A Performance Evaluation of RAID Architectures

Shenze Chen and Don Towsley

CMPSCI Technical Report 92-67
Replaces COINS TR91-71
September 1992

A Performance Evaluation of RAID Architectures[*]

Shenze Chen *Don Towsley*

Department of Computer Science
University of Massachusetts
Amherst, MA 01003

Abstract

In today's computer systems, the disk I/O subsystem is often identified as the major bottleneck to system performance. One proposed solution is the so-called *redundant array of inexpensive disks (RAID)*. In this paper, we examine the performance of two of the most promising RAID architectures, the *mirrored array* and the *rotated parity array*. First, we propose several scheduling policies for the mirrored array and a new data layout, *group-rotate declustering*, and compare their performance with each other and other data layout schemes. We observe that group-rotate declustering, coupled with a policy that routes reads to the disk with the smallest number of requests, provides the best performance, especially for workloads that generate large I/O requests. Second, through a combination of simulation and analysis, we compare the performance of this mirrored array architecture to the rotated parity array architecture. This latter study shows that, *i)* given the same storage capacity (approximately double the number of disks), the mirrored array considerably outperforms the rotated parity array and, *ii)* given the same number of disks, the mirrored array still outperforms the rotated parity array in most cases, even for applications where I/O requests are for large amounts of data. The only exception occurs when the I/O size is very large; most of the requests are writes, and most of these writes perform *full strip write* operations.

Index terms — Disk arrays, I/O subsystems, performance analysis, queueing model, RAID, scheduling policies.

[*]This work is supported, in part, by the Office of Naval Research under contract N00014-87-K-0796, by NSF under contract IRI-8908693, and by an NSF equipment grant CERDCR 8500332.

1 Introduction

In many computing systems, the disk I/O subsystem is quite often identified as the major bottleneck to system performance. During the past several years, CPU speed has increased at a rate of 40% to 100% per year [1, 14], whereas disk seek times have only improved by 7% per year [12, 32]. This has led to a big gap between the speed of the CPU/main memory and that of the disk I/O subsystem [30] which is expected to increase further in the near future. Based on these observations, we predict that further increases in the processor speed will bring little gain to the overall system performance. This presents the following fundamental problem: based on the current foreseeable future disk technology, how does one reduce the speed gap between the CPU main memory and the disk I/O subsystem.

One attractive idea is the so-called *disk array*, where the disk I/O subsystem consists of multiple disks with data spread over these disks. The ideas of *disk interleaving* and *disk striping* were first introduced by Kim [17] and Salem et al [40], respectively. Since then, a great deal of work has focused on various design issues related to the performance of disk arrays [30, 29, 18, 21, 36, 4, 44] and to their reliability [42, 9, 27]. Disk array architectures fall into one of five different classes proposed in [35, 34, 15] referred to as *Redundant Arrays of Inexpensive Disks (RAID)*. Among the five, the two most promising candidates for high performance computing systems appear to be the *mirrored disk array (RAID 1)* and the *rotated parity array (RAID 5)*.

In this paper we propose several scheduling policies suitable for RAID 1 and its variants. We compare the performance of these policies in the case that all disks are operational (normal mode). We also propose a scheduling policy for RAID 5 that solves the *write synchronization problem* inherent in that architecture. Last, the performances of RAID 1 and RAID 5 are compared to each other. All of these performance evaluations are conducted for two types of applications: *i)* applications in which I/O requests are for small amounts of data (e.g., transaction processing, workstation), and *ii)* applications in which I/O requests are for large amounts of data (e.g., supercomputing, image processing). The main results of this study are:

- in the normal mode of operation, a newly-proposed *group-rotate declustering* architecture, coupled with a policy that assigns read requests to the disk containing the data with the shortest queue, provides the lowest mean response time of all of the RAID 1 variants that we consider. This is true for both types of applications described above.

- the above mentioned RAID 1 architecture significantly outperforms RAID 5 when appplications generate I/O requests for small amounts of data. This is true for the case that both architectures have the same number of disks as well as when they have the same storage capacity. In the case of applications that generate I/O requests for large amounts of data, the results are not as clear. RAID 5 performs better when most requests are very large, most requests are writes, and most writes perform *full stripe writes.*

The above studies are performed through a combination of simulation and analysis. For example, the RAID 5 architecture is approximately analyzed via decomposition. Each disk is modeled as a priority queueing system. This and other analyses are of independent interest.

Other related works on performance evaluation of disk arrays can be found in [22, 37, 31]. However, in these works, reads and writes are treated in the same way when the performance of RAID 5 is examined. Therefore, the high cost suffered by *partial stripe writes* under RAID 5 is ignored.

I/O performance can also be improved by introducing a disk cache [41, 28]. The effectiveness of a disk cache depends on the I/O access pattern as well as the cache size. For applications where disk accesses show a high locality, disk caching may satisfy most of the read requests and therefore reduce the I/O traffic to the disk. For transaction processing, however, disk caching may be less effective, because I/O requests randomly access the disks and it is impractical to cache the entire database. In any case, disk caches can be combined with disk arrays and the results of our study remain valid for such systems as well.

The remainder of this paper is organized as follows. Section 2 describes different RAID architectures and the basic disk model. Section 3 proposes several scheduling policies suitable for different disk array architectures. Section 4 includes the discussions of disk arrays supporting transaction processing and workstation environments. The performance of disk arrays for scientific computing and image processing systems is addressed in Section 5. The model validations are discussed in Section 6. Last, Section 7 summarizes this paper.

2 Disk Array Architectures

In this section, we briefly describe several disk array architectures of most interest to us, RAID 1 and its variants, and RAID 5.

We consider disk arrays consisting of N identical disks. In the architectures of interest to us, data is *block interleaved* across W disks, where W is the *stripe width*, $W \leq N$. For example, a file f is logically divided into blocks $f_1, f_2, ..., f_n$, which are stored on contiguous disks in the stripe (n might be larger than the stripe width W). If a request accesses exactly W blocks of data in a stripe (i.e., aligned with the stripe boundaries), it is called a *full stripe I/O* request. Otherwise, it is called a *partial stripe I/O* request[1]. The main advantage of block interleaving is that it supports the concurrent execution of multiple small I/O requests. Other alternatives are *bit* or *byte interleaving*, as exemplified by RAID 2 and RAID 3, respectively. However, both of these require that all disks in the array be involved in servicing each I/O request; and, therefore, at most, one I/O request can be executed at a time. The most significant benefit achievable from these architectures is the fast transfer rate. While these architectures may be suitable for applications in which each request transfers a large amount of data, they are not appropriate for applications such as transaction processing and workstation environments. In this case, since each request typically transfers a small amount of data, the advantage of a fast transfer rate diminishes. On the other hand, since all disks are needed to serve a request, the disk array cannot support multiple small I/O's concurrently. A common advantage of interleaving data across stripes in an array is the load sharing of heavily used files among multiple disks, given that the unit of interleaving (or striping unit) is not too coarse.

2.1 RAID 1 and Its Variants

In this subsection, we describe the RAID 1 architecture, also known as the *mirrored array*. We consider several variations that differ from each other according to how the data is placed on the disks. These include *mirrored declustering* [34, 9], *chained declustering* [13], and a new variant, *group-rotate declustering*, which combines the advantages of both mirrored and chained declustering.

Mirrored Declustering: In mirrored declustering, the N disks are configured as $N/2$ pairs of mirrored disks, with the i-th pair termed *mirror i*. Two copies of data are striped over these $N/2$ pairs. Each pair contains the same data, which is not necessarily stored at the same location on the two disks. However, contiguous logical blocks of both copies are allocated sequentially on the two $N/2$–striped disk pairs. Figure 1 shows a scenario where file f, consisting of 6 blocks, is allocated on an array of 8 disks. In the figure, we use F_i to denote the first copy and f_i to denote the second copy of the i-th file block, $i = 1, 2, \cdots$. A read request can be satisfied by either copy, but a write request requires an update of both

[1]If a request accesses multiple blocks of data which cover several stripes plus a partial stripe, we still consider it to be a "partial stripe I/O".

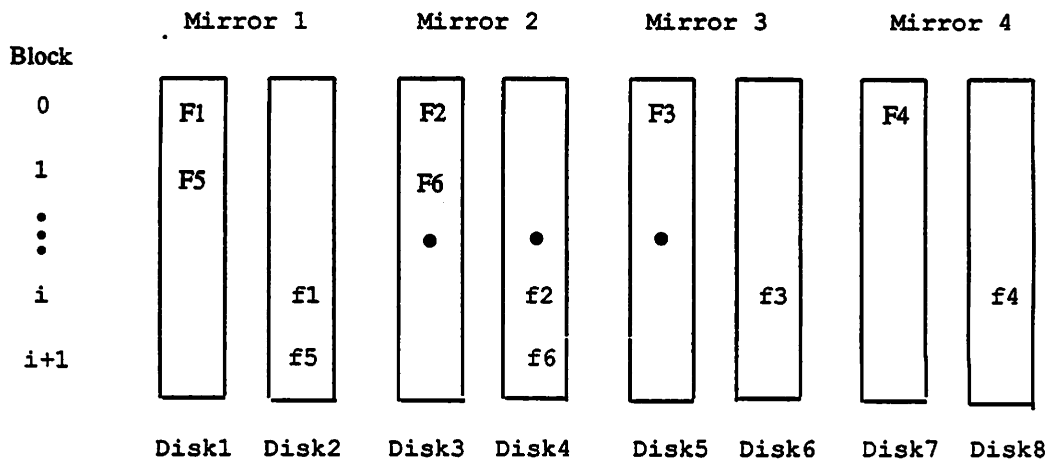


Figure 1: The RAID 1 (Mirrored Declustering) Architecture ($N = 8$).

copies. One of the drawbacks of this architecture is that when a disk fails and the disk array operates in a *failure* or *rebuild* mode, all of the traffic originally directed to the failed disk is now redirected to its mirror, which may become a bottleneck.

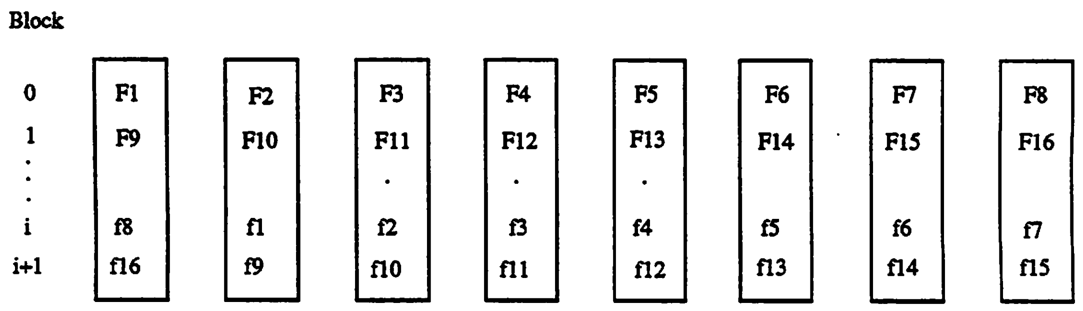


Figure 2: The Chained Declustering Architecture ($N = 8$).

Chained Declustering: To overcome the above bottleneck during recovery period, Hsiao and DeWitt [13] proposed the *chained declustering* architecture, where two physical copies of data, termed the *primary copy* and the *backup copy*, are maintained. As shown in Figure 2, if a primary data block is allocated on disk i, its duplicate backup block is allocated on disk $(i \bmod N) + 1$. In the original description, [13], a read is only allowed to access the primary copy during normal mode. We will improve its performance by allowing a read to be executed on either copy.

There are several variants on this architecture according to how the two data copies are

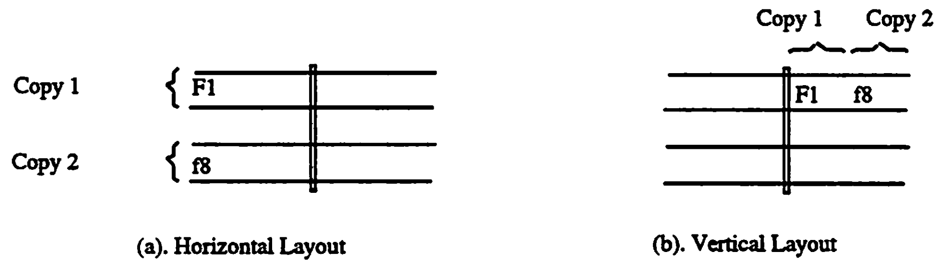


Figure 3: Data Layout Approaches for Chained Declustering.

placed on the disks. We study two of these data placements in this paper, as shown in Figure 3. Typically, a disk spindle consists of a collection of platters. Each platter contains a number of tracks. A cylinder consists of tracks at the same position on each platter. For workstation disks, usually read/write heads are attached to an actuator which positions the heads to an appropriate track, and only one head transfers data at a time. The two approaches are described as follows:

- **Horizontal Layout (HL)**: By using this approach, half of the platters on each spindle is used to store the first copy, and the remaining platters are used to store the second copy;
- **Vertical Layout (VL)**: By using this approach, the first copy occupies half of the cylinders, and the second copy occupies the other half.

For example, consider a disk spindle consisting of 4 platters labeled 0 to 3 and 1000 cylinders. The *horizontal* approach will allocate copy 1 on platter 0 and 1, occupying all tracks on these two platters, and copy 2 on platter 2 and 3. On the other hand, the *vertical* approach will allocate copy 1 on cylinder 0 to 499, including all the tracks in each cylinder, and copy 2 on cylinder 500 to 999 (we assume that each track has the same capacity). As we will see later in Section 5.1, the HL approach is favored for writes, whereas the VL approach is favored for reads in large I/O environments.

Notice that a main difference between mirrored declustering and chained declustering is that when a read request needs to access a large amount of data, mirrored declustering can support two such "large" reads concurrently, whereas chained declustering can only support one at a time. However, when a disk fails, chained declustering can evenly distribute the burden of the failed disk to all of the remaining $N-1$ operational disks.

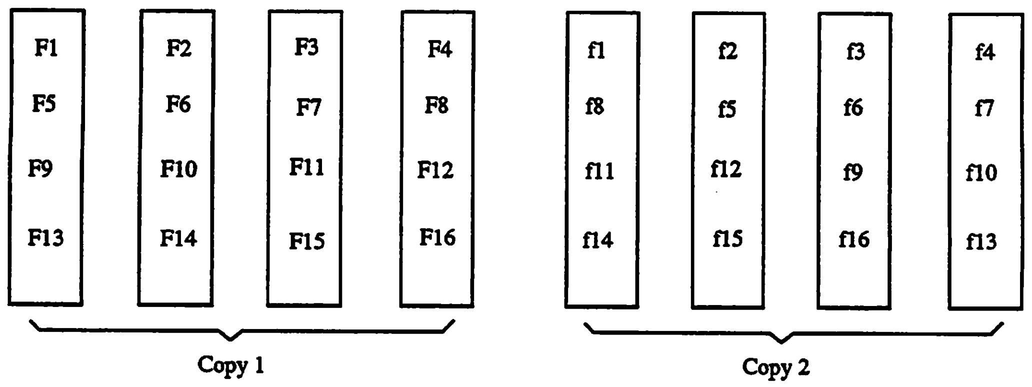


Figure 4: The Group-Rotate Declustering Architecture ($N = 8$).

Group-Rotate Declustering: By taking advantage of both mirrored and chained declustering, we propose a third architecture, called *group-rotate declustering*, as shown in Figure 4. In this architecture, the first copy is stored in the same way as mirrored declustering, but the second copy is rotated among the remaining $N/2$ disks. Clearly, group-rotate declustering can still support two "large" reads concurrently as mirrored declustering above, but during failure it distributes the burden of the failed disk evenly among $N/2$ disks.

There is another architecture proposed in the literature, called *interleaved declustering* [45]. Since this architecture is not appropriate for applications in which I/O requests need to update a large amount of data, we exclude it from our discussions.

2.2 The RAID 5 Architecture

Unlike RAID 1 and its variants which duplicate each data item, the RAID 5 architecture provides parity information to achieve reliable service. This is attractive because only one block of redundant information is required for every $N - 1$ data blocks. This is in contrast to RAID 1 and its variants which double the number of disks.

The RAID 5 architecture is shown in Figure 5. For reliability purposes, each stripe contains a parity block, which is the XOR of all of the other data blocks in the same stripe. If any one disk fails, the array is still operational, since the failed data can be restored by XORing data from all of the other disks. The price paid for this reliability is that each write operation is required to update the parity block(s) as well as the data block(s). This creates

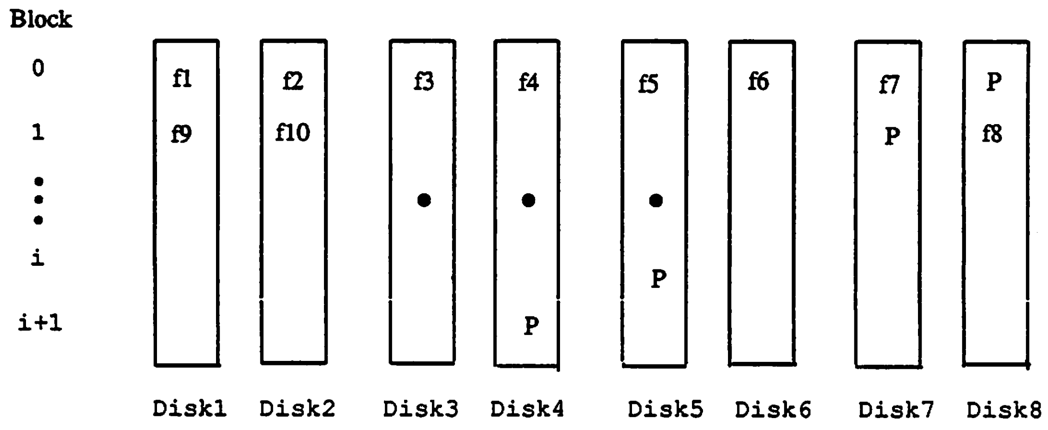


Figure 5: The RAID 5 Architecture ($N = 8$).

additional workload to the array. If a write request desires to update a partial stripe of data, which we refer to as a "partial stripe write", the new parity block for that stripe is calculated by

$$new\_parity = new\_data \oplus old\_data \oplus old\_parity \tag{1}$$

where $X \oplus Y$ corresponds to the XOR of X and Y. Thus, prior to updating the new data and parity, the old data and parity have to be read out first in order to calculate the new parity. This is referred to as the *read-update procedure* required by a write request. On the other hand, if a request updates a complete stripe of data (full stripe write), the new parity block for that stripe can be calculated by XORing all of the new data blocks. Thus it is not necessary in this case to read the old data and old parity blocks. In order to avoid the parity update operation from becoming a bottleneck, parity blocks are rotated among the N disks in RAID 5. Numerous parity placement strategies have been studied in [21].

Notice that for a *partial stripe write*, the parity update operation cannot proceed before the corresponding data block(s) has (have) been read out, since the calculation of a new parity block is based on the old data information (see Equation (1)). We call this the *write synchronization problem*, which will be addressed further in Section 3.2.

Another way to calculate the new parity is to read out those blocks not being updated in a stripe and XOR them with the new data blocks. This approach is beneficial only when most of the requests access more than half of the disks in a stripe, since it creates extra workloads to those disks not being updated. Simulation results (not reported here) show that when the request size (in term of number of blocks) is uniformly distributed within a stripe, this approach performs worse than reading from the disks to be updated.

2.3 A Disk Model

We present the disk model which will be used in subsequent discussions. Since we are focusing on the disk I/O subsystem, we count the I/O response time as the time elapsed between the arrival of a request to the subsystem and the departure of this request from the subsystem. Typically, the I/O response time to a disk subsystem consists of four components: queueing delay at the controller, seek time, rotational latency, and data transfer time. We assume that each device has its own independent data path. Thus, we only focus on the queueing delay, which is a function of the I/O load, and the disk access time, which consists of seek, rotation, and transfer times.

Define,

C: the number of cylinders for each disk;

N_b: the number of blocks in each track;

τ: the transfer time for a single block;

S_{max}: the maximum seek time;

R_{max}: the full rotation time;

D: a random variable ($r.v.$) denoting the disk seek distance;

V: a $r.v.$ denoting whether an access is sequential ($V = 0$) or not ($V = 1$);

p_s: the probability that the seek distance is equal to zero, $p_s = P\{V = 0\}$;

S: a $r.v.$ for disk seek time;

R: a $r.v.$ for rotational latency;

X: a $r.v.$ for the sum of seek and rotation time, $X = S + R$;

B: a $r.v.$ for the number of blocks to be transferred in a request;

T: the data transfer time, $T = \tau B$;

Y_r, Y_w, Y: $r.v.$s for read, write, and overall disk service times.

Z_r, Z_w, Z: *r.v.*'s for read, write, and overall I/O response times.

First, consider the disk seek pattern. Since it is observed that in reality, most of the time the disk arm doesn't move [23, 18], we introduce a *sequential access probability*, p_s, which is defined to be the probability that the seek distance is equal to zero. We identify two kinds of access patterns, the sequential access pattern and non-sequential access pattern. Under the *sequential access* pattern, the disk arm does not move; whereas under the *non-sequential access* pattern, the arm has to move to serve a request. In the case of a *non-sequential* access, the arm is assumed to move to any other cylinder with equal probability.

According to the definition of V, we have the following conditional distributions for the seek distance,

$$\begin{aligned} P\{D=i|V=0\} &= \begin{cases} 1 & i=0, \\ 0 & i=1,2,\cdots,C-1; \end{cases} \\ P\{D=i|V=1\} &= \begin{cases} 0 & i=0, \\ \frac{2(C-i)}{C(C-1)} & i=1,2,\cdots,C-1, \end{cases} \end{aligned}$$

where the term $2(C-i)/C(C-1)$ is derived based on the assumption that the arm and the target cylinder are randomly located on the disk and that they are distinct.

Removal of the conditioning yields

$$P\{D=i\} = \begin{cases} p_s, & i=0, \\ (1-p_s)\frac{2(C-i)}{C(C-1)}, & i=1,2,\cdots,C-1. \end{cases} \tag{2}$$

We also use the following expression for the seek time, S, as a function of the seek distance D [43, 2, 3, 31],

$$S = \begin{cases} 0, & D=0, \\ a+b\sqrt{D}, & D>0, \end{cases}$$

where a is the arm acceleration time and b is the mechanical seek factor.

For ease of analysis, we approximate the seek distance D as a continuous *r.v.* with probability density function (*pdf*)

$$f_D(x) = \begin{cases} p_s u_0(x), & x=0, \\ (1-p_s)\frac{2(C-x)}{C^2}, & 0<x\le C, \end{cases}$$

where $u_0(x)$ is the unit impulse function (see [19] pp. 342). The cumulative distribution function (CDF) of S, $F_S(x) = P\{D < x\}$, is (see [7] for details),

$$F_S(x) = \begin{cases} F_D(0), & 0\le x\le a, \\ F_D\left((\frac{x-a}{b})^2\right), & a<x\le S_{max}, \\ 1, & S_{max}<x, \end{cases} \tag{3}$$

where the maximum seek time $S_{max} = a + b\sqrt{C-1}$. The mean seek time can be accurately approximated by (see [7] for details),

$$E[S] \approx (1-p_s)\left[a + b\sqrt{C-1}\frac{8}{15}\right]. \tag{4}$$

The rotation time is assumed to be uniformly distributed in $[0, R_{max}]$ with *pdf*

$$f_R(x) = 1/R_{max} \qquad\qquad 0 \le x \le R_{max}. \tag{5}$$

Let $X = S + R$ be the sum of seek and rotation time, then the *pdf* of X is

$$f_X(x) = \int_0^{S_{max}} f_R(x-s)\, dF_S(s).$$

For the parameters of interest to us (see end of this section), $R_{max} \le b\sqrt{C-1}$ and $f_X(x)$ takes the following form,

$$f_X(x) = \begin{cases} f_R(x)p_s + \int_{a^+}^{x} f_R(x-s)\, dF_S(x), & 0 \le x \le R_{max}, \\ \int_{a^+}^{x} f_R(x-s)\, dF_S(s), & R_{max} < x \le R_{max} + a, \\ \int_{x-R_{max}}^{x} f_R(x-s)\, dF_S(s), & R_{max} + a < x \le S_{max}, \\ \int_{x-R_{max}}^{S_{max}} f_R(x-s)\, dF_S(s), & S_{max} < x \le S_{max} + R_{max}; \end{cases} \tag{6}$$

The disk service time for a typical request is

$$Y = X + T \tag{7}$$

with mean

$$\begin{aligned} E[Y] &= E[S] + E[R] + E[T], \\ &= E[X] + E[T] \end{aligned} \tag{8}$$

where T is the transfer time, $T = \tau B$.

Last, the disk parameters used in our study are summarized as follows which are typical under today's technology: number of cylinders $C = 1200$; transfer rate= 3MB/sec; maximum rotation time $R_{max} = 16.7$ ms; block size = 4096 bytes; number of blocks per track $N_b = 12$; acceleration time $a = 3$ ms; seek factor $b = 0.5$ [31]. Thus, the time required to transfer a single block is $\tau = 1.3$ ms.

3 Scheduling Policies

Scheduling policies suitable for different disk array architectures are described in this section.

3.1 Policies for RAID 1 and its Variants

Since there are two data copies stored in the array, a read request can be satisfied by either copy, whereas a write request must be performed on both copies. In the following, we will focus on three policies. Discussions of other policies can be found in [6]. For all three policies to be described below, two queues are maintained by the system, one for each copy (Figure 6). Each queue is served in a first-come-first-serve manner. At the time of arrival, a write request generates two tasks that enter each of the two queues. The various policies differ from each other according to the way that they decide upon the disk to which a read request will be assigned.

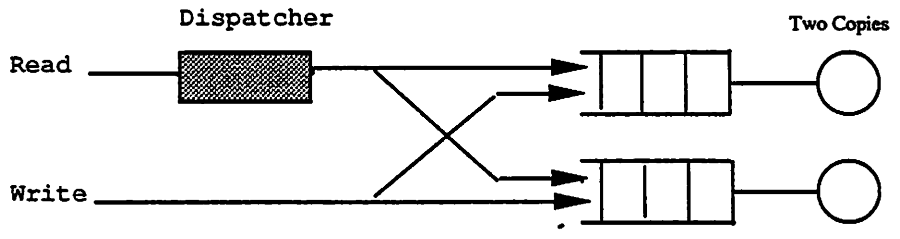


Figure 6: A Queueing Model for RAID 1 Policies.

- **The Random Join (RJ) Policy**

  Under this policy, a read request is randomly assigned to the queue associated with disk i with probability α_i, $i = 1, 2$ ($\alpha_1 + \alpha_2 = 1$), at the time of its arrival.

- **The Shortest Queue (SQ) Policy**

  Under the SQ policy, a read request selects the disk with the shortest queue at the time of its arrival. Ties are broken in an arbitrary manner. When a read request arrives to an empty system, i.e., both disks are idle, it is assigned to the disk whose arm is closest to the cylinder containing the desired block.

- **The Minimum Seek (MS) Policy**

  At the time of arrival of a read request, the MS policy checks the cylinder address of the last request in each queue and directs the read to the queue with the minimum seek distance. When each queue is served in a FCFS manner, this policy reduces seek times for read requests.

Remark: Among all of these policies, the RJ policy is suitable for systems in which the two disks containing the two target data copies are physically located at two distant sites, since it requires no information interchange between the two disks. The other policies, however, are more suitable for systems in which all disks in the array are located close to each other because those policies have to know the status of each queue in order to schedule read requests.

3.2 A Synchronized Policy for RAID 5

In this subsection, we propose a synchronized I/O scheduling policy, the *after read-out (AR)* policy, for RAID 5 that is suitable for applications in which requests are made for small amounts of data. This policy accounts for the *write synchronization problem* described in Section 2.2. In all cases, a write request completes only when both the data and the parity have been updated.

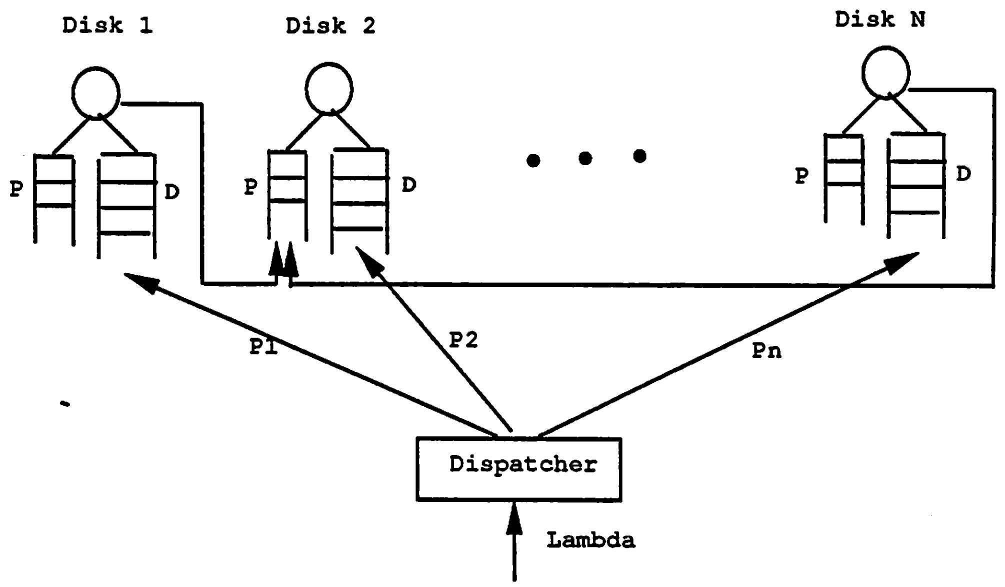


Figure 7: A Queueing Model for the AR Policy.

We first describe the AR policy for the case that a write request desires to update a single block. Under ths policy, two queues are maintained for each disk in the array, one for arriving read and write data requests (*D queue*) and the other for parity update requests (*P queue*), as illustrated in Figure 7. When a read or write request arrives at the disk subsystem, the dispatcher sends it to the D queue of the target disk.

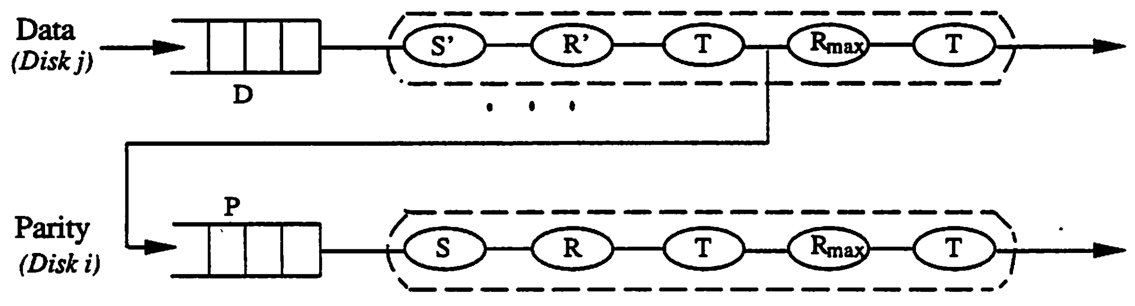


Figure 8: Scenario for Serving a Write Request.

To serve a write request, both the data and parity update requests will experience the following 5 service stages: seek, rotation, read (for the calculation of the new parity), full rotation, and write. Figure 8 shows a scenario for serving a write request. When it reaches the head of the queue and is scheduled for service, it generates a parity update request to the P queue of the corresponding disk containing its parity block(s) after the old data has been read out. Thus, we call it the *after read-out* policy. The requests in the P queue are given higher priority than requests in the D queue to ensure that an outstanding parity request begins service as soon as possible, since its corresponding data update operation has already started. The disk operation is assumed to be non-preemptive; therefore, a new parity request cannot commence service until the current I/O finishes.

If a write request desires to modify more than one block on a stripe, then the parity request is generated when the *last* old data block has been read out.

Remark: Another policy, called the BS policy, was proposed in [7]. This policy generates a parity update request as soon as a write begins its service. We have observed little difference in the performances of theses two policies (the BS policy is slightly better at low loads). However, the AR policy is simpler to implement, especially in a distributed system where the disks containing the data and parity blocks may be located at different sites.

4 Disk Arrays for Transaction Processing and Workstation Environments

In this section, we study disk arrays for applications such as transaction processing and workstation environments. We present analytic and simulation results for disk array architectures coupled with different scheduling policies. Since it has been observed in the literature that for applications such as transaction processing systems and workstation/engineering environments, I/O requests typically access a small amount of data [25, 11, 26, 33], we assume that each request accesses a single block of data (4096 bytes [31]) for ease of discussion. Because data is interleaved among multiple disks, we further assume that a request is routed to any disk with equal probability.

4.1 Variations of the RAID 1 Architecture

In this subsection, we study and compare RAID 1 and its variants, *mirrored declustering, chained declustering*, and *group-rotate declustering*, coupled with the RJ, SQ, and MS policies. The performance results are obtained from simulation experiments on a disk array of 16 disks. In [18], Kim reported several real disk reference traces, which show the percentage of sequential accesses ranging from 25% to 50%. For disk arrays, however, since data is spread over multiple disks, the sequential access probability on each disk is expected to be smaller than that of conventional disk systems. Hence, we selected $p_s = 0.2$ in our studies. We also tested the case of $p_s = 0.5$, and observed similar behavior. These results are averaged over 40 runs. Each run includes 51,000 requests executed, with the first 1000 requests being considered to belong to the transient period, and therefore excluded from our statistics. Ninety-five percent confidence intervals are shown in the figures, and the widths of these confidence intervals are less than 3% of the estimated values

4.1.1 Disk Arrays Coupled with the RJ Policy

The performance of RAID 1 and its variants, coupled with the RJ policy, is illustrated in Figure 9 for the case of read probabilities $p_r = 0.75$ and $p_r = 0.25$. As expected, in small I/O environments where each request accesses one block of data, the three architectures basically perform the same. There is only a slight difference when the disk array is highly loaded and most of the requests are writes. In this case mirrored declustering provides slightly better performance than the other two architectures. Therefore, we conclude that the three variants under the RJ policy are equivalent in such "small" I/O environments.

4.1.2 Disk Arrays Coupled with the MS Policy

¿From Figure 10, we observe that, under the MS policy, the performance of different disk array architectures is also close to each other, especially when most of the requests are writes. When most of the requests are reads, the group-rotate architecture is observed to provide the best performance.

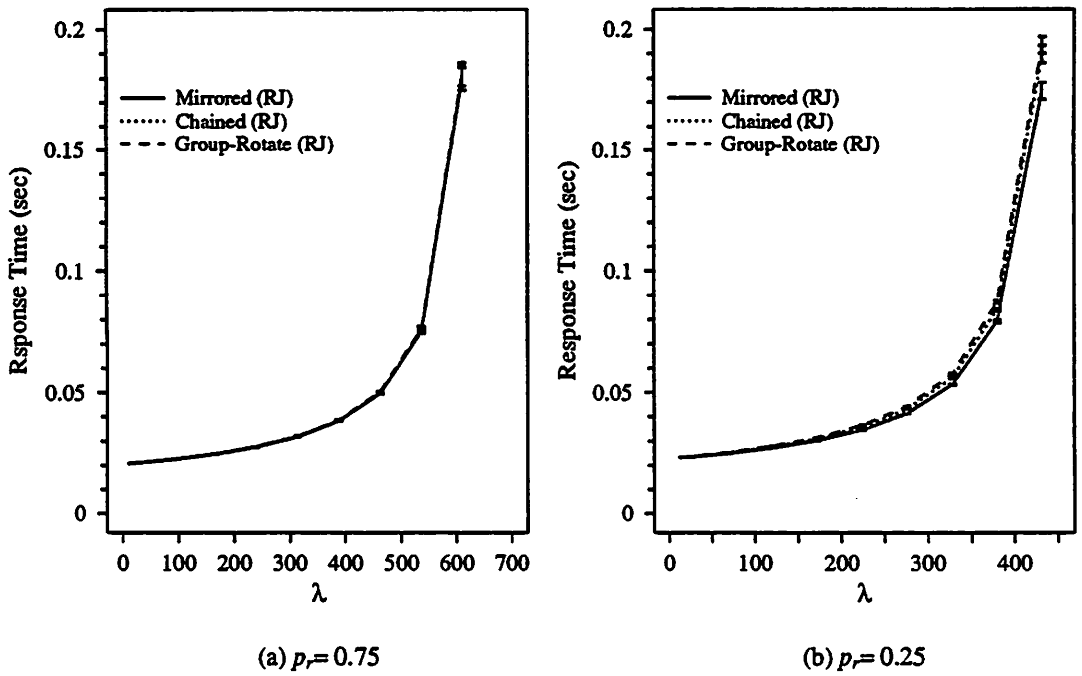


Figure 9: RAID 1 and Its Variants Coupled with RJ in Small I/O Environments.

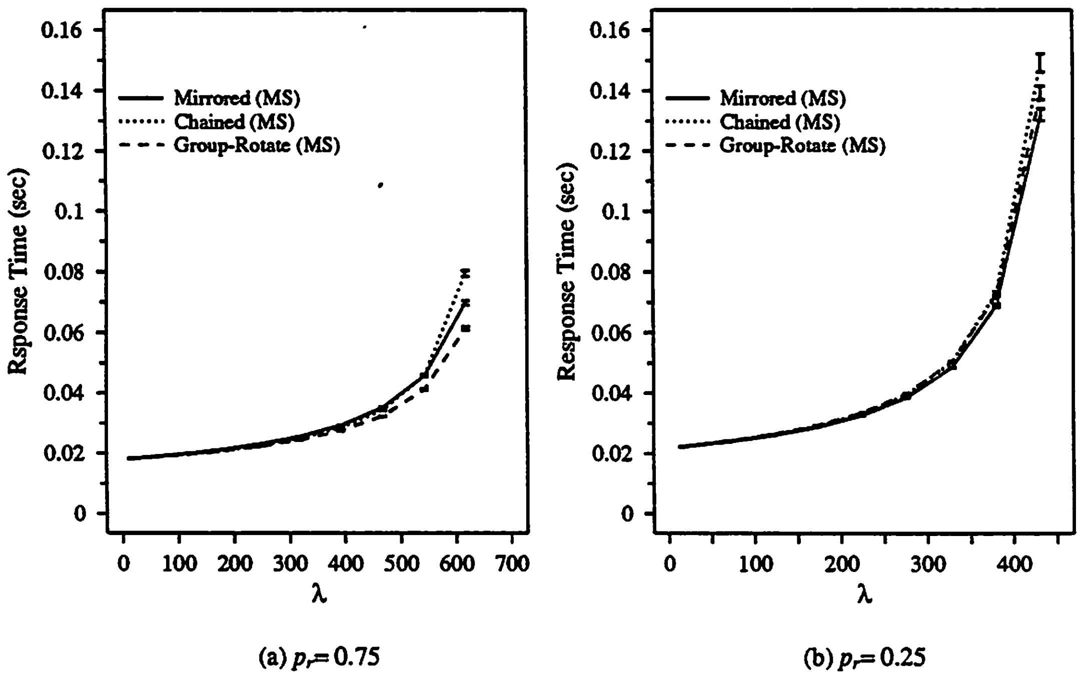
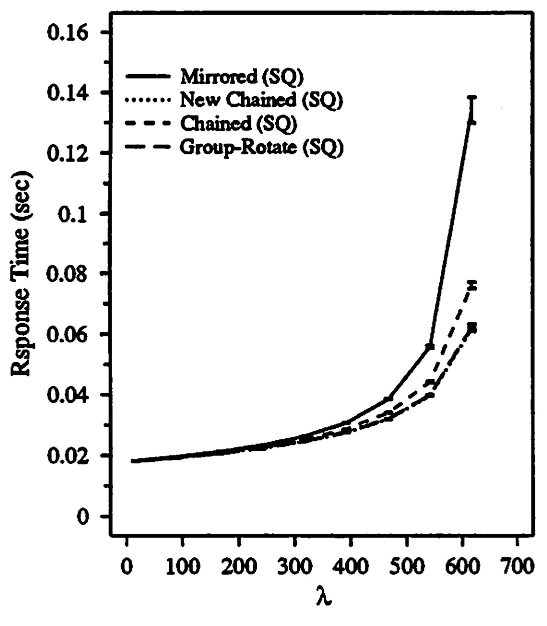
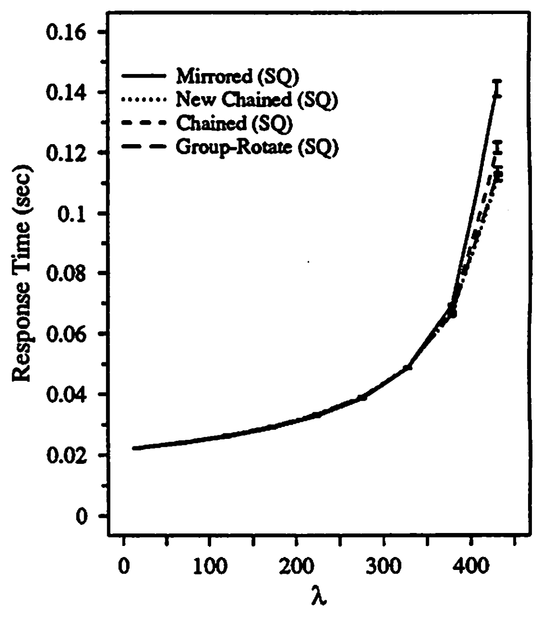


Figure 10: RAID 1 and Its Variants Coupled with MS in Small I/O Environments.



(a) p_r= 0.75



(b) p_r= 0.25

Figure 11: RAID 1 and Its Variants Coupled with SQ in Small I/O Environments.

4.1.3 Disk Arrays Coupled with the SQ Policy

When the shortest queue policy SQ is used, however, the three variants exhibit quite different performances, especially at high loads (Figure 11). We see that group-rotate declustering is the best, and mirrored declustering is the worst. The reason for this is that, under SQ, mirrored declustering only provides load balancing of reads between two queues, since each mirrored pair is independent of the other pairs. In group-rotate declustering, data on each disk is replicated over $N/2$ disks. Therefore, the shortest queue policy has the ability to balance load among $N/2$ queues. Chained declustering falls in between these two architectures.

Based on the above observations, we suggest a variant of chained declustering which has the potential of improving performance further. This variant, instead of placing the second copy of the blocks on disk D_i onto disk D_{i+1}, rotates them among the disks other than D_i in a similar way as the group-rotate declustering architecture. For example, consider disk 1 shown in Figure 2, on which F_1 and F_9 are allocated. Under this new variant, while f_1 remains on disk 2, f_9 is now placed on disk 3. This variant is expected to achieve a better load balance when the SQ policy is employed. In separate experiments, we have observed that this modified chained declustering architecture provides equivalent performance to the group-rotate declustering, with differences in mean response times on the order of tenths of milliseconds (Figure 11). Note that this declustering architecture shares all of the desirable features of the original chain declustering architecture in the presence of a failed disk.

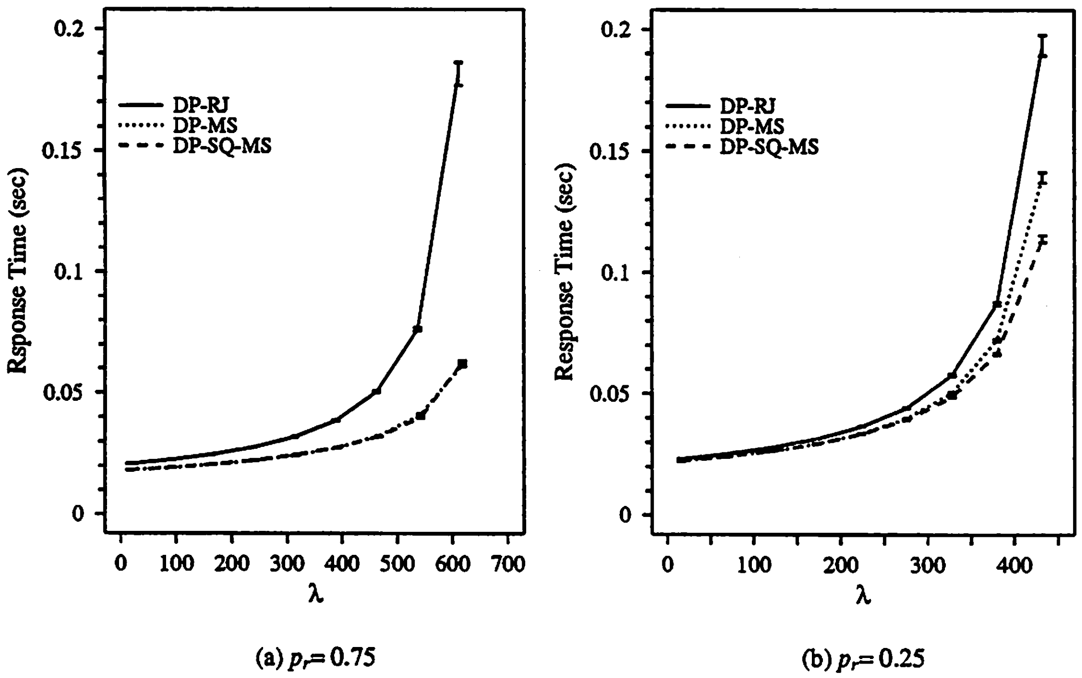


Figure 12: Different policies under Group-Rotate Declustering.

4.1.4 Different policies under Group-Rotate Declustering

We end this subsection by comparing different policies under the group-rotate declustering architecture. For the results shown in Figure 12, the disk utilization varies roughly from 0.02 to 0.93. From the figure, we observe that RJ is significantly worse than the other two. When most of the requests are reads, there is little difference between the MS and SQ policies. When most of the requests are writes, SQ outperforms MS. However, since the disk service times for read requests under MS are less that under SQ, the SQ system is expected to saturate at a lower load than the MS system.

4.2 Comparison of RAID 1 and RAID 5

The most significant advantage of RAID 5 over RAID 1 is its lower cost, since RAID 1 requires nearly double the number of disks in order to achieve the same capacity, or by using the same number of disks, RAID 1 only achieves half capacity as that of RAID 5. Given the high cost paid by RAID 1, we are interested in the performance gain achievable by RAID 1 over RAID 5.

In this subsection, we first present an accurate priority queueing model for RAID 5 coupled with the AR policy. Then, using this model, we compare the performance of RAID 5 to that of the group-rotate declustering variant of RAID 1 coupled with the SQ policy. Henceforth, whenever there is no confusion, we will use RAID 1 to refer to group-rotate declustering coupled with SQ in the remainder of this section. The validation of the analytic model will be addressed later in Section 6.

4.2.1 An Analysis of RAID 5 coupled with the AR Policy

We consider a disk array of N disks. The disk model has been described in Section 2.3. Based on that model, the disk service time for a read request is (Equation (7))

$$Y_r = X + T \tag{9}$$

and for a write request is

$$Y_w = X + T + R_{max} + T \tag{10}$$

where X is the sum of seek and rotational latency, T is the transfer time (since we assumed a single block access, $T = \tau$), and R_{max} is the full rotation time.

For ease of reference, we introduce the following additional notation in our analysis:

λ: the arrival rate to the disk I/O subsystem;

$\lambda_p(i), \lambda_d(i)$: the arrival rates for the high priority parity update requests and the low priority read, write requests to disk i, $i = 1, 2, \cdots, N$;

$Q_p(i), Q_d(i)$: *r.v.*'s for queueing times of high and low priority requests on disk i;

Y_p, Y_d: *r.v.*'s for service times of high and low priority requests;

In our model, I/O requests are assumed to arrive to the disk array according to a Poisson process with rate λ. As shown in Figure 7, two queues are maintained in front of each disk, one for read and write data requests (D queue), and the other for parity requests (P queue). When a request arrives at the subsystem, it is directed to disk i with probability p_i, $i = 1, 2, \cdots, N$. Therefore, the arrivals to the D queue of disk i are described by a Poisson process with parameter $\lambda_d(i) = p_i\lambda$, where a request is a read with probability p_r

and a write with probability $p_w = 1 - p_r$. Arrivals to the P queue of disk i are described by a superposition of $N-1$ parity request generating processes which direct requests to disk i with probability $1/(N-1)$. When N is large, it can be approximated by a Poisson process [16] with parameter $\lambda_p(i) = (1-p_i)p_w\lambda/(N-1)$. Strictly speaking, the $N-1$ parity generating processes are not independent of each other. However, when N is large, the correlations between these processes become small, and therefore we conjecture that they become independent in the limit as $N \to \infty$. In fact, when $N = 16$, our simulation results yield a close match to those of our analysis. Last, the FCFS discipline is used within both the D and P queues.

Thus, disk i ($i = 1, \cdots, N$) can be modeled as a two-priority class non-preemptive priority queue, where the service time for low priority customers (read and write requests in the D queue) is obtained by Equations (9) and (10). Define θ to be a *r.v.* associated with a request which takes value one if it is a read and zero otherwise. The service times for high and low priority requests are

$$\begin{aligned} Y_p &= Y_w, \\ Y_d &= \theta Y_r + (1-\theta)Y_w, \end{aligned}$$

where $P\{\theta = 1\} = p_r$ and $P\{\theta = 0\} = p_w$.

The first two moments for Y_p and Y_d are

$$\begin{aligned} \overline{Y_p} &= \overline{X} + 2\tau + R_{max}, \\ \overline{Y_p^2} &= \overline{X^2} + 2(2\tau + R_{max})\overline{X} + (2\tau + R_{max})^2, \\ \overline{Y_d} &= \overline{X} + \tau + p_w(\tau + R_{max}), \\ \overline{Y_d^2} &= p_r\left(\overline{X^2} + 2\tau\overline{X} + \tau^2\right) + p_w\left(\overline{X^2} + 2(2\tau + R_{max})\overline{X} + (2\tau + R_{max})^2\right), \end{aligned}$$

where

$$\begin{aligned} \overline{X} &= \overline{S} + \overline{R}, \\ \overline{X^2} &= \overline{S^2} + 2\overline{S}\,\overline{R} + \overline{R^2}, \end{aligned} \tag{11}$$

and the first two moments for S and R are derived from (3) and (5).

Based on results for non-preemptive priority queues, the mean queueing delays at the P and D queues on disk i, $i = 1, \cdots, N$, are ([20] pp. 121)

$$\overline{Q_p(i)} = \frac{\lambda_d(i)\overline{Y_d^2} + \lambda_p(i)\overline{Y_p^2}}{2[1 - \lambda_p(i)\overline{Y_p}]}, \tag{12}$$

$$\overline{Q_d(i)} = \frac{\lambda_d(i)\overline{Y_d^2} + \lambda_p(i)\overline{Y_p^2}}{2[1 - \lambda_p(i)\overline{Y_p}][1 - \lambda_d(i)\overline{Y_d} - \lambda_p(i)\overline{Y_p}]}. \tag{13}$$

The mean read response time on disk i is

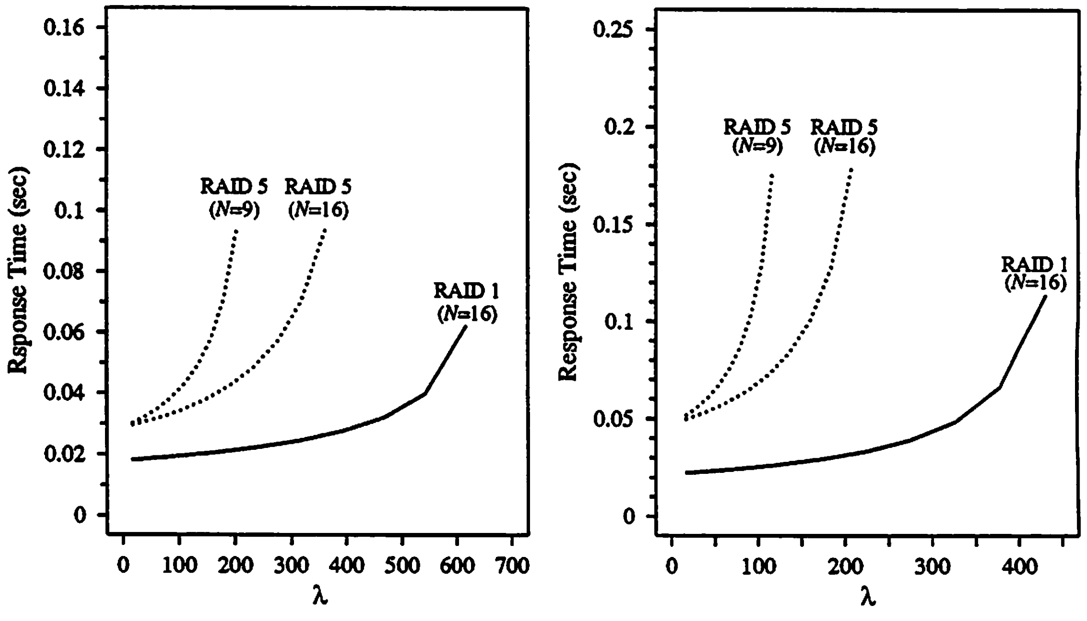
$$\overline{Z_r(i)} = \overline{Q_d(i)} + \overline{X} + \tau, \tag{14}$$

and the mean write response time is

$$\overline{Z_w(i)} = \overline{Q_d(i)} + 2\overline{X} + \frac{1}{N-1} \sum_{j=1;j\neq i}^{N} \overline{Q_p(j)} + 2\tau + R_{max}.$$

Finally, the mean response times are averaged over all disks

$$\begin{aligned} \overline{Z_r} &= \sum_{i=1}^{N} p_i \overline{Z_r(i)}, \\ \overline{Z_w} &= \sum_{i=1}^{N} p_i \overline{Z_w(i)}, \\ \overline{Z} &= p_r \overline{Z_r} + p_w \overline{Z_w}. \end{aligned} \tag{15}$$



(a) p_r=0.75 (b) p_r=0.25

Figure 13: Compare RAID 1 and RAID 5 in Small I/O Environments.

4.2.2 Performance Results

The results reported here for RAID 5 are obtained by setting $p_i = 1/N$, $i = 1, 2, \cdots, N$. Since in this study RAID 5 uses a small striping unit (4K bytes) and spreads data across multiple disks, it is reasonable to assume that requests go to each disk with equal probability.

The performance comparisons are conducted in two different ways. First, we let both RAID 1 and RAID 5 have the same capacity, i.e., the number of disks in RAID 1 is $2n$, and in RAID 5 is $n + 1$, where n is selected to be 8 in our experiment. Second, we compare the two architectures with the same number of disks, where RAID 1 loses half of its useful capacity. The results are shown in Figure 13, from which we observe that when both RAID 1 and RAID 5 have the same capacity, RAID 1 can support more than three times the arrival rate of I/O requests as RAID 5. When both arrays contain the same number of disks, RAID 1 can still support more than twice the number of I/O requests per second as compared to RAID 5. Therefore, we conclude that RAID 1 performs significantly better than RAID 5 in small I/O environments at the cost of doubling the number of disks or losing half of the capacity.

5 Disk Arrays for Supercomputing and Image Processing Systems

Applications such as supercomputing and image processing usually carry lower multiprogramming levels and, consequently, fewer I/O requests are issued per unit time. However, each I/O request typically accesses a large amount of data. Generally, computation parameters are moved in bulk from disks to memory resident data structures, and results are periodically written back to disks [15]. In this case, multiple disks work together as a single logical device providing a faster transfer rate. In this section, we assume that all of the disks that work together to serve a request are synchronized in terms of rotation and arm movement [17]. For example, for the mirrored declustering and group-rotate declustering architectures, disks are divided into two groups, one for each data copy. Thus each group is considered to be synchronized, but the two groups may work independently. For chained declustering and RAID 5, since all disks in the array may be involved in serving a request, they are assumed to be all synchronized. Some discussions on asynchronous RAID 1 and RAID 5 can be found in a previous study [8].

5.1 Variations of the RAID 1 Architecture

In this subsection, we first present accurate analytic models for the chained declustering architecture, and then conduct a study of the performance of variations of RAID 1.

5.1.1 An Analysis of Chained Declustering

As described in Section 2, there are two ways to layout data under chained declustering, *horizontal layout (HL)* and *vertical layout (VL)*.

Horizontal Layout: The *horizontal layout* is favored for write operations, since the corresponding data belonging to the same stripe in the two copies (e.g., F_1 and f_8 in Figure 2 and 3) are allocated at the same cylinder. Therefore, a *large* write can update the two copies by moving the arm to the destination cylinder, updating the first copy, followed by a rotational delay and then updating the second copy. Hence, only one seek is needed. The disk service times for reads and writes are

$$\begin{aligned} Y_r &= X + T, \\ Y_w &= X + T + R + T, \end{aligned}$$

where T is the transfer time which depends on the number of blocks to be transferred from each disk.

Thus, this system can be modeled as a $M/G/1$ queue. The service time is

$$Y = \theta Y_r + (1-\theta) Y_w$$

with moments

$$\begin{aligned} \overline{Y} &= \overline{X} + \overline{T} + p_w(\overline{R} + \overline{T}), \\ \overline{Y^2} &= \overline{X^2} + 2\overline{X}\,\overline{T} + \overline{T^2} + p_w\left(3\overline{T^2} + \overline{R^2} + 2\overline{X}\,\overline{R} + 2\overline{X}\,\overline{T} + 2\overline{R}\,\overline{T}\right). \end{aligned}$$

By the Pollaczek-Khinchin formula [19], the mean waiting time in the queue is

$$\overline{Q} = \frac{\lambda \overline{Y^2}}{2(1-\rho)}$$

where $\rho = \lambda \overline{Y}$ is the device utilization.

Finally, the mean I/O response time, $\overline{Z}$, is given by

$$\begin{aligned} \overline{Z_r} &= \overline{Q} + \overline{Y_r}, \\ \overline{Z_w} &= \overline{Q} + \overline{Y_w}, \\ \overline{Z} &= p_r \overline{Z_r} + p_w \overline{Z_w}, \end{aligned}$$

where $\overline{Z_r}$ and $\overline{Z_w}$ are the mean response times for read and write requests, respectively.

Vertical Layout: The *vertical layout (VL)* strategy favors read requests, since the two copies of data are allocated on different cylinders with a space of $C/2$ tracks between copies, where C is the total number of cylinders on each device. Thus, a read can go to the copy closest to the current arm position. In any case, a read never seeks more than $C/2$ tracks. While the VL strategy reduces the read service time, it increases the overhead for write operations. When a write operation starts, the arm moves to the copy closest to the current arm position, updates the copy, and then moves $C/2$ tracks to update the other copy. Thus, the read and write service times are

$$\begin{aligned} Y_r &= X + T, \\ Y_w &= X + T + S_c + R + T, \end{aligned}$$

where X is the sum of seek time and rotational latency and S_c is the seek time of $C/2$ tracks.

In a similar way, we can model the system by a $M/G/1$ queue and calculate the moments for overall service time by

$$\begin{aligned} \overline{Y} &= \overline{X} + \overline{T} + p_w(\overline{R} + \overline{T} + S_c), \\ \overline{Y^2} &= \overline{X^2} + 2\overline{X}\,\overline{T} + \overline{T^2} + p_w\Big(3\overline{T^2} + \overline{R^2} + 2\overline{X}\,\overline{R} + 2\overline{X}\,\overline{T} + 4\overline{R}\,\overline{T} + S_c^2 \\ &\quad +2\overline{X}S_c + 2\overline{R}S_c + 4\overline{T}S_c\Big). \end{aligned}$$

The moments $\overline{X}$ and $\overline{X^2}$ are obtained from Equation (11), where the calculations for the first two moments of the shortest seek time S are given in Appendix A.

5.1.2 Performance Results

The performance of the three variations of RAID 1 is reported here by using the accurate models for chained declustering above and simulations for mirrored and group-rotate declustering. The number of disks in the array is assumed to be $N = 64$. If a workstation disk has a capacity of 500M bytes, a disk array consisting of 64 such disks can yield 32G bytes of raw capacity, which provides ample capacity to replace mainframe disks.

Each request is assumed to access N_s stripes plus a partial stripe tail, i.e., the request size is $B = N_sW + T_s$ blocks, where W is the stripe width. In this study, we assume that N_s has a geometric distribution with parameter q, and T_s has the following distribution,

$$P\{T_s = i\} = \begin{cases} p_f, & i = 0, \\ (1-p_f)\frac{1}{W-1}, & i = 1, \cdots, W-1, \end{cases} \tag{16}$$

where p_f is the probability that a request is to perform a full stripe I/O. In other words, given a request contains a partial stripe tail, the size of the tail is uniformly distributed in a stripe. The mean request size $\overline{B}$ is assumed to be 256 blocks (1M bytes). Thus the moments for N_s are

$$\begin{aligned}\overline{N_s} &= \frac{\overline{B}-\overline{T_s}}{W}, \\ \overline{N_s^2} &= \frac{2-q}{q^2},\end{aligned}$$

where $\overline{T_s} = (1-p_f)W/2$ and $q = 1/\overline{N_s}$.

Hence, the moments for the transfer time on each disk, T, are

$$\begin{aligned}\overline{T} &= \left(\overline{N_s}+1-p_f\right)\tau, \\ \overline{T^2} &= \left[\overline{N_s^2}+(1-p_f)(2\overline{N_s}+1)\right]\tau^2. \qquad (17)\end{aligned}$$

We have studied the effect that p_f has on perfomance and have observed that the mean response time is insensitive to changes in p_f in the case of RAID 1 (results not shown here) but that this is not true for RAID 5 (results shown in the next subsection). The results shown in Figure 14 are obtained for $p_f = 0.5$.

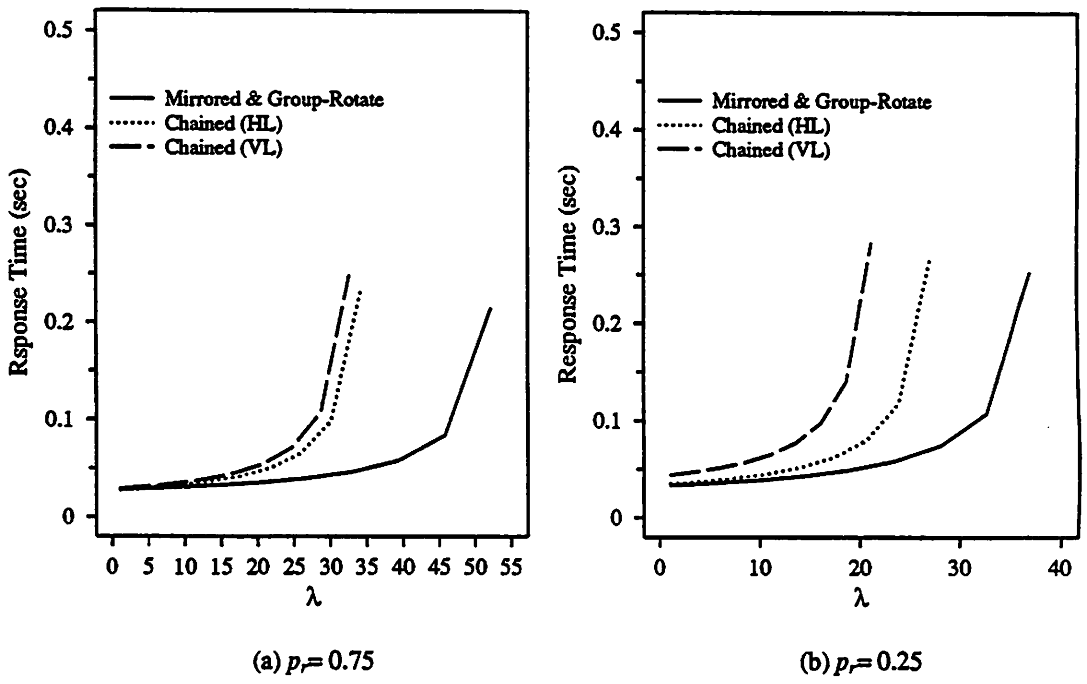


Figure 14: RAID 1 and Its Variants in Large I/O Environments.

Figure 14 plots the mean I/O response time as a function of workload for read probabilities $p_r = 0.75$ and $p_r = 0.25$. ¿From the figure, we observe that mirrored declustering and group-rotate declustering perform much better than chained declustering, especially when most of the requests are reads. This is because they can serve two read operations concurrently, whereas chained declustering can only serve one request at a time. Between the two data layout strategies of chained declustering, HL is observed to perform better than VL because of the high cost for writes under VL. However, as the read probability, p_r, increases, the performance of VL approaches that of HL, and we have observed that when $p_r > 0.85$, VL outperforms HL.

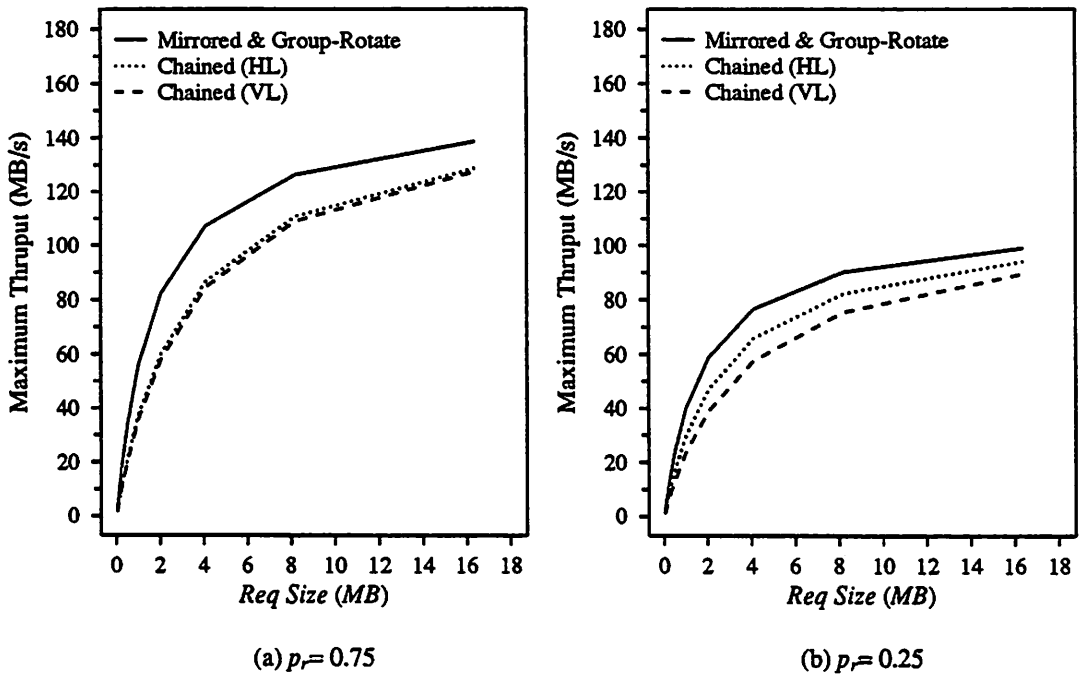


Figure 15: Maximum Throughput of RAID 1 as Function of Request Size.

In Figure 15 and 16, we illustrate the maximum throughput supported by the three architectures as a function of the mean request size and the number of disks in the array, respectively. Again, mirrored declustering and group-rotate declustering provide higher throughput than chained declustering.

5.2 RAID 1 vs. RAID 5 in Large I/O Environments

In the following, we briefly present an analytic model for synchronized RAID 5 and then compare it with RAID 1 for the cases of the two arrays having (1) identical capacity; and (2) the same number of disks, as we did in the small I/O context.

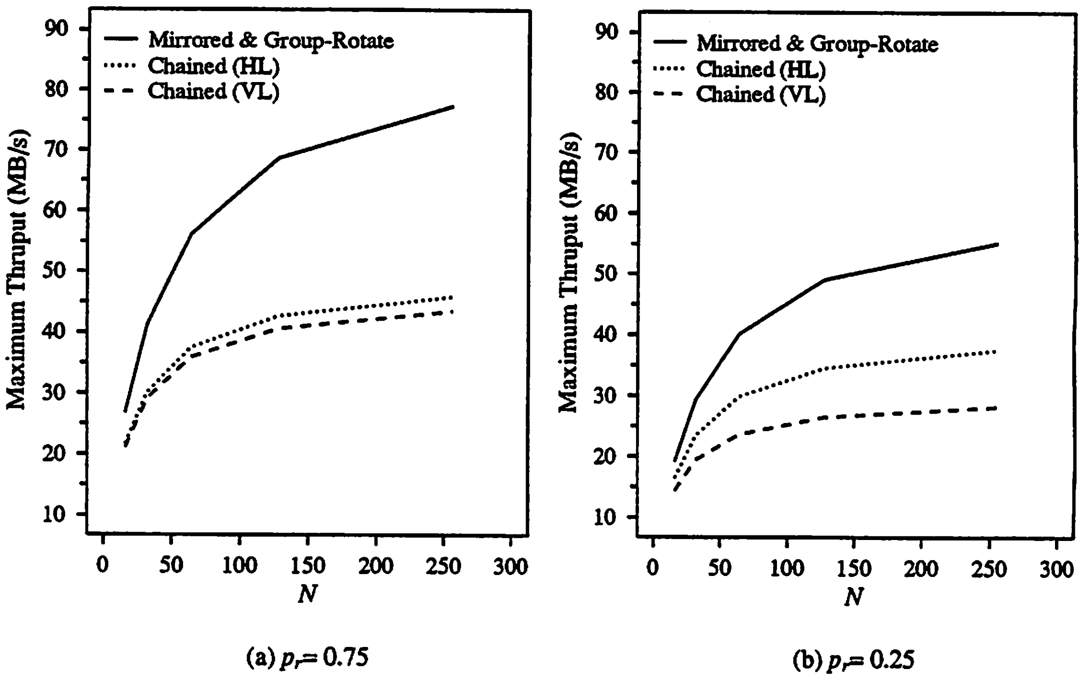


Figure 16: Maximum throughput of RAID 1 as Function of Number of Disks.

5.2.1 A RAID 5 Model for Large I/O Environments

Since we have assumed that all of the disks in the array are synchronized, the service time for a read request is $Y_r = X + T$. For write requests, since a large I/O operation may involve several stripes of data, we distinguish between two cases. In the case of a *full stripe write*, where the data blocks to be updated start and end at stripe boundaries, the write service time is the same as that of a read because the parity block of each stripe can be calculated from the new data blocks. In this case, no extra read operation is needed. However, if either the starting block or the end block is not aligned with the stripe boundary, a *partial stripe write* occurs and a *read-update* procedure has to be followed. For simplicity, we assume that a partial stripe write is only required for the last stripe of each I/O, i.e., the starting block is always aligned with the stripe boundary [5]. This may yield an optimistic estimate of the performance of RAID 5.

Let p_f be the probability that a write request is a full stripe write. For a partial stripe write, since it accesses multiple blocks from each disk, after reading out a block from the last stripe (for the purpose of a new parity block calculation) the disk will have rotated part way to the beginning of the N_s data blocks. Therefore, instead of a full rotation, it only needs to rotate the remaining distance prior to writing back the new data. Recall that N_b denotes the number of blocks in each track. The rotation time is

$$R = \left(1 - \frac{(N_s - 1) \bmod N_b}{N_b}\right) R_{max}$$

with first and second moments

$$\begin{aligned}\overline{R} &= (1-\eta)R_{max},\\ \overline{R^2} &= \left(1 - 2\eta + \frac{1}{N_b^2}E\left[((N_s-1) \bmod N_b)^2\right]\right) R_{max}^2,\end{aligned}$$

where

$$\begin{aligned}\eta &= E\left[\frac{(N_s-1) \bmod N_b}{N_b}\right],\\ &= \frac{1}{N_b}\sum_{k=0}^{N_b-1} k \sum_{m=0}^{\infty} P\{N_s = mN_b + k + 1\},\end{aligned}$$

and

$$E\left[((N_s-1) \bmod N_b)^2\right] = \sum_{k=0}^{N_b-1} k^2 \sum_{m=0}^{\infty} P\{N_s = mN_b + k + 1\}.$$

Thus, the service time for a partial stripe write is

$$Y_w = X + \tau + R + T.$$

The moments for the overall service times are

$$\begin{aligned}\overline{Y} &= p_r\overline{Y_r} + p_w p_f \overline{Y_r} + p_w(1-p_f)\overline{Y_w},\\ &= \overline{X} + \overline{T} + p_w(1-p_f)(\tau + \overline{R}),\\ \overline{Y^2} &= \overline{X^2} + 2\overline{X}\,\overline{T} + \overline{T^2} + p_w(1-p_f)\left(\overline{R^2} + \tau^2 + 2\overline{X}\,\overline{R} + 2\overline{R}\,\overline{T} + 2\overline{X}\tau + 2\overline{R}\tau + 2\overline{T}\tau\right).\end{aligned}$$

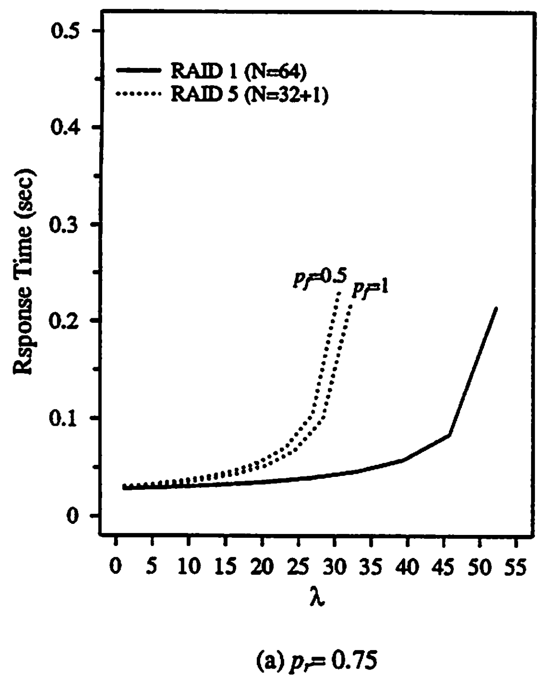
where the moments of transfer time T are obtained from Equation (17).

RAID 5 can now be modeled as a $M/G/1$ queue.

5.2.2 Comparison of RAID 1 and RAID 5

In this subsection, we compare the performances of RAID 1 and RAID 5 in a large I/O environment, in which the I/O request size is assumed to have the same distribution as described in Section 5.1.2, with a mean of 1M bytes.

First we compare RAID 1 and RAID 5 with the same capacity, i.e., the RAID 1 system contains $2n$ disks, whereas the RAID 5 system contains $n+1$ disks (with $n = 32$). The



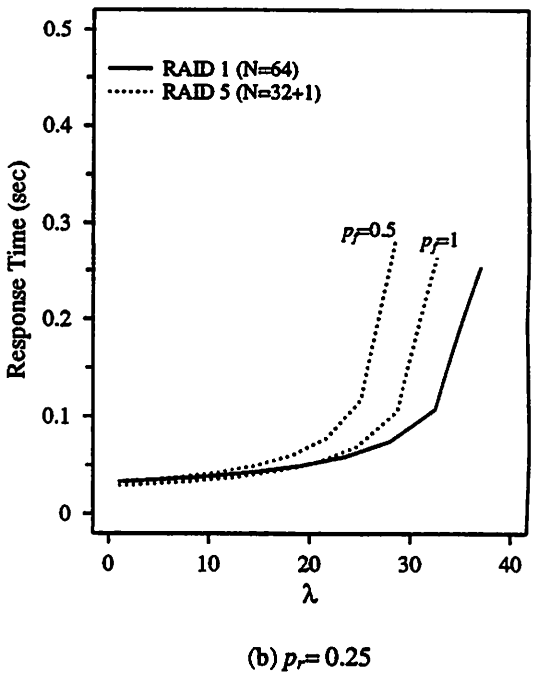
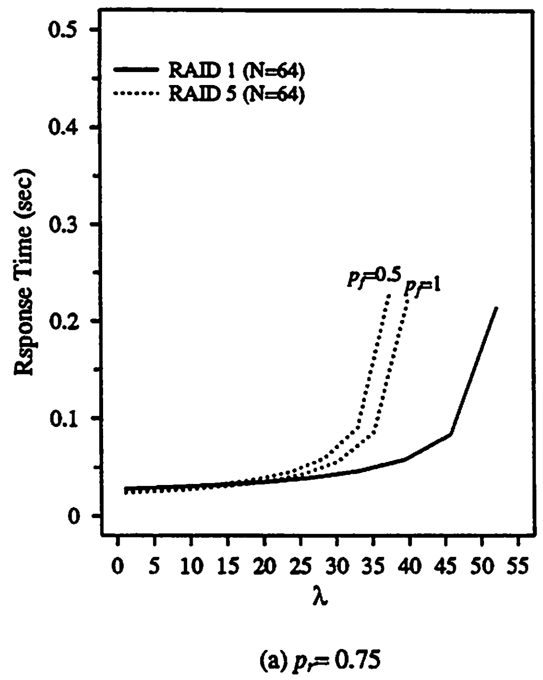


Figure 17: Comparison of RAID 1 and RAID 5 in Large I/O Environments (*Same Capacity*).



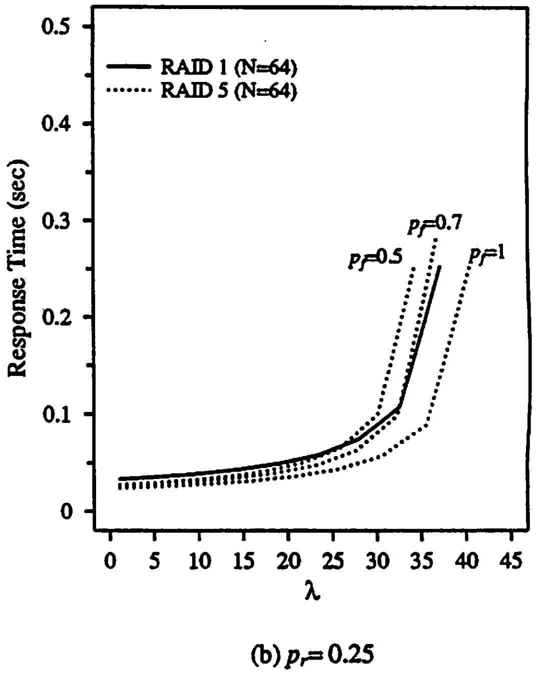


Figure 18: Comparison of RAID 1 and RAID 5 in Large I/O Environments (*Same No. of Disks*).

results are shown in Figure 17 for different values of the p_f, and RAID 1 is observed to be much better than RAID 5 in all cases. Figure 18 illustrates the situation in which both RAID 1 and RAID 5 contain the same number of disks. We observe from Figure 18 (a) that RAID 1 outperforms RAID 5 when most of the requests are reads. This is because RAID 1 can serve two reads simultaneously, while RAID 5 has to serve read requests sequentially. Although the transfer time under RAID 5 is only half of that under RAID 1, the reduction in queueing delay caused by concurrently serving multiple reads under RAID 1 outweighs the disadvantage of the longer transfer time. When most requests are writes, RAID 1 loses its advantage of being able to perform concurrent reads. Hence, RAID 5 may exhibit better performance because of its shorter transfer time. However, as shown in Figure 18 (b), the performance of RAID 5 degrades rapidly as the full stripe probability p_f decreases because of the additional disk service time required for the read-update procedure of partial stripe writes. When 30% of writes are to a partial stripe, RAID 1 is essentially equivalent to RAID 5 at high loads; and when half of the writes are to a partial stripe, RAID 1 is observed to outperform RAID 5.

In the above discussions, we assumed that most of the I/O requests to RAID 5 perform full stripe I/O operations ($p_f > 0.5$). This requires the file system or cache manager to be aware of the stripe width of the underling disk arrays. Whenever the number of disks in the array increases or decreases, the file systems or cache manager has to adapt to the change in order to maintain high performance.

6 Simulation Validations

In the previous sections, we have developed a priority queueing model for RAID 5 supporting transaction and workstation environments, $M/G/1$ models for chained declustering and RAID 5 supporting scientific and image processing systems. In order to validate the performance predicted by these models, we developed simulators for these disk array architectures. The distributions of seek times and rotational latencies are taken from Section 2.3. The simulations are conducted in the same way as those reported in Section 4.1. As a consequence, the analytical results match the simulation results very well. The estimate of the mean response times given by these queueing models lies within 2.7% of those given by simulation In the case of low and moderate loads (device utilizations less than 80%), the differences between the analysis and simulation results are less than 1.4%. Therefore, our analytic models provide accurate predictions on the performance of these disk array architectures.

7 Summary

The main contributions of this paper are the study of the design issues, development of mathematical models, and examination of the performance of different disk array architectures, both in small I/O and large I/O environments.

In particular, we studied existing disk array architectures and proposed a new architecture, called *group-rotate declustering*, which is a variation of RAID 1. We also presented scheduling policies suitable for different disk array architectures. As a result, our proposed architecture provides the best performance both in small and large I/O environments. Among the three scheduling policies for RAID 1 and its variants studied in this paper, both the shortest queue policy SQ and the minimum seek policy MS are good choices. But when the two data copies are stored on distant sites, the random join policy RJ is the choice. While the first-come-first-serve discipline is assumed above, one can easily adopt other algorithms, such as SCAN or SSTF (*shortest seek time first*) within each queue under the SQ and RJ policies. However, it is not clear how the MS policy should be coupled with disk scheduling policies such as shortest seek time first (SSTF) and SCAN.

Last, from the above studies, we can see that the main impact on the performance of RAID 5 is the high cost of the *partial stripe write.* Researchers have looked at different ways to reduce this cost. The most attractive way is the *Log-Structured File System* [32, 10, 38, 39], in which many small writes are accumulated in cache and converted into a large full stripe write. Another way is called *floating parity* [24], where parity blocks are clustered into cylinders each containing a spare track. A new parity block, instead of being written in place, is written on the nearest unallocated block following the old parity block. The cost paid is that the controller has to keep a map of the locations of all parity blocks.

A Seek Time for the Vertical Data Layout in Chained Declustering

In this Appendix, we obtain the seek distance distribution and the first two moments of seek times for the *vertical data layout* (VL) strategy in chained declustering (see Section 2.1 and 5.1.1).

Under the VL strategy, one copy of data is allocated to cylinders 0 to $C/2-1$, and the other copy to cylinders $C/2$ to $C-1$. Thus if a stripe is located on cylinder j, then its duplicate is on cylinder $(j + C/2) \bmod C$. The arm always moves to the copy closest to the current arm position, and the seek distance never exceeds $C/2$. Let A be a *r.v.* denoting the current arm position, and I be a *r.v.* denoting the target cylinder for a request (which corresponds to two physical cylinders), $I = 0, 1, \cdots, C/2 - 1$. We first obtain the distribution of A, $P\{A = i\} = a_i$. The status of the arm position can be modeled by a Markov chain $n(t)$, $n(t) = 0, 1, \cdots, C - 1$. Following the assumptions made in Section 2.3 that the arm needs to move to serve a request with probability $1 - p_s$, and the arm moves to any other $C/2 - 1$ cylinders with equal probability, we have the following equations describing the stationary distribution of the Markov chain,

$$a_i = \begin{cases} \frac{2}{C-2} \sum_{j=0, j\neq i}^{\lceil C/4\rceil - 1 + i} a_j, & i = 0, \ldots, \lceil \frac{C}{4} \rceil - 1, \\ \frac{2}{C-2} \sum_{j=0, j\neq i}^{\lfloor C/4\rfloor + i} a_j, & i = \lceil \frac{C}{4} \rceil, \ldots, \frac{C}{2} - 1, \\ \frac{2}{C-2} \sum_{j=i-\lfloor C/4\rfloor, j\neq i}^{C-1} a_j, & i = \frac{C}{2}, \ldots, \frac{C}{2} + \lceil \frac{C}{4}, \rceil - 1; \\ \frac{2}{C-2} \sum_{j=i-\lceil C/4\rceil + 1, j\neq i}^{C-1} a_j, & i = \frac{C}{2} + \lceil \frac{C}{4} \rceil, \ldots, C - 1, \end{cases}$$

$$\sum_{i=0}^{C-1} a_i = 1.$$

Solving these linear equations yields the distribution of A.

We now obtain the distribution of the seek distance. For ease of discussion, we will assume that C is a multiple of 4. Again, following the notation introduced in Section 2.3, the distribution for seek distance is $P\{D = 0\} = p_s$, and for $i > 1$

$$\begin{aligned} P\{D = i\} &= P\{D = i | V = 1\}(1 - p_s) + P\{D = i | V = 0\} p_s, \\ &= \begin{cases} (1 - p_s) \sum_{j=0}^{C-1} a_j \left[P\{I = (j - i) \bmod \frac{C}{2} | A = j, V = 1\} \right. \\ \left. + P\{I = (j + i) \bmod \frac{C}{2} | A = j, V = 1\} \right], & i = 1, \ldots, \frac{C}{4} - 1, \\ (1 - p_s) \sum_{j=0}^{C-1} a_j P\{I = (j + i) \bmod \frac{C}{2} | A = j, V = 1\}, & i = \frac{C}{4}, \\ (1 - p_s) \left[\sum_{j=0}^{C/2 - i - 1} a_j P\{I = j + i | A = j, V = 1\} \right. \\ \left. + \sum_{C/2+i}^{C-1} a_j P\{I = (j - i) \bmod \frac{C}{2} | A = j, V = 1\} \right], & i = \frac{C}{4} + 1, \ldots, \frac{C}{2} - 1, \end{cases} \end{aligned}$$

where V is a *r.v.* denoting whether an access is sequential ($V = 0$) or not ($V = 1$). Since we assumed that the target cylinder can be one of the $C/2 - 1$ cylinders with same probability, and is independent of the arm position, we have

$$P\{D = i\} = \begin{cases} (1 - p_s) \frac{2}{C-2} \left[\sum_{j=i}^{C-1} a_j + \sum_{j=0}^{C-i-1} a_j \right], & i = 1, \ldots, \frac{C}{4} - 1, \\ (1 - p_s) \frac{2}{C-2} \sum_{j=0}^{C-1} a_j, & i = \frac{C}{4}, \\ (1 - p_s) \frac{2}{C-2} \left[\sum_{j=0}^{C/2 - i - 1} a_j + \sum_{j=C/2+i}^{C-1} a_j \right], & i = \frac{C}{4} + 1, \ldots, \frac{C}{2} - 1. \end{cases} \quad (18)$$

Finally, in a similar way as in equation (4), the mean seek time is

$$\overline{S} = \sum_{i=1}^{\frac{C}{2}-1} P\{D = i\}(a + b\sqrt{i})$$

and the second moment is

$$\overline{S^2} = \sum_{i=1}^{\frac{C}{2}-1} P\{D = i\}(a + b\sqrt{i})^2.$$

Acknowledgments:

The authors gratefully acknowledge James F. Kurose for his contributions to this work, especially for the discussions on the ideas of this study. Thanks are also due to anonymous referees for their constructive comments which greatly helped to improve the quality of this paper.

References

[1] Bell, C. G. The mini and micro industries. *IEEE Comp. Mag.*, 17:14–30, October 1984.

[2] Bitton, D. and Gray, J. Disk shadowing. In *Proc. 14th VLDB Conf.*, Los Angeles, CA, August 1988.

[3] Bitton, D. Arm scheduling in shadowed disks. In *Proc. IEEE Spring ComCon*, pages 132–136, February 1989.

[4] Bultman, D. L. High performance SCSI using parallel drive technology. In *Proc. BUS-CON Conf.*, Anaheim, CA, February 1988.

[5] Chen, P., Gibson, G., Katz, R. H., and Patterson, D. A. An evaluation of redundant arrays of disks using an Amdahl 5890. *Perf. Eval. Rev.*, 18(1):74–85, May 1990.

[6] Chen, S.-Z. *Design, Modeling, and Evaluation of High Performance I/O Subsystems.* PhD thesis, University of Massachusetts at Amherst, September 1992.

[7] Chen, S.-Z. and Towsley, D. Raid 5 vs. parity striping: Their design and evaluation. Technical Report 92-31, COINS Dept. U. Massachusetts, Feb. 1992. To appear in Journal of Parallel and Distributed Computing.

[8] Chen, S.-Z. and Towsley, D. A queueing analysis of RAID architectures. Technical Report 91-71, COINS Dept. U. Massachusetts, May 1991.

[9] Copeland, G. and Keller, T. A comparison of high-availability media algorithms. In *Proc. ACM SIGMOD Conf.*, Portland, OR, May 1989.

[10] Douglis, F. and Ousterhout, J. Log-structured file systems. In *Proc. IEEE Spring ComCon.*, pages 124–129, February 1989.

[11] Gray, J., Host, B., and Walker, M. Parity striping of disk arrays: Low-cost reliable storage with acceptable throughput. In *Proc. 16th VLDB Conf.*, pages 148–161, Brisbane, Australia, 1990.

[12] Harker, J. M. et al. A quater century of disk file innovation. *IBM J. Res. Dev.*, 25:677–689, September 1981.

[13] Hsiao, H. and DeWitt, D. Chained declustering: A new availibility strategy for multiprocessor database machines. In *Proc. 6th Int'l Conf. on Data Engineering*, Los Angeles, CA, Feb. 1990.

[14] Joy, W. Presentation at ISSCC'85 Panel Session, February 1985.

[15] Katz, R. H., Gibson, G., and Patterson, D. A. Disk system architectures for high performance computing. *Proc. of the IEEE*, 77(12):1842–1858, December 1989.

[16] Khinchine, A. Y. *Mathematical Methods in the Theory of Queueing.* Hafner Publishing Co., New York, 1960.

[17] Kim, M. Y. Synchronized disk interleaving. *IEEE Trans. on Comp.*, C-35 (11):978–988, Nov. 1986.

[18] Kim, M. Y. and Tantawi, A. N. Asynchronized disk interleaving: Approximating access delays. *IEEE Trans. on Comp.*, C-40 (7):801–810, July 1991.

[19] Kleinrock, L. *Queueing Systems*, volume 1. John Wiley, New York, 1975.

[20] Kleinrock, L. *Queueing Systems*, volume 2. John Wiley, New York, 1976.

[21] Lee, E. K. and Katz, R. H. Performance consequences of parity placement in disk arrays. In *Proc. 4th Int'l Conf. on Archi. Sup. for Prog. Lang. and OS*, pages 190–199, April 1991.

[22] Livny, M., Khoshafian, S., and Boral, H. Multi-disk management algorithm. In *Proc. SIGMETRICS Conf.*, pages 69–77, May 1987.

[23] Lynch, W. C. Do disk arms move? *Perform. Eval. Rev.*, 1:3–16, December 1972.

[24] Menon, J. and Kasson, J. Methods for improved update performance of disk arrays. Technical Report RJ 6928 (66034), IBM Almaden Research Center, Nov. 1990.

[25] Menon, J., Mattson, D., and Ng, S. Performance of disk arrays in transaction processing environments. Technical Report RJ 8230 (75424), IBM Almaden Research Center, July 1991.

[26] Merchant, A. and Yu, P. S. Modeling and comparisons of striping strategies in data replication architectures. Technical report, IBM Thomas J. Watson Research Center, Sept. 1991.

[27] Muntz, R. R. and Lui, J. C. Performance analysis of disk arrays under failure. In *Proc. 16th VLDB Conf.*, pages 162–173, Brisbane, Australia, 1990.

[28] Nelson, M., Welch, B., and Ousterhout, J. Caching in the Sprite network file system. *ACM Trans. on Comp. Sys.*, 6(1):134–154, February 1988.

[29] Ng, S. Some design issues of disk array. In *Proc. IEEE Spring ComCon*, pages 137–142, 1989.

[30] Ng, S., Lang, D., and Selinger, R. Trade-offs between devices and paths in achieving disk interleaving. In *Proc. 15th Intern. Sym. on Comp. Archit.*, Hawaii, May 1988.

[31] Ogata, M. and Flynn, M. A queueing analysis for disk array systems. Technical Report CSL-TR-90-443, Stanford Univ., August 1990.

[32] Ousterhout, J. K. and Douglis, F. Beating the I/O bottleneck: A case for log-structured file systems. *Operating System Rev.*, 23(1):11–28, Jan. 1989.

[33] Ousterhout, J. K. et al. A trace-driven analysis of the UNIX 4.2 BSD file system. In *Proc. 10th Sym. on Operating Systems Principles*, pages 15–24, Dec. 1985.

[34] Patterson, D. A., Chen, P., Gibson, G., and Katz, R. H. Introduction to redundant arrays of inexpensive disks (RAID). In *Proc. IEEE Spring ComCon.*, pages 112–117, 1989.

[35] Patterson, D. A., Gibson, G., and Katz, R. H. A case for redundant arrays of inexpensive disks (RAID). In *Proc. ACM SIGMOD Conf.*, pages 109–116, July 1988.

[36] Poston, A. A high performance file system for UNIX. In *Proc. USENIX*, pages 215–226, September 1988.

[37] Reddy, A. L. N. and Banerjee, P. An evaluation of multiple-disk I/O systems. *IEEE Trans. Comp.*, 38(12), December 1989.

[38] Rosenblum, M. and Ousterhout, J. K. The LFS storage manager. In *Proc. Summer '90 USENIX Tech. Conf.*, pages 315–324, Anaheim, CA, June 1990.

[39] Rosenblum, M. and Ousterhout, J. K. The design and implementation of a log-structured fils system. Draft, March 1991.

[40] Salem, K. and Garcia-Molina, H. Disk striping. In *Proc. IEEE Data Engineering Conf.*, Los Angeles, CA, February 1986.

[41] Schroeder, M., Gifford, D., and Needham, R. A caching file system for a programmer's workstation. In *Proc. 10th Sym. on Operating System Principles*, pages 25–34, December 1985.

[42] Schulze, M., Gibson, G., Katz, R., and Patterson, D. How reliable is a RAID? In *Proc. IEEE Spring ComCon Conf.*, San Francisco, CA, February 1989.

[43] Scranton, R. A. and Thompson, D. A. The access time myth. Technical Report RC 10197, IBM, Sept. 1983.

[44] Stonebraker, M. and Schloss, G. A. Distributed RAID – a new multiple copy algorithm. In *Proc. 6th Int'l. Conf. on Data Engineering*, February 1990.

[45] Teradata Corp. *DBC/1012 Database computer system manual release 2.0*, Nov. 1985.