local copy of block i on disk 2. Because the two processes access

different parts of the same block, there is no true sharing between them. This is called data false sharing in the distributed RAID. The main source of data false sharing is the access of different fragments of the same block. For a distributed RAID, all parity blocks are shared by all processes. In the case of data fragmentation, data false sharing will trigger parity false sharing as well.

The phenomenon of data access of different fragments of the same block occurs very common in a distributed RAID. Usually, the size of a stripe unit can be as large as one track of a disk [23], say 32 KB. It is common that a small file, say 4 KB, can not occupy the entire block. In this case, if there are more than one process accessing the same data block, data false sharing happens. Because each write operation has to modify the parity information to keep the data consistency, if data false sharing happens, parity false sharing must happen.

Data false sharing only occurs when each process has a valid copy of a coherence data block. They both write to the same block of data of the same disk, but the address of each data is different, that is both data region are not overlap. Formally, data false sharing occurs whenever the following condition is met.

$$W_{d1,a1} \wedge W_{d2,a2} \wedge (d_1 = d_2) \wedge (b_1 = b_2) \wedge (o_1 \neq o_2) \wedge [(e_1 < o_2) \vee (e_2 < o_1)] \qquad (2)$$

Note that the fact $d_1=d_2$ indicates that both processes write to the same disk.

We assume that the accesses are requested from two different cluster nodes. Whenever the data false sharing occurs, the parity false sharing occurs at the same time. Therefore, we obtain the following revised condition for all parity false sharing:

$$W_{d1,a1} \wedge W_{d2,a2} \wedge (b_1 = b_2) \wedge (o_1 \neq o_2) \wedge [(e_1 < o_2) \vee (e_2 < o_1)] \qquad (3)$$

4. Performance Effects of False Sharing

We consider the basic unit of a disk access as a sector of 512 bytes. A set of sectors forms a block, denoted as b. Note that $b_j \cap b_k = \varnothing$, if $j \neq k$. The number of disk accesses made to the sector s_i is denoted as $s_{i,a}$ and the number of write accesses is denoted as $s_{i,w}$ .Similarly, we denote the total number of accesses to block b_j as $b_{j,a}$ and the number of write accesses to the block as $b_{j,w}$. All of these counts are taken over the time interval of interest.

The *processor set* S_i is defined as the set of processors accessing a given sector s_i. The processor set of a block is defined as the union of the processor sets of all sectors in the block:

$$B_j = \bigcup_{s_i \in b_j} S_i = \{ processors\ access\ b_j \} \qquad (4)$$

Let $|S_i|$ be the cardinality of a set S_i. Let B_j^1 be the set of sectors within a given block b_j, which are accessed by a single processor. That is:

$$B_j^1 = \bigcup_{s_i \in b_j} \{S_i \mid |S_i| = 1\} = \bigcup_{s_i \in b_j} \{only\ one\ processor\ access\ s_i\ in\ b_j\} \quad (5)$$

As for most cache coherence scheme, the primary cause of coherence overhead is due to write reference. With the invalidation-based protocol, writes cause the invalidation that in turn can cause false sharing misses, and with the updated-based protocol, it is the write references that cause false sharing updates. We denote $B_{j,a}^1$ as the set of sectors within a given block b_j accessed (for read or write) by only one processor and $B_{j,w}^1$ be the set of sectors within a given block b_j written by only one processor. Therefore, $B_{j,w}^1 \subseteq B_{j,a}^1$.

We define $F(j)$ as *the degree of false sharing* of block b_j by the following expression:

$$F(j) = \frac{|B_j^1|}{|B_j|} \times \frac{|B_{j,w}^1|}{|B_{j,a}^1|} \quad (6)$$

The first ratio indicates the number of sectors accessed by only one processor over the total number of sectors accessed in a given block b_j. The second ratio indicates the number of sectors written over the total number of sectors accessed by only one processor in a given block. The product indicates the total falsely shared sectors to total sectors accessed in a given block. This is the way we measure the degree of false sharing.

Thus, sectors used in a mostly read-only fashion have $F(j)$ close to zero, while for the sectors with high write-to-access ratios have high values of $F(j)$. If the write access covers a full stripe unit, there exists no false sharing problem. Otherwise, false sharing may occur in a distributed RAID. With prefetch, the source of data false sharing is the fragment access of a prefetched stripe unit.

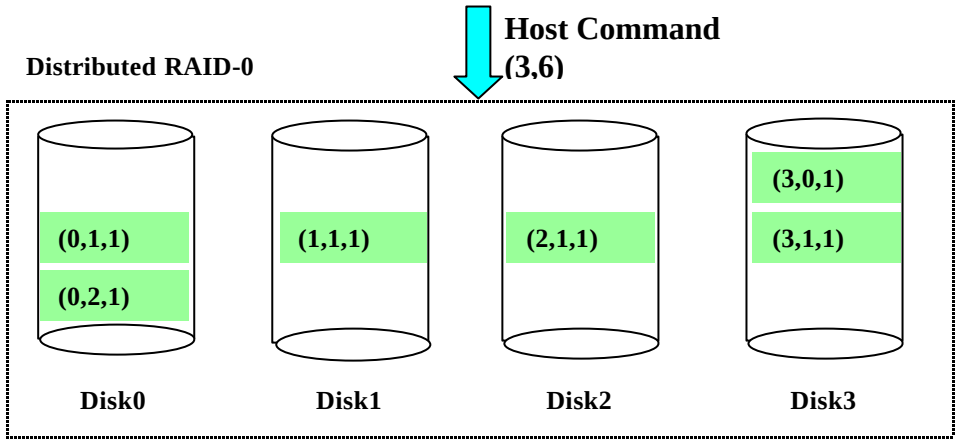
5. Grouped Sector Access (GSA) Scheme

Traditional disk I/O in a centralized RAID is accessed at the disk block level. A typical block size ranges between 4 KB and 32 KB. With such large block sizes, the false sharing problem becomes very severe. In order to reduce the false sharing effects in a distributed RAID, we propose a new parallel disk access method, called the *grouped sector access* (GSA) scheme. The implementation of this new RAID access scheme is described below for two distributed RAID architectures.

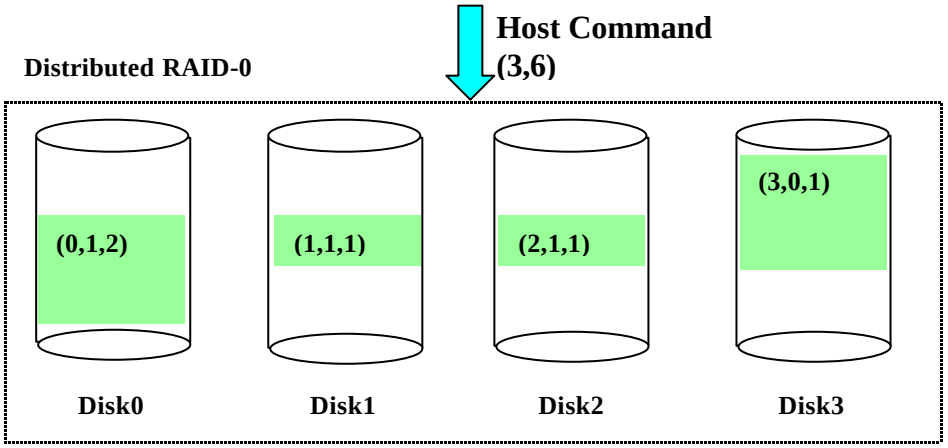
5.1 Implementation in a Distributed RAID-0

Both block-level and GSA schemes are implemented on a 4-disk RAID-0 in Fig. 6. The block-level access scheme for a distributed RAID-0 is specified as follows:

- Receive host I/O command.
- Calculate the number of I/O subcommands sent to the member disks in the RAID.
- Calculate the starting address and data length of each I/O subcommand.
- Send all the I/O subcommands to member disks to execute.



(a) Block-level access scheme



(b) The GSA scheme

Figure 6 Block-level vs. GSA schemes in accessing a distributed RAID-0

In our GSA scheme, we first decompose the host command into subcommand according to the sector size. The length of each subcommand is one sector. Before sending the subcommands to member disks, it uses a combining technique to combine the I/O subcommands to each disk to reduce the number of I/O subcommands. As in RAID-0, all the I/O subcommands are of the same type, reading or writing. There's no dependency between subcommands. The physical addresses of the subcommands are consecutive. The data length is the sum of all that of

these I/O subcommands.

We use a 2-tuple (*S*, *L*) to represent a host command. *S* is the starting address, *L* is the data length. For simplicity, we assume the unit of *S* and *L* is in the size of sector. We use a 3-tuple (*D*, *S*, *L*) to represent I/O subcommand after command decomposition. In this case, the meaning of *S* and *L* are the same as above, *D* represents the disk number. Figure 6 shows an example for traditional RAID operation and by using the GSA scheme.

5.2 Implementation in a Distributed RAID-5

In RAID-5, there exist both data blocks and parity blocks. It can still function even when one disk failed. The command decomposition in RAID-5 is far more complex than that in RAID-0. We use a 4-tuple (*T*, *D*, *S*, *L*) to represent a I/O subcommand. The meaning of *D*, *S*, *L* are the same as above. *T* represents the type of I/O subcommand, the value *R* and *W* indicates the *read* and *write* I/O subcommand, respectively. Similarly, we use a 3-tuple (*T*, *S*, *L*) to represent the host command.

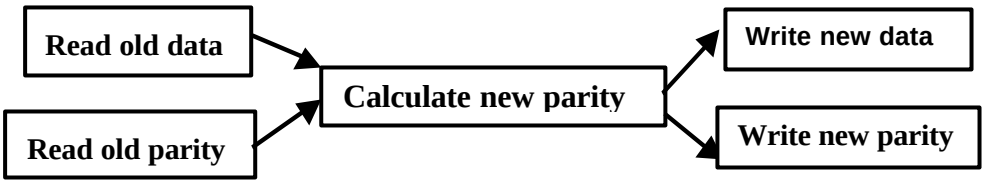
If the host command is (*R*, *S*, *L*), the command decomposition process of RAID-5 is the same as that of RAID-0. If the host command is (*W*, *S*, *L*), the command decomposition of block-level access of a distributed RAID-5 is specified as follows:

- Receive the host I/O command with the starting address and data length.
- Calculate the number of I/O subcommands. The new parity information for each parity group will generate four I/O subcommands, which are read old parity, read old data, write new data, write new parity. We call these four I/O operations a unit operation. The old parity, the old data and new data perform exclusive OR operation to generate new parity.
- Calculate the starting addresses and data lengths of each I/O subcommands.
- Send all these subcommands to member disks. Because of the data dependency between these commands, in order to avoid the *Read-Modify-Write* error, the four commands of a unit operation should execute in order. Figure 7(a) shows the execution flow of block-level RAID-5 write operations.

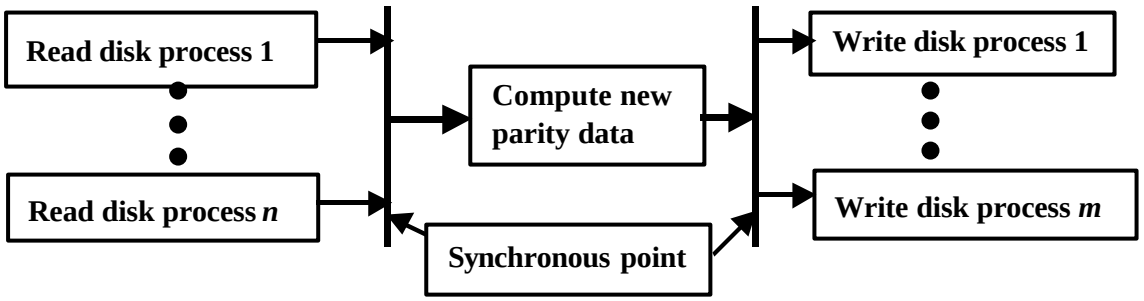
By using the GSA scheme, computation command for generating parity information should not be combined. It is used as synchronization point for the same type of I/O operation. All the other *read* or *write* subcommands to same member disk will generate a new combined I/O subcommand. Figure 7(b) shows the execution flow of RAID-5 write operation by using the GSA scheme. The command combining takes the following steps to maintain the data consistency:

- Computation of parity information or write operation should not initiate before the completion of read operation.
- The new computation of parity information should not initiate before the completion of former computation of parity information.

- Write operation should not initiate before the completion of parity update.



(a) Block-level write operations



(b) Write operation using the GSA scheme

Figure 7 Two schemes for parallel write operations in a distributed RAID-5

After the command recombination, there exist three types of I/O subcommands in the subcommand queue for each member disk. If the host command is the read operation, there exists only one read subcommand. If the host command is the write operation, there will be one write subcommand, one read subcommand and several parity computation subcommands for the corresponding member disks.

6. Performance Analysis of GSA Scheme

Listed below are major notations used in this section:

T_{block}: The execution time for RAID using traditional control method

T_{GSA}: The execution time for RAID using the GSA scheme.

N: The number of disks in RAID ($N = n + 1$, n data disks and one parity disk)

B: The stripe size (in byte)

L: Read/Write data length (counted by byte)

τ: The exclusive OR operation time

For simplicity, we assume that each disk has it own channel and can transfer data independently. The average time of each I/O access is *T*. Starting address of read/write command is at the beginning of parity group. Ignore the interrupt competition of parallel schedule, network congestion and bus arbitration.

6.1 Performance of a Distributed RAID-0

For block-level disk accesses, the read/write command from a host will generate ⌈L / B⌉ I/O subcommands. The execution time is:

$$T_{block} = \left\lceil L \Big/ (B \times n) \right\rceil \times T \tag{7}$$

For the GSA scheme, the read/write command from host will at most generate *n* I/O subcommands. The execution time is:

$$T_{GSA} = \begin{cases} T & if\ (L/(B \cdot n) \geq 1 \\ T_{block} & if\ (L/(B \cdot n) < 1 \end{cases} \tag{8}$$

6.2 Performance of a Distributed RAID-5

We analyze the time complexity of RAID-5 according to different cases.

Case 1: Host sends *read* command to RAID-5. In this case, the calculation is the same as that in RAID-0.

Case 2: Host sends *write* command to RAID-5.

For traditional command decomposition algorithm in this case, the number of full stripe write is $\left\lfloor \frac{L}{B \times n} \right\rfloor$ and the number of partial stripe write is $\left\lceil \frac{L\%(B \times n)}{B} \right\rceil$. For the full stripe write, the new parity is calculated by the exclusive OR operation of *n* data blocks. The number of exclusive OR operations is *(n* - 1*)* ´ *B*. For the partial stripe write, the new parity is the exclusive OR operation of the old data, the old parity and the new data. The total execution time is as below:

$$T_{block} = \left\lfloor \frac{L}{B \times n} \right\rfloor \times (n-1) \times B \times \boldsymbol{t} + \left\lceil \frac{L\%(B \times n)}{B} \right\rceil \times 2B \times \boldsymbol{t} + \left\lfloor \frac{L}{B \times n} \right\rfloor \times T + 2T \tag{9}$$

For GSA command decomposition algorithm, the time spent in exclusive OR operation keeps the same. But using the GSA scheme, one can reduce the number of read/write operations to just one read and one write. The total execution time is:

$$T_{GSA} = \left\lfloor \frac{L}{B \times n} \right\rfloor \times (n-1) \times B \times \boldsymbol{t} + \left\lceil \frac{L\%(B \times n)}{B} \right\rceil \times 2B \times \boldsymbol{t} + 2T \quad (10)$$

7. Benchmark Performance Results

We have developed an experimental RAID testbed in the Trojans PC cluster built at the University of Southern California [16]. We carried out the experiments by using some I/O traces as inputs to different processors in the Trojans cluster. Through this cluster, we can modify some architectural parameters over different I/O access traces. This distributed RAID testbed can be reconfigured as RAID-0, RAID-1, RAID-5, and RAID-x as reported in [16].

To explain the benchmark experiments, we tested on a small 4-disk RAID configuration of two traces driven by two 540MB Quantum SCSI disk drives. We use the index i = 0 or 1 to distinguish the two traces and use the index j = 1 or 2 to represent the disk drives. Therefore, the distributed disk array has four disks D[0,1], D[0,2], D[1,1] and D[1,2] under evaluation. We configure these four disk drives as a single-parity group following the RAID-5 architecture.

7.1 False-Sharing Benchmark Results

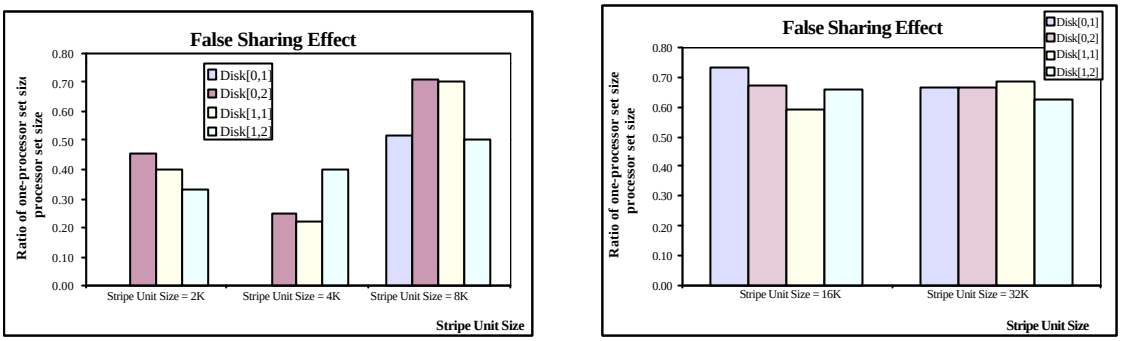
In order to evaluate the performance effect of false sharing in the distributed RAID, we need a testbed in which we can easily modify architectural parameters of distributed RAID and collect I/O access traces. We also need the I/O trace to measure the false sharing accurately in the distributed RAID. The testbed has the ability of reconfiguring as any architecture of RAID level 0, 1, 4, 5 and 10. We can easily change the architecture parameters, such as stripe unite size.

We design a program to simulate the behavior of the disk array in the cluster. We also develop a set of synthetic workloads as the workload of distributed RAID. The generation of workload is by using the I/O trace of Qbench [33], which is a disk drive benchmark designed by Quantum Co. to measure the disk service time and the channel interaction time of single disk drive as well as storage system.

For the type of I/O access, the percentage of read and write can be set flexible. In order to collect as many as I/O trace as the input of our simulation program, we can ignore the read access because read access has no contribution to the study of data sharing. In this way, we get the I/O trace and use it as the input of our simulation program to investigate the performance effect of false sharing in the distributed RAID.

We use the I/O trace of Qbench as the input of our simulation program. We configure the architecture of disk array to RAID level 5. We select the stripe unit size of disk array as 2KB, 4KB, 8KB, 16KB, 32KB, respectively. In the simulation, we use two processes, each with the synthetic workloads from Qbench I/O trace. For one process, the I/O access size each time is one sector, and for another process, the I/O access size each time is two sectors.

Figure 8 shows the performance effect of false sharing during the I/O accesses between these two processes with different stripe unit size. The *x*-axis shows different stripe unit sizes applied. The *Y*-axis shows the ratio of the processor set size for single accessed sectors over the total number of sectors accessed. This ratio reflects a major part of the false sharing effects.



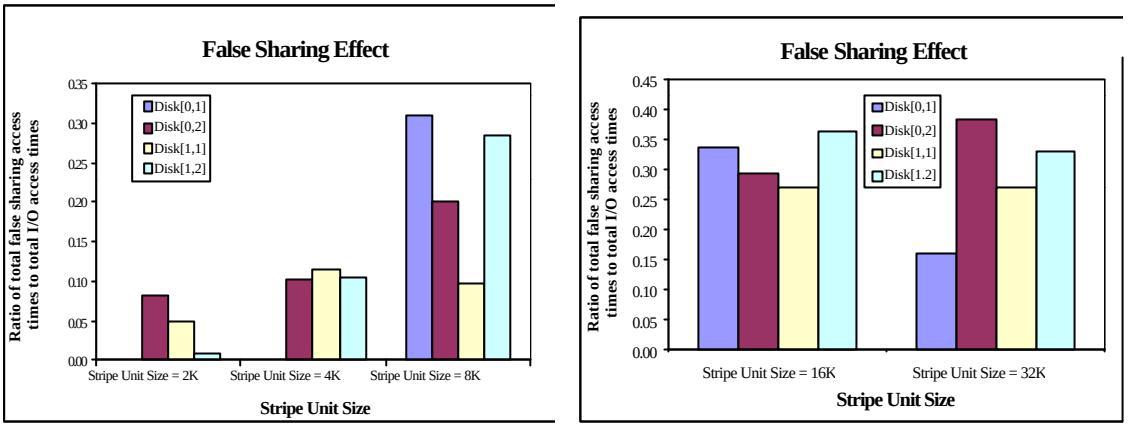
(a) Small Stripe Unit Size (b) Large Stripe Unit Size

Figure 8 False sharing effect of stripe unit size on the ratio of single-sector access over all the sectors accessed.

Figure 9 plots Eq.6 to reveal the performance effect of false sharing. The *Y*-coordinate shows the degree of false sharing. It reflects the ratio of false-sharing number to the total disk I/O requests. The plot is drawn against different stripe unit sizes.

The larger the stripe unit size, the heavy case of false sharing. This is quite different from the situation which disk array connects to the single machine. In the case of cluster or network environment, it is not the case that the large the stripe unit size, the better the performance. This is because the large size of the stripe unit, the large the data block will be cached for each processor. According to Torrellas, Lam and Hennessy [40], the false sharing phenomena will become severer when the cached data block achieves to certain size.

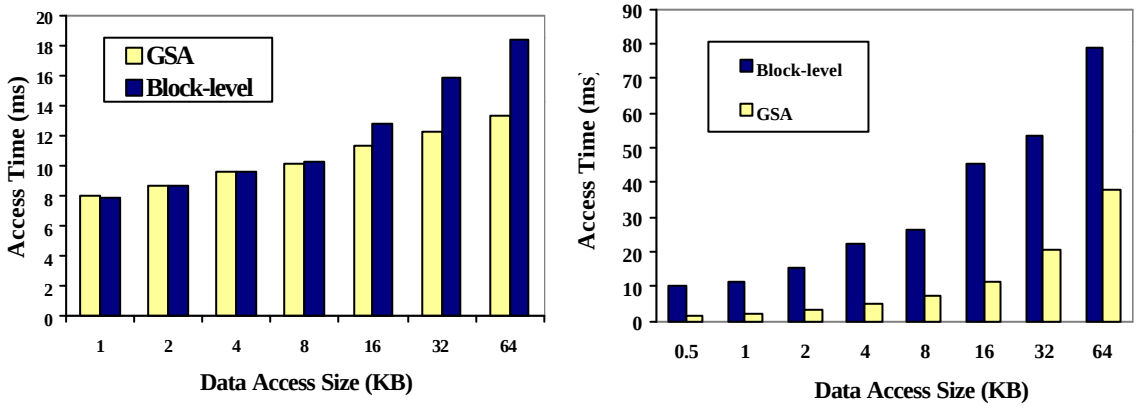
Our benchmark results suggest that the best choice of the block size is 4 KB. With this size, the false sharing percentage is as low as 12%. A block size greater than 8 KB will severely degrade the performance. The choice of block size is benchmark dependent. Experimenting with a different workload, the results may be different. But the shifting in block size should not be too far from our research findings.



(a) Small Stripe Unit Size (b) Large Stripe Unit Size

Figure 9 False sharing effects on different stripe unit sizes

The GSA scheme suggests the use of *sectors*, instead of blocks, as the basic grouping unit in RAID architecture. A sector has been considered the basic access element in disk. The small sector size implies many more disk accesses, while GSA combines these small disk accesses to one large disk access. This situation is similar to reducing the false sharing problem by replacing page-level memory sharing by cache lines in a DSM system. Figure 10 shows the data access time by using GSA sector grouping method compared with the access time of using blocks of 4 KB as a unit for data sharing among the disks in the RAID-0 and RAID-5.



(a) RAID-0 (b) RAID-5

Figure 10. Comparison of data access times between the GSA and block-level schemes

The GSA scheme uses a command combining technique to reduce the number of I/O accesses by 70% in the best case. This approach can be applied to optimize the selection of the

block size or the sector size. The results can guide the design of single I/O space in clusters.

7.3 Performance of Sector-sized Striping

Although sector-sized stripe unit avoids the false sharing problem in a distributed RAID, our GSA scheme outperforms the single-sector scheme. Let T_{tran} be the time to transfer 1KB data. From a host to a disk array. The time T_{conn} is used to setup a connection between a source node and a destination node. The time used for I/O operation in each node is T_i. When using our access scheme, the total execution time T_1 is calculated as follows:

$$T_1 = L \times (T_{conn} + T_{trans}) + \sum_{i=1}^{L} T_i \tag{11}$$

$$T_{GSA} = K \times T_{conn} + L \times T_{trans} + \sum_{i=1}^{K} T_i \qquad K = \left\lceil \frac{L}{B} \right\rceil \tag{12}$$

The form our experiments on a cluster with 4 nodes. The time transferring 1 sector data is 25 ms. The time used to setup connection is 400 ms. One node in the cluster sends data request with length of 2, 4, 8, 16, 32, and 64 KB, respectively. Stripe unit size for the GSA controlled RAID changes from 2 sectors, to 4, 8, and 16 sectors, respectively. The difference of T_{GSA} and T_1 is shown in Figure 11.

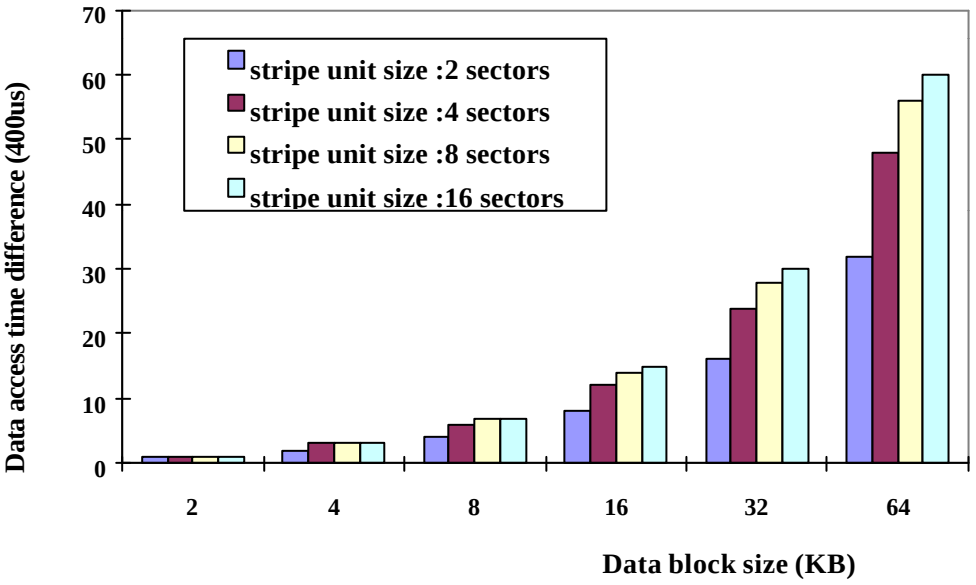


Figure 11 Time difference between block-level access using sector-sized striping and the GSA scheme in a distributed RAID-5

8. Conclusions

Finally, we summarize the research contributions of this work and identify the topics for further research. Our adaptive sector-grouping scheme has the following distinct advantages over the traditional block-level access of a distributed RAID:

(1). The embedded RAID in a cluster desires a single I/O space among all distributed local disks. In such a distributed subsystem, the false sharing problem severely limits the cluster I/O performance. Solving this problem greatly improves the on-line performance of a large distributed storage system.

(2). We offer a set-theoretic metric to reveal the degree of false sharing. This metric is measurable at run-time for implementing the grouping process adaptively. Distributed caching and prefetching strategies also affect the I/O performance.

(3). Using the GSA scheme can reduce the total number of I/O subcommands. This will effectively eliminate bringing irrelevant data to the local disk buffer. The experimental results have verified the accuracy of the set-theoretic prediction.

(4). The adaptive sector-grouping scheme reduces the granularity of disk sharing and thus reduces the chance of false sharing effectively. Furthermore, the scheme offers better disk space utilization by eliminating unshared fragments (sectors) in the group. The sector group has a variable size adaptively adjustable by software.

(5). Our scheme reduces the collective I/O access time without increasing individual disk buffer size. Both theoretical analysis and experimental results have demonstrated the performance gain. The larger is the stripe unit size, the higher is the expected I/O performance. The false sharing effect will increase after the stripe unit increases beyond certain limit, depending on applications.

Identified below are two shortcomings, which imply that additional work is needed for further research and development. These tasks will be among our continued efforts:

(1). Our benchmark experiments were carried out on a 4-disk array embedded in a small cluster. For academic purpose, this is sufficient to prove the concept. However, we plan to test the GSA scheme on a larger RAID, say 32 disks or more in the future. Results from extended experiments will further verify the performance gains.

(2). Industrial prototyping is needed for implementing the GSA scheme. More experiments are thus needed to prove the scalability of the access scheme. The USC research team will collaborate with the storage industry for possible joint development in this direction. Contact kaihwang@usc.edu in this regard.

References

[1] C. Amza, A. L. Cox, S. D. Warkadas, P. Kelehr, H. Lu, R. Rajamony, W. Yu and W. Zwaenepoel, "TreadMarks: Shared Memory Computing on Networks of Workstations", *IEEE Computer*, 29(2): 18-28, 1996.

[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] W. J. Bolosky and M. L. Scott, "False Sharing and its Effect on Shared Memory Performance", *Proceedings of the USENIX Symposium on Experiences with Distributed and Multiprocessor Systems*, Sept. 1993, pp.57-72.

[4] P. Brezany, *Input/Output Intensive Massively Parallel Computing: Language Support, Automatic Parallelization, Advanced Optimization, and Runtime Systems*, Lecture Notes in Computer Science, Vol.1220, Springer-Verlag, 1997.

[5] Rajkumar Buyya (ed.). *High Performance Cluster Computing – Architectures and Systems*, Prentice Hall PTR, New Jersey, 1999.

[6] L. F. Cabrera, and D. D. E. Long, "Swift: Using distributed disk striping to provide high I/O data rates", *Proceedings of USENIX Computing Systems*, Fall 1991, pp.405-433.

[7] 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.

[8] 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.

[9] T. Clark, and T. Clark, *Designing Storage Area Networks*, Addison-Wesley Pub., USA, 1999.

[10] S. J. Eggers, and T. E. Jeremiassen, "Eliminating False Sharing", *Proceedings of the 1991 International Conference on Parallel Processing*, August 1991, Vol.1, pp.377-381

[11] M. Farley, and T. A. May, *Building Storage Networks*, McGraw-Hill Professional Publishing, USA, 2000.

[12] G. Gibson*; Redundant Disk Arrays: Reliable, Parallel Secondary Storage*, MIT Press, 1992.

[13] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. M. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg and J. Zelenka, "File Server Scaling with Network-Attached Secure Disks", *Proceedings of the ACM Sigmetrics'97*, June 1997.

[14] 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.

[15] J. H. Hartman, I. Murdock, and T. Spalink. "The Swarm Scalable Storage System", *Proceedings of the 19th IEEE International Conference on Distributed Computing*

Systems (ICDCS '99), June 1999.

[16] K. Hwang, H. Jin, and R. Ho, "RAID-x: A New Distributed Disk Array for I/O-Centric Cluster Computing", *Proceedings of 9th IEEE International Symposium on High Performance Distributed Computing (HPDC-9)*, August 1-4, 2000, Pittsburgh, Pennsylvania, USA, pp.279-286.

[17] K. Hwang, H. Jin, E. Chow, C.-L. Wang, Z. Xu, "Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space", *IEEE Concurrency*, 7(1): 60-69, 1999.

[18] K. Hwang, Z. Xu*; Scalable Parallel Computing: Technology, Architecture, programming*, WCB/McGraw-Hill Co., 1998.

[19] R. L. Hyde and B. D. Fleisch, "Degenerate Sharing", *Proceedings of the 1994 International Conference on Parallel Processing*, 1994, Vol.1, pp.267-270.

[20] H. Jin, J. He, Q. Chen, and K. Hwang, "Grouped RAID Accesses to Reduce False Sharing Effect in Clusters with Single I/O Space", *Proceedings of Second International Symposium on High Performance Computing*, Lecture Notes in Computer Science, Vol.1615, Springer-Verlag, 1999.

[21] H. Jin, and K. Hwang, "False Sharing Problems in Distributed RAIDs", *Proceedings of 1999 ACM Symposium on Applied Computing*, San Antonio, Texas, Feb. 1999, pp.461-465.

[22] H. Jin, and K. Hwang, "Performance Effect Analysis of False Sharing Problem in Clusters with Single I/O Space", *Proc. of 1999 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications*, Vol.VI, Las Vegas, Nevada, July 1999, pp.2788-2794.

[23] H. Jin, X. Zhou, and P. Cheng, "Track-based Improved LRU Algorithm for Disk Array", *Proceedings of 15th IMACS World Congress on Scientific Computation Modeling and Applied Mathematics*, 1997, Berlin, Germany, Vol. IV, pp.583-588.

[24] R. H. Katz, "Network-attached storage systems", *Proceedings of Scalable High Performance Computing Conference*, April 1992, pp.68-75.

[25] V. Khera, R. P. Larowe, and C. S. Ellis, "An Architecture-Independent Analysis of False Sharing", *Computer Science Department, Duke University, TR 93-006*, Oct. 1993

[26] V. Khera, "Factors Affecting False Sharing on Page-Granularity Cache-Coherent Shared-Memory Multiprocessors", *Ph.D. Dissertation, Duke University, CS-1994-37*, Dec. 1994.

[27] E. Lee, R. Katz, "The Performance of Parity Placement in Disk Arrays", *IEEE Transactions on Computers*, Vol.C-42, No.6, 1993, pp.651-664.

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

[29] K. Li, "IVY: A Shared Virtual Memory System for Parallel Computing", *Proceedings of 1988 International Conference on Parallel Processing*, Vol. II, 1988, pp.94-101.

[30] D. A. Menascé, O. I. Pentakalos, Y. Yesha, "An Analytic Model of Hierarchical Mass Storage Systems with Network-Attached Storage Devices", *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*, 1996, pp.180-189.

[31] S. Nakamura, H. Minemura, T. Yamaguchi, H. Shimizu, T. Watanabe and T. Mizuno, "Distributed RAID Style Video Server", *IEICE Transactions on Communication*, E79-B(8): 1030-1037, August 1996.

[32] 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.

[33] Quantum Co.: Storage Basics. http://www.quantum.com/src/storage_basics/

[34] E. Riedel, "Active Disks: Remote Execution for Network-Attached Storage", *Ph.D. Dissertation, CMU-CS-99-177*, November 1999.

[35] E. Riedel, and G. A. Gibson, "Understanding Customer Dissatisfaction With Underutilized Distributed File Servers", *Proceedings of the Fifth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies*, Sept. 1996.

[36] E. Riedel, G. A. Gibson and C. Faloutsos, "Active Storage For Large-scale Data Mining and Multimedia", *Proceedings of the 24th International Conference on Very large Databases*, August 1998.

[37] M. Stonebraker and G. A. Schloss, "Distributed RAID – a New Multiple Copy Algorithm", *Proceedings of the Sixth International Conference on Data Engineering*, Feb. 1990, pp.430-437.

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

[39] J. M. Toigo, and M. R. Toigo. *The Holy Grail of Data Storage Management.* Prentice Hall PTR, 1999.

[40] J. Torrellas, M. S. Lam and J. L. Hennessy, "False Sharing and Spatial Locality in Multiprocessor Caches", *IEEE Transactions on Computers*, C-43(6): 651-663, June 1994.

Biographical Sketches

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. In 1996, he was awarded the scholarship by German Academic Exchange Service (DAAD) at Technical University of Chemnitz in Germany. He received International Who's Who for Professional in 2000. He has worked at the University of Hong Kong, where he participated in the HKU Cluster project. Presently, he works as a visiting scholar at the Internet and Cluster

Computing Laboratory at the University of Southern California.

Dr. Jin is a member of IEEE and ACM. He served as workshop chair of *2001 International Workshop on Cluster Infrastructure on Web Server and E-Commerce Applications (CISCA'01), First International Workshop on Internet Computing and E-Commerce*, and *International Workshop on Cluster Computing Technologies, Environments, and Applications (CC-TEA'00)*. He served as program committee chair of *Asia-Pacific International Symposium on Cluster Computing (APSCC'2000)* and program committee vice-chair of *2001 International Symposium on Cluster Computing and Grid (CCGrid'01)*. 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.

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

Dr. Hwang has published six books and over 160 technical papers in computer science and engineering, lectured worldwide on computer and information technology, and performed consulting and advisory work for IBM, MIT Lincoln Lab., ETL in Japan, CERN Computing School in the Netherlands, ITRI in Taiwan, and GMD in Germany. His current research interest lies in network-based PC or workstation clusters and the middleware support for security, availability, and single-system-image services. He can be reached by Email: kaihwang@usc.edu.